〔非公開〕

ＴＲ－Ｃ－０１４４

手振りと言語による仮想物体形状生成インタフェース

吉田　美寸夫　　ジュリ　ティヘリノ
Mikio YOSHIDA　　Yuri A. Tijerino

宮里　勉　　岸野　文郎
Tsutomu MIYASATO　　Fumio KISHINO

１９９６　３．１５

ＡＴＲ通信システム研究所

手振りと言語による
仮想物体形状生成インタフェース

吉田 美寸夫† ジュリ A. ティヘリノ† 宮里 勉† 岸野 文郎†

†エイ・ティ・アール通信システム研究所

平成8年3月15日

目 次

あらまし

本報告では，物体の形状を表現する手振りと言語を用いた3次元形状入力のための直観的なインタフェースシステムを提案する．コンピュータの3次元形状設計の作業は現在，2次元のポインティングデバイスの使用が中心であり，直観的な操作性に乏しい．したがって，3次元仮想作業空間において直接的，直観的な作業が期待されている．そこで，3次元位置センサと手形状入力デバイスで計測した手振りの軌跡に対して，超2次関数で表わされる曲面を近似することで3次元形状を推定するとともに，ドメイン依存な形状に関する知識により，言語情報を形状生成に反映させるシステムを提案する．また，本方式を用いた簡易3次元形状入力システムを構築し，その有効性を検討する．

1 はじめに

筆者らは，臨場感通信会議システム(図1)[1]を提案し，その要素技術の研究を行っている．このシステムは，遠隔地に離れた複数の会議参加者どうしが，コンピュータグラフィックスで合成された会議室であたかも一堂に会しているような環境を提供するものである．さらに，仮想物体を遠隔地にいる会議参加者と相互に操作することなどにより，高度な協調作業を可能とする．また，会議参加者の様子は3次元コンピュータグラフィックスで合成されることから，ブレーンストーミングなどで有効とされ，このシステムを発想の場として利用することが期待される[2]．このような創造的な活動の場としての優れた機能を備えている環境を，例えば生産物のデザイン設計などの検討の場として利用することを考えてみると，会議参加者が抱いている漠然とした形状に関するイメージを仮想物体として可視化し，それを更にインタラクティブに編集することにより，より具体化していくことが期待できる．このような状況では，あらゆる会議参加者がモデリングのためのシステムの使用法に通じていることは実際的でなく，少数のオペレータを通じてイメージを可視化するなどの手段がとられる．この可視化の過程に計算機を介在させ，企画立案を行う人間に対して仮想物体による表現能力を与えることができれば，このプロセスの効率化に貢献することができる．

図 1: 臨場感通信会議システムにおける協調作業

従来から，我々が日常使用している言語や手振りなどの対話手段をコンピュータとの対話に利用する研究が行われてきており，言語的な表現と組合された手振りによって表現される

操作を，仮想物体の指示，移動，回転や拡大縮小などの操作に反映するためのインタフェースが開発されている [3, 4, 5, 6, 7].

それらの中には，3次元位置センサや手形状入力装置などの入力デバイスを用い，仮想物体を直接的に操作するインタフェースを備えたシステムがある．杉浦ら [8] は，このようなインタフェースは手作り風のデザインに有効であると考え，3次元空間の位置を検知するスタイラスペンを使って3次元形状の変形や色付けといった加工ができるインタフェースを実現した．また，David と Ganapathy[9] らは，手形状入力デバイスと3次元位置センサ，音声認識システムを統合し，仮想空間で自由曲面を設計できるインタフェースシステムを実現した．これらのシステムは，形状を設計するために，基本形状を変形していくことが必要とされるという点で非効率である．

このような問題に関連して，3次元形状設計の初期段階を対象としたCADシステムの研究が行われており，Pentland[10] は SuperSketch と呼ばれる CAD ツールを開発した．このツールは形状を粘土のように変形させながら意図する形状を設計できるだけでなく，3次元形状の2次元投影断面をドローイングすることで直接意図する3次元形状を生成することができる．同様な投影断面入力方式のインタフェースを持つシステムである 3-Draw[11] はより複雑な形状の設計を可能にした．ユーザは3次元位置センサを装着した手帳サイズのボードとスタイラスペンにより，3次元空間中の任意の位置と向きで断面形状を描くことができる．スケッチは正確に形状の特徴を入力さえできれば，特に入力したい形状が粘土などで既に作られている場合，意図に近い形状が生成できるので有効である．しかし，我々にとって馴染みの深い形状を表現する場合，わざわざスケッチで入力しなければならないのは，やはりわずらわしく効率的ではない．

そこで，筆者らは，自然言語や手振りを用いて，会議参加者が抱いているイメージを3次元仮想物体として可視化する手法の提案を行ってきた [5, 12, 13, 7]．本報告では，軸対称な3次元物体に対象を限定し，入力される言語的情報と手振りの軌跡からその物体の輪郭形状を推定し，仮想物体として 生成することができるプロトタイプシステムを試作したので，報告をする．

2 形状を表現する言語と手振りの意味

筆者らは，臨場感通信会議システムを総合評価するため，3地点間を接続した実験システムを構築した [14]．このシステムでは，遠隔地に離れた3人が，仮想物体として表示された神輿のモデルの組み立てを行った．当初，システムは仮想物体を言語と手振りによりハンドリングすることができたが，物体をつかんで操作する以外の認識可能な手振りは仮想物体のポインティング指示操作のみで，仮想物体の生成，編集には不十分であった．筆者らはこのシステムに対して，メンタルイメージの可視化手段としての高度な3次元形状の生成・編集インタフェースが必要であると考える．

本報告では，上記の考え方に基づき，手振りと言語による仮想物体形状生成のための一方式を提案する．

我々の日常の中で，3次元形状の特徴を話し相手に伝える目的で手振りを利用する場合として，例えば，“壷” の形状を伝えるために，手振りを交えながら，次のような表現を用いることが考えられる．

「こんな形をした丸口の壷…」

この例で，手振りは，“壷” という言語情報だけからではその形状を具体化できないため，言

2 形状を表現する言語と手振りの意味

語が表現できない情報を補完する役割を果たしていると考えられる．また，言語から，話し手が表現しようとしている形状が“壷”というドメインに属していることが分かるので，対話相手は“壷”に関する知識を用いることで，比較的正確にその形状をイメージすることができると想像できる．我々は，手振りで表現されら形状が，壷のいずれの側面の形状を表現したものであるかを容易に識別できるだろう．また，手振りを使い，次のような方法で相手のイメージしている形状を補正しようとする場合がある．

「壷の首の部分はこのくらいの太さで..」

この場合，話し手は壷の高さや，太さ等の形状に関するより詳しい特徴を段階的に付加していくことで，対話相手が抱いているイメージの漠然とした部分を明らかにしようとしている．この場面においても，対話相手は“壷”というドメインの中で形状をイメージしており，“首”という隠喩的な表現から,「壷の胴体に部分的に細い箇所が存在し，その太さが手振りで表わされた長さである」と理解でき，やはり，ドメイン依存な形状に関する知識を利用していることが想像できる．すなわち，計算機との対話においては，計算機がこのような知識を持っていることが必要であると考えられる．

以上のことから，３次元形状を意図する手振りと言語から形状生成するためには，そのような言語と手振りについて以下の点を考慮する必要がある．

1. 比較的単純な形状を表現する場合，手振りはそれほど重要ではなく，言語的な表現が主に用いられる．このとき，幾何学的な言語表現を用いるよりも，”ボール”や”棒”等の概念的な表現を利用することが多い．この場合，言語によって表現される概念的な表現を形状に変換する必要があるが，言語の意味はそれを利用する人間の育った社会的な背景やイデオロギーや文化的なコンテキスト等に依存しているとされ，その解釈は困難である．そのため，３次元形状の常識的な概念に関する知識が必要である．
2. 基本的な形状と単純な変形の組み合わせでは表現が困難な形状を表現しようとする場合，ある基本的な形状を変形させるような表現を用いて，特殊な形状を表現する場合がある．例えば,「とがった棒」や「丸みのある箱」などと表現する場合である．これらには，”とがった”や”丸み”等の曖昧な表現が含まれているが，このような言語の曖昧さを補うために，我々は手振りを利用することがあり，それにより，初めて形状を正確に理解したと実感することができる．このように，まず言語表現により基本となる形状をイメージさせ，次に手振りによりイメージされた形状を変形させるような手順を，計算機が形状に反映できる機構が必要である．
3. 基本的な形状を表わす言語表現とある部分の形状の特徴を表わす手振りを組み合わせて形状を表現する場合がある．この場合，言語で表現された形状のどの部分に対して，手振りが反映されるべきかという情報が与えられない場合もあり得，形状に対する手振りの位相的関係を自動的に識別する機構が必要である．
4. 複雑な形状や形状の特徴を細部まで伝えようとする場合，言語や手振りの表現を繰り返し用いることにより伝えようとすることがある．この場合，先に述べたような方式で伝えられた形状に対して，手振りや言語を繰り返し形状に反映させることができる機構が必要となる．

以上のことより，形状を意図する手振りと言語の解釈するには，それらを合成するような過程が重要であることが理解できる(図2)．したがって，３次元形状を表現する言語と手振りの解釈においては，

1. 形状を意図して表現される言語には，概念的な表現が存在し，しかもその頻度が高いこと

2 形状を表現する言語と手振りの意味

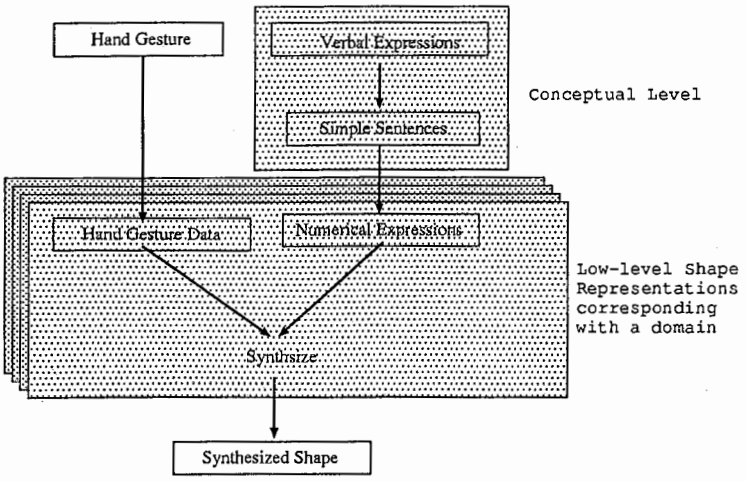

図 2: 形状を意図する言語と手振りを解釈するためのプロセスモデル

2. 形状に変形を施す表現が用いられるため，そのような変形を形状に反映できる必要があること
3. 手振りが表現する形状を既存の形状に反映できる必要があること
4. さらに，手振りが表現する形状は意図する物体の形状を包括的に表現するものではないため，どの部分の形状を表現しているかを識別する必要があること．

が問題である．

本報告では，上記問題点のうち，先の３つの問題点に対して，それぞれ以下のような解決策に基き実現したインタフェースシステムを提案する．

1. 形状に関する概念的な表現と“立方体”や“円柱”などのような幾何学的な形状の表現を翻訳するために２つのレベルに分割した知識を利用する．これは，ドメイン依存の常識的な形状概念を“概念レベルの形状に関する知識”として体系化し，ドメイン非依存の形状表現を“基本的な形状に関する知識”として体系化している．この２つのレベルの知識については文献 [12, 13, 15] を参照されたい．
2. 形状の特徴を表わす言語的表現と対応するようなパラメータを持つ形状表現方式である超２次関数を採用する．これにより，言語的表現から形状を変形することが容易に実現できることに加え，パラメータから形状の特徴を計算機が判断することが可能である．
3. 手振りを離散点集合としてデータ化し，手振りの形状への反映を形状の離散点集合への近似問題として扱うようにする．手振りデータは部分的な特徴を表現し，そのデータはノイズが大きいことから，手振りを忠実に反映させることは良い結果を生じないと予想される．そこで，言語で得られる意図する物体が属するドメインに応じ，適当なパラメータと制約条件を課す．

3 手振りから形状を生成する手法

3.1 手法概要

本章では，形状を表現する言語とそれに付随する手振りから仮想物体を生成するための具体的機構について述べる．

手振りを扱うシステムでは，手振りをどのようにデータ化するかは設計の上で重要な要素となる．先に述べたように，筆者らは手振りを離散点集合としてデータ化する(図3)が，この方式は，従来から報告されているような2種類の手振りを区別せず扱うことができるという点でも有効である．2種類の手振りとは，手振りの過程で手形状や手の位置が変化しないPostureと手振りの過程で手形状あるいは手の位置が変化するGestureである．これらの手振りの分類については，過去の研究者の報告においても概ね一致した見解が得られている[16].

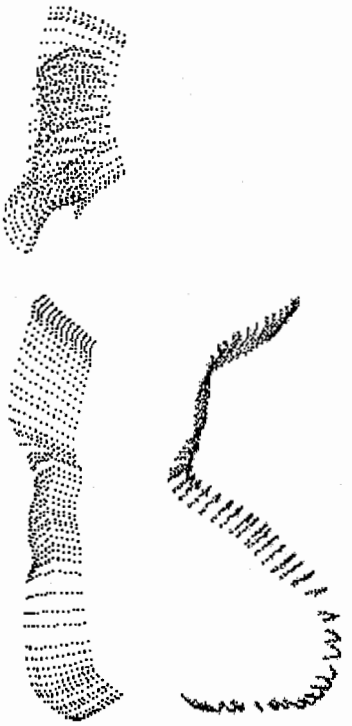

図 3: 手振りによって得られた離散点集合 (3 面図)

このようなデータから形状を構築するには，3次元デジタイザ等を用いて計測された形状表面上の点集合から形状を再構築する手法が適用できる，これまでに，このような手法に関する様々な研究が行なわれており，多くの成果が得られている．それらの中には，我々と同様な超2次関数を対象としたTerzopoulos [17]，Pentland [18] らの研究がある．

Pentlandらは，超2次関数に対して，言語と対応付けが容易な変形パラメータを作用させることで，ある程度一般的な形状を表現できると主張しており，限られた変形形態により超2次関数を3次元点に近似させる手法を提案している．彼らの手法によれば，超2次関数に対する変形形態として，ある形状特徴に対応するようなパラメータを持つような変形形態を

設けることで，特定のドメインに属する形状の特徴のバリエーションを表現することができる．そこで，そのパラメータを変化させ，手振りに近似させることにより，特定のドメインの特徴を持つ形状を手振りから推定することが可能になる．

先に，ドメイン依存な形状に関する制約を決定すると述べたが，例えば，“壷”のような形状はどのような特徴を持つか考えてみると，壷を正面から見た場合の形状は口の部分が広がっているもの，すぼまっているもの，くびれのあるものないものなど様々である．しかし，その形状の輪郭は大体３次曲線で表現できそうである．筆者らは，形状を推定する手法として，Pentland らの手法を利用することとし，超２次関数の輪郭を物体の特徴に合う形にする変形形態を設け，それらをドメインにおける形状の知識とみなすことで，言語と手振りの統合を実現することとした．以下，Pentland ら [18] の手法及びそれに基づく我々の手法を概説する．

3.2 パラメータの推定法の概要

超２次関数 [19] は式 (1) のようなベクトル $\mathbf{x} = [x, y, z]^T$ によって定義される．T は転置を表わす．

$$\begin{aligned} x &= a_1 \cos^{\varepsilon_1}\omega \cos^{\varepsilon_2}\eta \\ y &= a_2 \cos^{\varepsilon_1}\omega \sin^{\varepsilon_2}\eta \\ z &= a_3 \sin^{\varepsilon_1}\omega \end{aligned} \tag{1}$$

ここで，x, y, z は超２次関数で表わされる曲面上の点のデカルト座標，a_1, a_2, a_3はそれぞれ x, y, z 方向のスケールパラメータである．極座標 η, ωはそれぞれ経度，緯度．また，ε_1は Z-X(Y) 断面における輪郭の角張り具合を示すパラメータ，ε_2は X-Y 断面の輪郭線の角張り具合を示すパラメータである．式 (2) は式 (1) を陰関数表現で表わしたものである．

$$f(\mathbf{x}) = \left[\left(\left(\frac{x}{a_1}\right)^{2/\varepsilon_2} + \left(\frac{y}{a_2}\right)^{2/\varepsilon_2}\right)^{\varepsilon_2/\varepsilon_1} + \left(\frac{z}{a_3}\right)^{2/\varepsilon_1}\right] - 1 \tag{2}$$

曲面 $f(\mathbf{x}) = 0$ を回転，平行移動，変形したものは式 (3) で表わされる．

$$\bar{\mathbf{x}} = M\mathcal{D}_u\mathbf{x} + \mathbf{b} \tag{3}$$

ここで，M，b，$\mathcal{D}_u$は，回転マトリックス，移動ベクトル，変形マトリックスをそれぞれあらわす．$\mathcal{D}_u$は $3\mathcal{X}3$ のマトリックスで表わされ，曲面に対して大域的変形として作用し，以下のように定義される．

$$\begin{aligned}
d_{00} &= u_6 + yu_{12} + zu_{15} - (u_{13} + u_{16})\, sgn(x) - u_{14} - u_{17}, \\
d_{01} &= u_{11} + 2y\,(u_{13} + sgn(x)\, u_{14}), \\
d_{02} &= u_{10} + 2z\,(u_{16} + sgn(x)\, u_{17}), \\
d_{10} &= u_{11} + 2x\,(u_{19} + sgn(y)\, u_{20}), \\
d_{11} &= u_7 + xu_{18} + zu_{21} - (u_{19} + u_{22})\, sgn(y) - u_{20} - u_{23}, \\
d_{12} &= u_9 + 2z\,(u_{22} + sgn(y)\, u_{23}), \\
d_{20} &= u_{10} + 2x\,(u_{25} + sgn(z)\, u_{26}), \\
d_{21} &= u_9 + 2y\,(u_{28} + sgn(z)\, u_{29}), \\
d_{22} &= u_8 + xu_{24} + yu_{27} - (u_{25} + u_{28})\, sgn(z) - u_{26} - u_{29}.
\end{aligned} \tag{4}$$

上式において，パラメータ u_iは変形の形態に対応しており，u_6から u_8はスケーリング，u_9から u_{11}は X,Y,Z 軸に対するせん断変形，u_{12+3j}はテーパリング，$u_{12+3j+1}$は曲げ，$u_{12+3j+2}$は

ピンチング（つまむ）に対応している．u_0からu_5は平行移動と回転に対応し，M,b の要素となる．筆者らは，この$\mathcal{D}_u$ に新たな変形形態を追加することにより，ドメイン依存な形状の特徴を表現するようなパラメータを構築することを狙いとしているが，このような$\mathcal{D}_u$の構造から，新規にパラメータを追加することはドメインに対応するような形状表現を構築するよりも容易であることが分かる．

いま，n 個の３次元の計測点の集合

$$\overline{\mathbf{X}} = [\overline{\mathrm{x}}_1, ..., \overline{\mathrm{x}}_n], \qquad \overline{\mathrm{x}}_i \in E^3 \tag{5}$$

が与えられているものとする．もし，適当な$\mathcal{D}_u$が得られれば，変形する以前の曲面上の点を$\mathcal{D}_u$で変換すると$\overline{\mathbf{X}}_i$になるような，曲面上の点x_iが求まる．逆に，この対応する点と計測点から$\mathcal{D}_u$を求めることができる．そこで，変形する以前の超２次関数で表わされる曲面上で対応する点の集合 $\mathbf{X} = [\mathrm{x}_1, ..., \mathrm{x}_n]$ を求める ($\overline{\mathrm{x}}_i$に x_i が対応する)．その前に，まず計測点集合の重心と慣性主軸を求め，曲面が計測点の近くにくるように移動とスケーリングを施し，それらの点を$\mathbf{X}$ の候補とする．

Pentland らは，この対応点を求める方法として，計測点を楕円極座標系に投影することを提案しているが，我々が対象としたタスクである計測点が曲線を表わす場合には，計測点が投影方向に対して水平に並び，計測点の特徴が結果に反映されにくくなるという問題があった(図4)．そこで，計測点と曲面との間の距離が最も小さくなるような曲面上の点を見つけ，対応する点とすることとした．

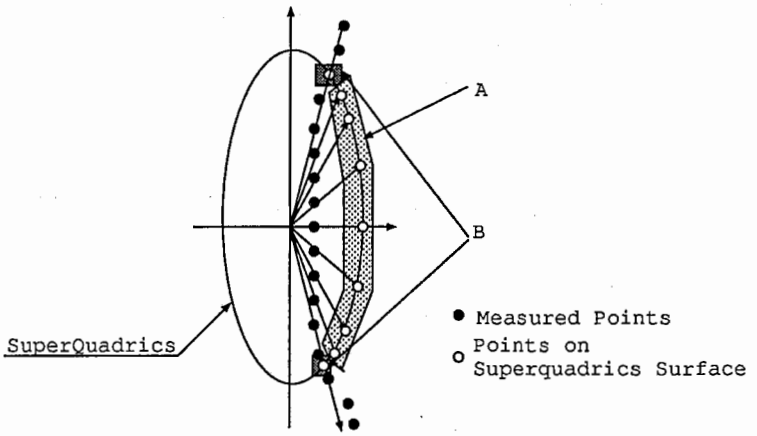

図 4: 離散点集合の楕円座標系への投影　領域 B に対応される離散点は複数の点が 1 点に投影される為，その特徴が形状に反映されない．

変形パラメータは，未変形の曲面上の点$\mathbf{X}$ をそれに対応する計測点$\overline{\mathbf{X}}$に近似するようなパラメータとして，最小自乗法により求めることができる．

$$\mathcal{D}_u\mathbf{X} \rightarrow \overline{\mathbf{X}} \tag{6}$$

3.3 ドメインによる変形パラメータの制約

前節で紹介した手法を用いて得られる形状は変形パラメータに大きく依存する．例えば，表 1 の制約を変形パラメータに課すことで，Z 軸対称な形状を必ず生成することができる．

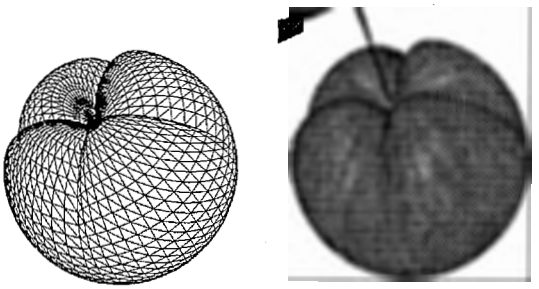

図 5: 林檎らしく変形された超 2 次関数:**左**:ワイヤ表示. **右**:テクスチャを貼りシェーディングした.

このように，ドメインに含まれる物体の形状の特徴を変形パラーメタの制約として課すこと

表 1: Z 軸に軸対称な形状を生成するためのパラメータの制約

変形形態	パラメータの制約
スケーリング	$u_6 = u_7$
せん断	$u_9 = u_{10} = u_{11} = 0$
テーパリング	$u_{12} = u_{18} = u_{24} = u_{27} = 0,$ $u_{21} = u_{15}$
曲げ	$u_{13} = u_{16} = u_{19} = u_{22} = 0,$ $u_{25} = u_{28}$
ピンチング	$u_{14} = u_{20} = u_{26} = u_{29} = 0,$ $u_{17} = u_{23}$

で，例えば，Z 方向の長さと X(Y) 方向の長さの比や，ある先細りの角度をもった形状等を生成することが可能になる．また，ドメイン特有な形状を表わす変形パラーメータを追加し，その値を定数とすることで，手振りに影響されない形状の特徴を持つ形状を生成できるようになる．例えば，林檎はその上下に独特な凹凸を持っているが，適当な変形パラメータを追加することにより，林檎らしい形状を表現できる (図 5).

4 実験システム

4.1 システムの概要

筆者らは，物体形状を表わす手振りと言語表現から仮想物体形状を生成するために，先に紹介した手法に基づき，実験システムを構築した．以下にその概要を紹介する．

筆者らは，両手を用いた動的な手振りおよび静的な手振り(手形状)を最終的な対象としているが，本システムでは，右片手による動的な手振りのみを対象として実験を行った．手振りの軌跡は，手の複数個の代表点を3次元位置センサと手形状入力装置を用いて計測して得られた3次元点の集合として記憶装置に保存される(図6)．代表点として，手の中指から手のひらの一部までのほぼ一直線上に載る13 点を選択した(図7)．現在，本システムには音声認識装置は結合されていないが，ユーザはメニューから意図する形状の特徴やドメインを入力することができる．また，手振りの開始と終了はユーザが明示的にキーボードのキーを押すことでシステムに伝えているが，将来は，音声の有無と手振りの特徴の変化から，必要な区間の手振りデータだけを採取できるようにする予定である．

対象物体は，軸対称な形状に限定し，その対称軸は仮想空間の高さ方向に平行な軸とすることとした．また，右手振りを対象としているので，手振りは形状の右片側の断面を表現しているとみなし，主軸方向の各サイズは手振りの軌跡の範囲を基準として求めることとしている．このことにより，特に回転体の半径は操作者の意図を反映できないと予想され，本システムでは推定後の形状に対して，親指と人差し指のそれぞれの先端間の距離を，壷の口，底，首（口，底以外でもっとも細い部分）の径として指定できるようにした．

図 6: 実験システムを使用している様子

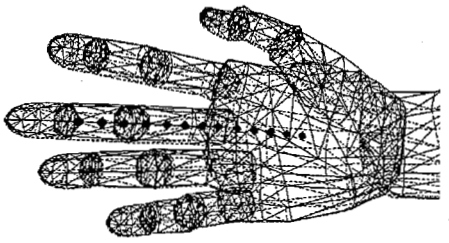

図 7: 手のモデルと手振りの軌跡計測のための13 個の代表点

4.2 実験システムの変形パラメータと制約の例

本節では，“壷”というドメインを例として，採用した変形パラメータを説明する．
先に述べたように，壷の輪郭を3次関数で表現するために，変形マトリックスのd_{00}, d_{11}にパラメータu_{30}からu_{33}を式(7)のように追加した．

$$\begin{aligned} d_{00} &= u_6 + yu_{12} + zu_{15} - (u_{13} + u_{16})\, sgn(x) - u_{14} - u_{17} \\ &\quad + z^2 u_{30} + z^3 u_{31}, \\ d_{11} &= u_7 + xu_{18} + zu_{21} - (u_{19} + u_{22})\, sgn(y) - u_{20} - u_{23} \\ &\quad + z^2 u_{32} + z^3 u_{33}. \end{aligned} \tag{7}$$

制約には，Z軸に軸対称な形状を表わすための制約(表1)と，新たに追加したパラメータに対し，式(8)を課した．

$$\begin{aligned} u_{30} &= u_{32}, \\ u_{31} &= u_{33}. \end{aligned} \tag{8}$$

4.3 拡張した超2次関数

壷の形状は上部と底が平らである．超2次関数を用いると，ε_1を0に近づけることで，ある程度このような形状を表現可能であるが，コーナにフィレットのような丸みが生じる(図8左)．$\varepsilon_1 = 0$とすると，式(1)より，$z = a_3$となり，パラメータωを変化させても，Zは変化せず面を表わさない．また，側面のポリゴン頂点は，ε_1を0に近づけるにつれコーナに集中していき，形状変形やポリゴン化等実装上の問題がある．筆者らは，比較的単純な手法として，xとyおよびzのべき乗を表わすε_1をそれぞれ独立な変数とすることにより，問題に対処した．

$$\begin{aligned} x &= a_1 \cos^{\varepsilon_{11}} \omega \cos^{\varepsilon_2} \eta \\ y &= a_2 \cos^{\varepsilon_{11}} \omega \sin^{\varepsilon_2} \eta \\ z &= a_3 \sin^{\varepsilon_{12}} \omega \end{aligned} \tag{9}$$

上式に$\varepsilon_{11} = 0.0, \varepsilon_{12} = 1.0$を代入すると，図8右のような円筒面を表わす．またこの表現方法を用いると，ε_{11}を0の変化はzに影響を及ぼさず，実装上の問題が解消できる．

$$\begin{aligned} x &= a_1 \cos^{\varepsilon_2} \eta \\ y &= a_2 \sin^{\varepsilon_2} \eta \\ z &= a_3 \sin \omega \end{aligned} \tag{10}$$

5 実験システムの使用例

実験システムを使い，手振りと言語から形状を生成する過程を説明する．
ここでは例として，操作者が手振りを交えながら，次のような表現をした場合を想定する．

「こんな形をした丸口の壷」

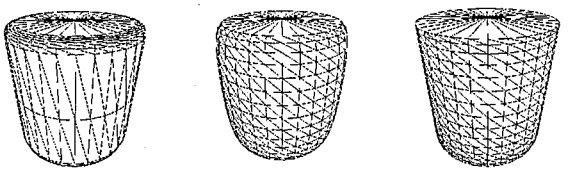

図 8: 超 2 次関数と拡張された超 2 次関数: **左**:$\varepsilon_1 = 0.1, \varepsilon_2 = 1.0$, **中央**:$\varepsilon_{11} = 0.1, \varepsilon_{12} = \varepsilon_2 = 1.0$, **右**:$\varepsilon_{11} = 0, \varepsilon_{12} = \varepsilon_2 = 1.0$.

まず，操作者がイメージしている形状のドメインは丸口の“壷”であるので，超 2 次関数で表わされる曲面の形状を円筒形にし，手振りから得られた計測点から慣性主軸を求め，曲面の軸を主軸に合わせる (図 9 **左**)．このとき，4.1 で述べたように，Z 方向の軸は高さ方向に固定し，水平方向の主軸に対してのみ軸を合わせている．

次に，計測点と曲面上の点との対応を求める．図 9 **右**は，計測点と曲面との対応を線で結んでいる様子を示している．最後に変形パラメータを推定し，超 2 次関数に適用して結果を得る (図 10 **左**)．前述したように，本システムは片側の輪郭形状から全体形状を推定するため，半径を推定できないので，推定後にそれを指定する (図 10 **右**)．

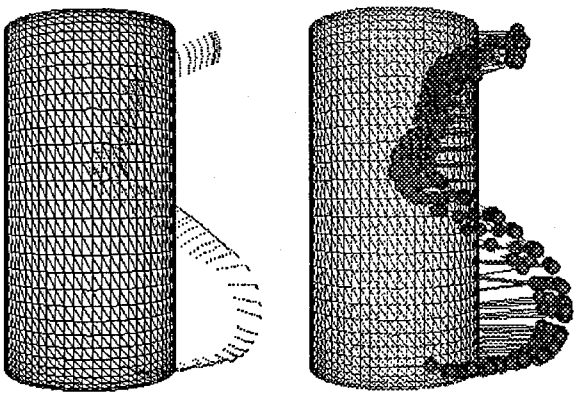

図 9: 壷の生成:**左**: 初期形状と計測点 (5200 点), **右**:計測点と曲面上の点との対応.

異なる形状を生成する 2 通りの手振りについて，その初期形状と推定結果を合わせて図 11,12 に示す．また，林檎の形状の特徴を表わす変形パラメータを用いて林檎の形状を推定した結果を図 13 に示す.

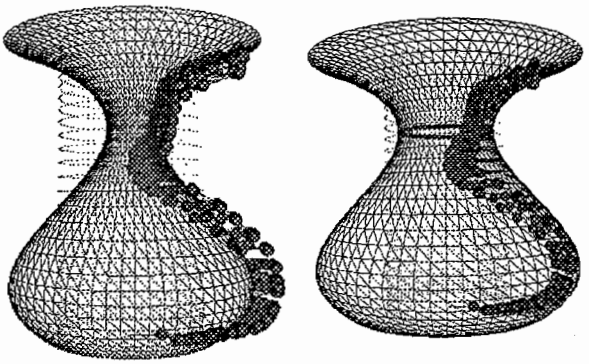

図 10: 壷の生成:**左**:手振りから推定された形状, **右**:壷の首 (太線で表わされている個所) の径を変更.

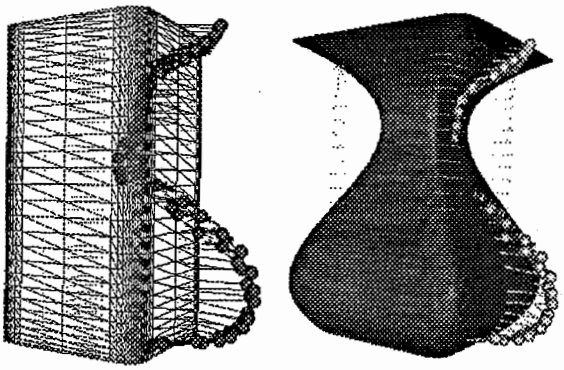

図 11: 角口の壷の例:**左**:手振りから得られた計測点 (1599 点) と初期形状, 計測点と曲面上の点との対応. **右**:推定結果.

6 考察

6.1 形状編集への適用

本報告では, 形状の生成に対する言語と手振りの利用を対象としているが, この方式を編集に適用することが可能である. 生成の過程では, 離散点集合の主軸方向のサイズを形状のサイズとしたが, 編集過程では, 仮想物体をグラフィックスで確認しながら手振りを行うことができることから, 物体の任意の部分の変形が可能となる. また, 超 2 次関数を用いていることから, その固有なパラメータに対応する概念的な言語表現による形状編集も可能である.

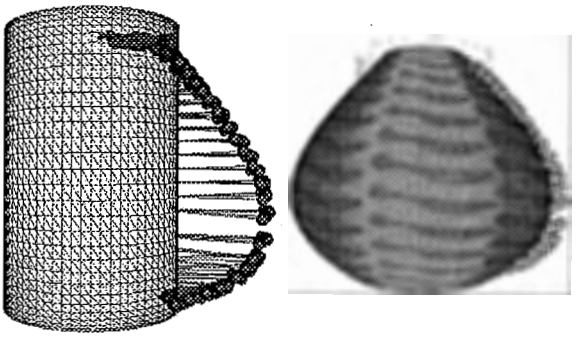

図 12: 首のない壷の例:**左**:手振りの軌跡から得られた計測点 (1313 点) と初期形状, 計測点と曲面上の点との対応. **右**:推定結果.

6.2 手振りで表現される形状と物体との対応

実験システムでは，手振りで表現される形状の特徴は回転体を正面から見た場合の輪郭線の形状であることとした．しかし，ドメインに関する情報が与えられている場合，我々は手振りが物体のどの部分の特徴を表現したものであるかを概ね推測することができている．これは，手振りで表現された形状と，ドメインに属する物体を実際にあてはめ，妥当性から判断していると単純に推測することができる．そこで，そのような物体の部分的な特徴に関する知識を利用することができれば，物体に対する手振りの位相的な関係を識別できると考えられる．

6.3 変形形態と言語との対応

筆者らは，計算機が形状の特徴を理解できるようにすることで，ユーザが直観的に仮想物体を操作・編集できるような環境の実現を目的としている．当初，手振りと言語から壷のような曲線を持つ形状を推定するに際し，誤差が小さいことから区分曲線補間による形状推定を検討した (図 14) が，この方法では自由度が分割数によって増大し，言語との対応が困難なことから使用を止めた．このような経緯から，言語的表現と対応付けられたパラメータを持つような形状表現方法である超 2 次関数を採用した．また，その超 2 次関数に対して一般的な形状表現能力を与える目的で用いられた変形形態についても，言語との対応について検討する必要がある．

先に述べたように，Pentland らは言語と対応付けられた変形パラメータを用いた．一方，筆者らは曲面を表現する目的で 3 次関数を表現できるような変形形態を用い，その結果，スケーリングパラメータ (u_6, u_7), テーパリングパラメータ (u_{15}, u_{21}) はその意味を喪失し，新たな言語的な意味との対応を検討する必要が生じてしまった．

3 次曲線は 4 自由度を持ち，これを言語と一意に対応付けることは一般に困難である．そこで，比較的に直観的な理解が容易な 3 次エルミート表現に基づいて検討を行うこととする．この表現に基づくと，3 次曲線はその 2 端点 $\mathbf{p}_0, \mathbf{p}_1$ と接ベクトル $\mathbf{m}_0, \mathbf{m}_1$ から，表現することができるが (図 15)，例えば，その定義域が $[-1, 1]$ の場合には，次のようなことが直

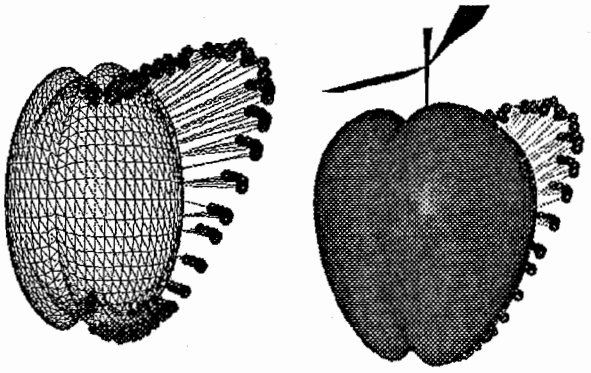

図 13: 林檎の例:**左**:手振りから得られた計測点 (871 点) と初期形状, 計測点と曲面上の点との対応. **右**:推定結果.

観的に分かる. $\mathrm{m}_0 = \mathrm{m}_1$かつ $\mathrm{p}_0 + 2\mathrm{m}_0 = \mathrm{p}_1$のとき, 2次, 3次の各項の係数が 0 となり, 直線を表わす. また, $\mathrm{p}_0 + 2\mathrm{m}_0 > \mathrm{p}_1$ かつ $\mathrm{p}_0 > \mathrm{p}_1 - 2\mathrm{m}_1$, あるいは $\mathrm{p}_0 + 2\mathrm{m}_0 < \mathrm{p}_1$ かつ $\mathrm{p}_0 < \mathrm{p}_1 - 2\mathrm{m}_1$のとき, 3次曲線は, 区間 $(-1, 1)$ において, 線分 $\mathrm{p}_0, \mathrm{p}_1$ と交点を持つ. 後者は, 壷の例では, 首の部分を持つか否かに対応するが, このような言語との対応がパラメータから得られそうである.

このような3次曲線の表現である3次エルミート多項式 H_i^3は, 次式 (11) のように表せる [20].

$$\begin{aligned} \mathrm{p}(t) &= \mathrm{p}_0 H_0^3(t) + \mathrm{m}_0 H_1^3(t) + \\ &\quad \mathrm{m}_1 H_2^3(t) + \mathrm{p}_1 H_3^3(t) \\ &\quad t \in [0,1] \end{aligned} \tag{11}$$

この表現方法を用いるには, 変形パラメータから $\mathrm{p}_0, \mathrm{p}_2, \mathrm{m}_0, \mathrm{m}_1$ が求める必要がある. $z \in [-1, 1]$ であるので, $z = t + (1-t)(-1) = 2t - 1$ とおいて定義域変換し, 式 (7) の d_{00}から次式が得られる.

$$\begin{bmatrix} 2 & 1 & -1 & 2 \\ -3 & -1 & -1 & 3 \\ 0 & -1 & 1 & 0 \\ 1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \mathrm{p}_0 \\ \mathrm{m}_0 \\ \mathrm{m}_1 \\ \mathrm{p}_1 \end{bmatrix} = \begin{bmatrix} u_6 \\ u_{15} \\ u_{30} \\ u_{31} \end{bmatrix} \tag{12}$$

2つの点 $\mathrm{p}_0, \mathrm{p}_1$ と2つの接ベクトル $\mathrm{m}_0, \mathrm{m}_1$ はパラメータをから以下の式より求められる.

$$\begin{bmatrix} \mathrm{p}_0 \\ \mathrm{m}_0 \\ \mathrm{m}_1 \\ \mathrm{p}_1 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & -2 & 3 \\ 0 & 1 & 2 & 3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_6 \\ u_{15} \\ u_{30} \\ u_{31} \end{bmatrix} \tag{13}$$

ただし, この議論は, “つまむ” に対応する変形パラメータが 0 のときに有効である.

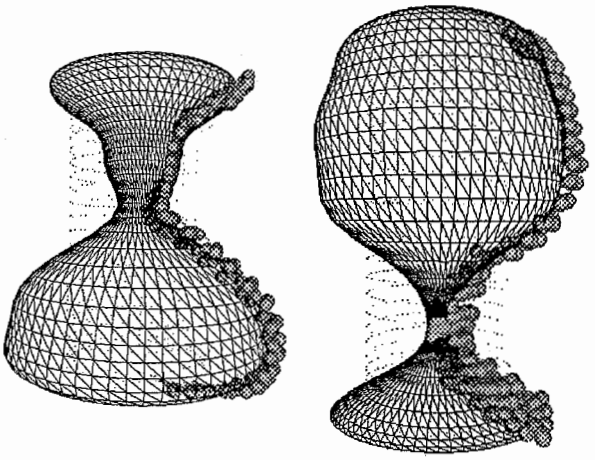

図 14: 区分曲線補間によって推定された壷 (左) とゴブレット (右)

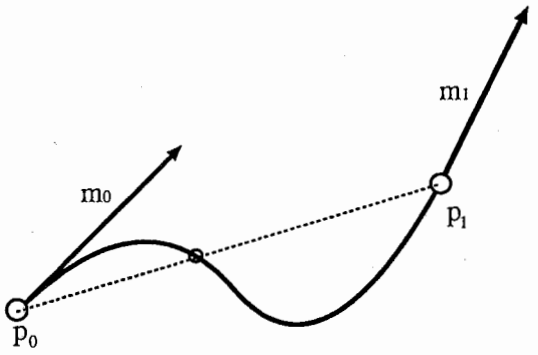

図 15: 端点と接ベクトルによる 3 次曲線表現

このように，3 次曲線のパラメータを言語と対応付けることはできそうだが，3 次曲線で表現できる形状はある程度限定される．より複雑な形状を表現するためには，先に触れたような，Spline,B-Spline 等の区分曲線補間による推定も考えられるが，言語との対応付けという点で不利である．このような目的に対し，アウトラインからディテールまでの形状の特徴を階層的に持つ表現，あるいはそのような特徴を自由に取り出せるような表現方法が提案されており [21, 22]，今後我々が提案した手法をより複雑な形状に適用するためにこれらの手法を適用することも考えられる．

7 まとめ

本報告では，臨場感通信会議システムに代表される仮想空間における形状生成インタフェースとして，言語と手振りを統合し仮想物体形状を生成できるインタフェースを提案した．我々

は，形状記述手法として拡張した超２次関数を使用し，形状生成インタフェースシステムを試作，人間が普段の対話で用いる表現手段により，メンタルイメージを直接的に可視化することを可能にした．本インタフェースは，メンタルイメージの形状入力以外にも，実在する物体形状の入力インタフェースや仮想物体の形状変形インタフェースへの適用が考えられる．

実験システムの使用例で述べたように本システムには制約が多く，軸対称でない形状生成，両手手振りへの対応により，一般的な形状を生成できるようにするのが今後の課題である．また，手振りの軌跡とメンタルイメージとの適合性について，今後その関連を評価実験を通して明らかにしていく予定である．

参考文献

[1] F. Kishino, T. Miyasato and N. Terashima: "Virtual Space Teleconferencing "Communication with Realistic Sensations" ", Proc of 4th IEEE International Workshop on Robot and Human Communication, Tokyo, pp. 205–210 (1995).

[2] 西本, 安部, 宮里, 岸野 : "連想記憶を用いた発散的思考支援システムにおける提供情報の分野制御の試み", 計測自動制御学会 第15回システム工学部会研究会「発想支援技術」, pp. 17–24 (1994).

[3] R. Bolt: "Put That There: voice and gesture at the graphics interface", Computer Graphics, **14**, 2, p. 262 (1980).

[4] K. Mochizuki, H. Takemura and F. Kishino: "Object manipulation and layout in a 3-D virtual space using a combination of natural language and hand pointing", Proc. of 4th Sapporo International Computer Graphics Symposium, HW-1, pp. 38–42 (1992).

[5] Y. A. Tijerino, K. Mochizuki and F. Kishino: "Interactive 3-D Computer Graphics Driven Through Verbal Instructions: Previous and Current Activities At ATR", Comput. & Graphics., **18**, 5, pp. 621–631 (1994).

[6] A. D. Wexelblt: "Natrural Gesture in Virtual Environments", Master's thesis, Massachusetts Institute of Technology (1994).

[7] M. Yoshida, Y. A. Tijerino, S. Abe and F. Kishino: "A Virtual Space Teleconferencing System that Supports Intuitive Interaction for Creative and Cooperative Work", Proc. of the 1995 Symposium on Interactive 3D Graphics, Monterey, CA., pp. 115–122 (1995).

[8] 杉浦, 中村, 吉村 : "３次元直接操作による手作り風造形インターフェース", 情報処理学会第45回全国大会予稿集, 第2巻, pp. 399–400 (1992).

[9] D. Weimer and S. Ganapathy: "A Shynthetic Visual Environment With Hand Gesture And Voice Input", Proc. of CHI'89, pp. 235–240 (1989).

[10] A. P. Pentland: "Towards and Ideal 3-D CAD System", SPIE conference on Machine Vision and the Man-Machine Interface, San Diego, CA., pp. 165–171 (1987).

[11] E. Sachs, A. Roberts and D. Stoops: "3-Draw: A Tool for Designing 3D Shapes", IEEE Computer Graphics & Applications, pp. 18–26 (1991).

[12] Y. A. Tijerino, S. Abe, T. Miyasato and F. Kishino: "What You Say Is What You See, - Interactive generation manipulation and modification of 3-D shapes based on verbal descriptions-", Artificial Intelligence Review Journal, **8**, 2, pp. 123–152 (1994).

[13] Y. A. Tijerino, S. Abe and F. Kishino: "Intuitive graphics representation of mental images in a virtual environment through natural language", Proc. of AAAI Fall Symposium Series (1995).

[14] 岸野, 志和, 北村：“３地点の参加者が一堂に会して仮想物体を生成・操作する”, ATR ジャーナル, **17**, pp. 2–3 (1994).

[15] 安部, 吉田, ジュリ A., 岸野:“マルチモーダル３-D形状編集システムの実現”, 信学技報 HC94-57, pp. 39–46 (1994).

[16] A. P. Harling: "Gesture input using neural networks", Technical report, University of York, Department of Computer Science (1993).

[17] D. Terzopoulos and D. Metazas: "Dynamic 3D Models with Local and Global Deformations : Deformable Superquadrics", IEEE Transactions On Pattern Analysis And Machine Intelligence, **13**, 7, pp. 703–714 (1991).

[18] S. Sclaroff. and A. Pentland.: "Generalized Implicit Function For Computer Graphics", Computer Graphics, **25**, 4, pp. 247–250 (1991).

[19] A. H. Barr: "Global and Local Deformations of Solid Primitives", Computer Graphics, **18**, 3, pp. 21–30 (1984).

[20] G. Farin: "Curves and Surfaces for Computer-Adided Geometric Design", Academic Press, Boston, second edition (1990).

[21] D. R. Forsey and R. H. Bartels: "Surface Fitting with Hierarchical Splines", ACM Transactions on Graphics, **14**, 2, pp. 134–161 (1995).

[22] A. Finkelstein and D. Salesin: "Multiresolution curves", Wishful Research Result 94-01-06b, University of Washington, Department of Computer Science and Engineering (1994).

A プログラムリストの解説

先に述べたように，ここで紹介したシステムはPentlandらの手法を拡張したており，その一部は彼らのプログラムに準ずるところがある．そこで，実験システムを開発するために拡張した部分のリストを掲載する．その他の部分については，Pentland[18] らの文献およびプログラムを参照されたい．

実験システムで使用した離散点集合からドメインに基づき形状を推定するプログラムはインタープリタである．このプログラムはスクリプトに従い，離散点集合である手振りデータの読み込み，形状の推定，推定結果等の描画モードの変更，ステイタスの表示等を行なうことができる．インタープリタの開発は，lex, yaccを使用した．また，形状推定の核となるプログラムはライブラリとし，インタープリタからコールされる最上位ルーチンはプログラムリストのmy_main.cで定義している．

形状推定のソースコード（一部抜粋）

- my_main.c
  形状推定のためのライブラリの最上位ルーチンを定義．その他に，ドメインに対応するmy_mode.cで定義される変形形態や超2次関数パラメータ等のテーブルを定義している．
- my_fit_multi.c
  形状推定モジュール
- my_mode.c
  変形形態に対応する変形マトリックスを生成するための関数群の定義
- my_mode.h
  ヘッダファイル
- my_phi.c
  形状推定に使用される変形パラメータに対するノード位置の微分値を有限差分により求めるための関数定義
- my_fit_ellipsoid.c
  離散点集合から慣性主軸およびそれらの方向へのサイズを求め，未変形な超2次関数曲面に座標変換を施し，離散点集合に対して適当な位置に移動するためのモジュール

インタープリタの全ソースコード

- hand.c
  インタープリタのメインモジュール
- hand.h
  ヘッダファイル
- handlex.l
  インタープリタの字句解析モジュール
- handparse.y
  インタープリタの構文解析モジュール

B プログラムリスト

次ページ以降に各プログラムリストを添付．

```
/*
 *
 * my_main.c : メイン関数および上位関数群
 * ------------------------------------------------------------------------------
 *
 */
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <assert.h>
#include "sq.h"
#include "my_main.h"
#include "my_mode.h"
#include "my_mat.h"

POINT3D*                gData_points = NULL;
long                    gN_data_points = 8;     /* should be determined by collectGes
tureDatra() */
SUPERQUAD*              gSq = NULL;             /* supports just one SQ */
POINT3D*                gWeights = NULL;        /* used by weighted LSM */
POINT3D*                gVariances = NULL;
POINT3D*                gFitPoints = NULL;      /* points on the SQ */
int                     gFitPointsNum;
int**                   gFittedPoints = NULL;   /* points in the mesured data */
int                     gFlagPassPointMap = 0;
int                     gFlagPassGenpoly = 0;
FLOAT_TYPE              gSampling = PI/24.0e0;
FILE*                   gFp = NULL;
FLOAT_TYPE*             gDmap = NULL;
char*                   gRangefname = NULL;
int                     gN_iterations = 1;
int                     gIterate_flag;
int                     gN_modes;
int                     gW_step = 1;
FLOAT_TYPE              gSmoothingRadius;       /* unused */
FLOAT_TYPE              gGamma;                 /* */
int                     gUse_svd = 1;
int                     gSolution_technique = FIT_MODES;
int                     gPartial_flag = 1;      /* always 1 */
int                     gDomain =  0;
FLOAT_TYPE              gMaxWidth = 0.0e0;
int                     gSetRadiusFlag = 0;
FLOAT_TYPE              gRadiusZ;
FLOAT_TYPE              gRadiusR;

/* Static Definitions
 */
static int              startFlag = 0;
static char             info_string[256];

int                     gDomainInfoNum = 0;
int                     gTextureDefNum;

/* 静的に定義されたドメイン情報 */

/* アップルのドメインの範囲や順番を変えるばあいはdraw_gl.cのappleパー */
/* ツドローイング部分のドメイン依存部も変更すること*/
DomainDependentParameters       gDomainInfo[] = {
  /* NORMAL */
  {1.0e0, 1.0e0, 0.0e0, 0.0e0, 0.0e0, 1.0e0, 35, getDM_Slant, -1.0, 1.0, 0, MAT_SHINY
BRASS}, /* 0 */
  {1.0e0, 1.0e0, 1.0e0, 1.0e0, 0.0e0, 1.0e0, 45, getDM_RBF, -1.0, 1.0, 5, MAT_SHINYBR
ASS}, /* 1 */
  {1.0e0, 1.0e0, 1.2e0, 1.2e0, 1.2e0, 1.2e0, 30, getDM_NoSym, -1.0, 1.0, 0, MAT_SHINY
BRASS}, /* 2 */
```



```
  /* APPLE */
  {1.0e0, 1.2e0, 1.0e0, 1.0e0, 0.9e0, 0.9e0, 35, getDM_2, -1.9, 1.9, 2, MAT_APPLE},
   /* 3 */
  {1.0e0, 1.2e0, 0.5e0, 0.5e0, 0.9e0, 0.9e0, 35, getDM_2, -1.9, 1.9, 2, MAT_APPLE},
   /* 4 */
  {1.0e0, 1.2e0, 0.0e0, 0.0e0, 0.9e0, 0.9e0, 35, getDM_2, -1.9, 1.9, 2, MAT_APPLE},
   /* 5 */

  {1.0e0, 1.2e0, 1.0e0, 1.0e0, 0.9e0, 0.9e0, 35, getDM_Apple_Special, -1.9, 1.9, 2, M
AT_APPLE}, /* 6 */
  {1.0e0, 1.2e0, 0.5e0, 0.5e0, 0.9e0, 0.9e0, 35, getDM_2_Apple, -1.9, 1.9, 2, MAT_APP
LE},       /* 7 */
  {1.0e0, 1.2e0, 0.0e0, 0.0e0, 0.9e0, 0.9e0, 35, getDM_2_Apple, -1.9, 1.9, 2, MAT_APP
LE},       /* 8 */

  /* TSUBO */
  /* square from front view */
  {1.0e0, 1.3e0, 1.0e0, 1.0e0, 1.0e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 10, MAT_PEWT
ER},       /* 9 */
  {1.0e0, 1.3e0, 1.0e0, 1.0e0, 0.5e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 10, MAT_PEWT
ER},       /* 10 */
  {1.0e0, 1.3e0, 1.0e0, 1.0e0, 0.0e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 10, MAT_PEWT
ER},       /* 11 */

  {1.0e0, 1.3e0, 1.0e0, 1.0e0, 0.0e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 10, MAT_PEWT
ER},       /* 12 */
  {1.0e0, 1.3e0, 2.5e0, 2.5e0, 0.0e0, 1.0e0, 34, getDM_3_Sym, -0.99, 0.99, 6, MAT_PEW
TER},      /* 13 */
  {1.0e0, 1.3e0, 1.5e0, 1.5e0, 0.0e0, 1.0e0, 34, getDM_3_Sym, -0.99, 0.99, 11, MAT_PE
WTER},     /* 14 */
  /* fillet from front view */
  {1.0e0, 1.3e0, 1.0e0, 1.0e0, 0.2e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 4, MAT_PEWTE
R},        /* 15 */
  {1.0e0, 1.3e0, 0.5e0, 0.5e0, 0.2e0, 1.0e0, 34, getDM_1_2, -1.0, 1.0, 1, MAT_PEWTER}
,  /* 16 */
  {1.0e0, 1.3e0, 0.2e0, 0.2e0, 0.2e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 0, MAT_PEWTE
R},        /* 17 */

  {1.0e0, 1.3e0, 0.0e0, 0.0e0, 0.2e0, 0.0e0, 34, getDM_3_Sym, -1.0, 1.0, 4, MAT_PEWTE
R},        /* 18 */
  {1.0e0, 1.3e0, 2.5e0, 2.5e0, 0.2e0, 0.0e0, 34, getDM_3_Sym, -1.0, 1.0, 5, MAT_PEWTE
R},        /* 19 */
  {1.0e0, 1.3e0, 1.5e0, 1.5e0, 0.2e0, 0.0e0, 34, getDM_3_Sym, -1.0, 1.0, 6, MAT_PEWTE
R},        /* 20 */
  /* round from front view */
  {1.0e0, 1.3e0, 1.0e0, 1.0e0, 1.0e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 7, MAT_PEWTE
R},        /* 21 */
  {1.0e0, 1.3e0, 0.5e0, 0.5e0, 1.0e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 8, MAT_PEWTE
R},        /* 22 */
  {1.0e0, 1.3e0, 0.2e0, 0.2e0, 1.0e0, 1.0e0, 34, getDM_3_Sym, -1.0, 1.0, 9, MAT_PEWTE
R},        /* 23 */

  {1.0e0, 1.3e0, 0.0e0, 0.0e0, 1.0e0, 0.0e0, 34, getDM_3_Sym, -1.0, 1.0, 10, MAT_PEWT
ER},       /* 24 */
  {1.0e0, 1.3e0, 2.5e0, 2.5e0, 1.0e0, 0.0e0, 34, getDM_3_Sym, -1.0, 1.0, 11, MAT_PEWT
ER},       /* 25 */
  {1.0e0, 1.3e0, 1.5e0, 1.5e0, 1.0e0, 0.0e0, 34, getDM_3_Sym, -1.0, 1.0, 1, MAT_PEWTE
R},        /* 26 */
};

/*---*/
/*------------------------------------------------------------------------------
 *
 * Name         : genPolygons
 *
 * Description  :
```

```
 *
 *
 * Arguments     :
 *
 *
 * Returns       : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author        : M.Yoshida
 *
 * Date          : 17:27:00 @ 05Dec1995
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/

int
genPolygons(FLOAT_TYPE sampling)
{
  if(!gSq)
    return 0;

  if(sampling >= 0.0e0)
    {
      if(sampling != gSampling)
        {
          gFlagPassGenpoly = 1;
          gSq->set_sampling(gSq, sampling);
          gSmoothingRadius = gSampling * 0.5e0;
        }
    }
  gSq->polygons(gSq);
  return 1;
}

/*-------------------------------------------------------------------------
 *
 * Name          : drawSQ
 *
 * Description   :
 *
 *
 * Arguments     :
 *
 *
 * Returns       : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author        : M.Yoshida
 *
 * Date          : 17:25:00 @ 05Dec1995
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/

int
drawSQ(char* string)
{
  int   f = 0;
  if(gData_points)
    {
      _draw_points(gData_points, gN_data_points, 0);    /* draw_gl.c */
      f++;
    }
  if(gSq)
    {
      gSq->deform(gSq);
```



```
      gSq->draw(gSq);
      f++;
    }
  if(f)
    swapbuffers();
  return 1;
}

/*-------------------------------------------------------------------------
 *
 * Name          : promptSQ
 *
 * Description   :
 *
 *
 * Arguments     :
 *
 *
 * Returns       : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author        : M.Yoshida
 *
 * Date          : 17:25:00 @ 05Dec1995
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/

int
promptSQ(char* string)
{
  prompt_and_wait(string);
  return 1;
}

/*-------------------------------------------------------------------------
 *
 * Name          : initMyMain
 *
 * Description   : Evaluate command arguments.
 *                 Some global values are initialized.
 *
 * Arguments     : argc
 *                 argv
 *
 * Returns       : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author        : M.Yoshida
 *
 * Date          : 13:14:00 @ 01Dec1995
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/

int
initMyMain(int argc, char** argv)
{
  static int flag = 1;

  if(flag)
    {
      flag = 0;
      fprintf(stderr, "initMyMain: initialize\n");

      /* collect hand points data set */
```

```
    if(argc && argv)
      {
        if (scanargs(argc, argv,
                     "% n%-iterate_num!d range_file%s",
                     &gIterate_flag, &gN_iterations,
                     &gRangefname) == 0)
          {
            exit(1);
          }
      }

    if(gN_iterations < 0)
      {
        fprintf(stderr, "Invalid arguments: positive or 0 value is allowed\n");
      }

  }

/* determin the number of modes for the current object */
/* Should I use GUI for setting this parameter? 24Nov1995 */
gN_modes = 34;

/* initialize Super Quadrics data */
if(gSq)
  {
    free_sq_type(gSq);
    gSq = NULL;
  }
if((gSq = make_sq_type(TYPE_SUPERQUADRIC_MULTI, gN_modes)) == NULL)/* my_sq.c */
  {
    fprintf(stderr, "Failed to careate SUPERQUAD record with %03d modes.\n",
            gN_modes);
    exit(1);
  }

/* This is how much we allow data to smooth out its influence on the SQ surface
 */
gSampling = PI/32.0e0;

gSmoothingRadius = gSampling * 0.5e0;

/* initialize SuperQuadric
 */
gSq->set_sampling(gSq, gSampling);     /* sq.c */
gSq->polygons(gSq);                    /* sq.c */
gSq->deform(gSq);                      /* sq.c */

/* This is the variance assigned to ummatched SQ surface points
 */
gGamma = 10.0e0;

return 1;
}

/*-------------------------------------------------------------------------
 *
 * Name         : collectGestureData
 *
 * Description  : Read data of gesture from the specified file.
 *                Data will be stored in the record "gN_data_points",
 *                And variances will be computed and be stored in gVariances.
 *
 * Arguments    : fname : data file path
 *                gData_points : pointer to a record in which data will be stored.
 *                gVariances : pointer to a record in which computed variances
 *                will be stored.
```



```
 *
 * Returns      : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author       : M.Yoshida
 *
 * Date         : 13:15:00 @ 01Dec1995
 *
 * Changes      :
 *
 *-------------------------------------------------------------------------*/

int
collectGestureData(char* fname)
{
  int   ret;

  ret = collect_hand_gesture_data(fname,
                                  &gData_points, &gVariances);
  return ret;
}

/*-------------------------------------------------------------------------
 *
 * Name         : fitEllipsoidToData
 *
 * Description  : Fit ellipsoid coordinates to the gesture data.
 *                This computation is performed on XY plane, not Z.
 *
 * Arguments    :
 *
 * Returns      : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author       : M.Yoshida
 *
 * Date         : 13:22:00 @ 01Dec1995
 *
 * Changes      :
 *
 *-------------------------------------------------------------------------*/

int
fitEllipsoidToData(void)
{
  if(!gSq)
    return 0;

  /* fit super quadrics to measured data (around Y axis) */
  find_sq_ellipsoid_xy(gSq, gData_points, gN_data_points);      /* my_fit_ellipsoid.c
 */

  return 1;
}

/*-------------------------------------------------------------------------
 *
 * Name         : setDomain
 *
 * Description  : set the current domain.
 *
 * Arguments    : domainid: number which is correspond to a domain uniquely.
 *
 * Returns      : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author       : M.Yoshida
 *
 * Date         : 13:42:00 @ 01Dec1995
```

```
 *
 * Changes      :
 *
 *-----------------------------------------------------------------------------*/

int
setDomain(long domain)
{
  gDomainInfoNum = sizeof(gDomainInfo) / sizeof(DomainDependentParameters);
  if((domain >= 0) && (domain < gDomainInfoNum))
    {
      int       ret;
      gDomain = domain;
      gSq->n_modes = getDeformationModeNum(domain);
      if(set_mode(domain) != 1)/* my_mode.c */
        {
          fprintf(stderr, "setDomain: Failed to set_mode\n");
        }
      fprintf(stderr, "setDomain : youe domain id was allowed to set.\n");

      /* load texture */
      ret = loadTexture(gDomainInfo[gDomain].texid);
      switch(ret)
        {
        case 0:
          fprintf(stderr, "A specified Texture image has been loaded already. @ setDo
main ");
          break;
        case -1:
          fprintf(stderr, "Failed to load a texture image @ setDomain");
          break;
        case 1:
          break;
        }
    }
  else
    {
      fprintf(stderr, "setDomain : invalid domain id was specified.\n");
    }

  return 1;
}

/*-----------------------------------------------------------------------------
 *
 * Name         : computePhiMatrix
 *
 * Description  : compute Phi matrix.
 *
 * Arguments    :
 *
 * Returns      : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author       : M.Yoshida
 *
 * Date         : 15:00:00 @ 01Dec1995
 *
 * Changes      :
 *
 *-----------------------------------------------------------------------------*/

int
computePhiMatrix(void)
{
  if(!gSq)
    return 0;
```



```
  gSq->deform(gSq);
  /* calculate phi matrix */
  gSq->compute_phi(gSq);

  return 1;
}

/*-----------------------------------------------------------------------------
 *
 * Name         : mapClosestPoints
 *
 * Description  : establish the correspondence between data and model points.
 *                This function calls closest_map_points @ point_mapping.c
 *
 * Arguments    :
 *
 * Returns      : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author       : M.Yoshida
 *
 * Date         : 15:27:00 @ 01Dec1995
 *
 * Changes      :
 *
 *-----------------------------------------------------------------------------*/

int
mapClosestPoints(void)
{
  if(!gSq || !gData_points || !gFitPoints || !gVariances || !gWeights)
    {
      fprintf(stderr, "mapClosestPoints: <%p,%p,%p,%p,%p>\n",
              gSq, gData_points, gFitPoints, gVariances, gWeights);
      return 0;
    }
  closest_map_points(gSq,
                     gData_points,
                     (int)gN_data_points,
                     gFitPoints,
                     gVariances,
                     gWeights,
                     gSmoothingRadius,
                     gGamma);
  gFitPointsNum = gSq->n_points;
  gFlagPassGenpoly = 0;
  return 1;
}

/*-----------------------------------------------------------------------------
 *
 * Name         : fitModel
 *
 * Description  : fit model to the measured data points.
 *
 * Arguments    :
 *
 * Returns      : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author       : M.Yoshida
 *
 * Date         : 15:33:00 @ 01Dec1995
 *
 * Changes      :
 *
 *-----------------------------------------------------------------------------*/
```

```
int
fitModel(void)
{
  FLOAT_TYPE    lambda = 1.0e-5;
  int           ret;

  if(!gSq || !gData_points || !gFitPoints || !gVariances || ! gWeights)
    return 0;

  /* fit model   */
  ret = gSq->fit_model(gSq, gFitPoints, gWeights, gN_iterations, gUse_svd,
                       gSolution_technique, lambda);

  return ret;
}

/*---*/
/*-------------------------------------------------------------------------
 *
 * Name         : collect_hand_gesture_data
 *
 * Description  : read hand data
 *                This function is to be called from collectGestureData().
 *
 * Arguments    :
 *
 * Returns      : Normally returns 1. Resturns 0 if error is occured.
 *
 * Author       : M.Yoshida
 *
 * Date         : 13:56:00 @ 01Dec1995
 *
 * Changes      :
 *
 *-------------------------------------------------------------------------*/

static int
collect_hand_gesture_data(char* fname,          /* file name */
                          POINT3D** points_p,   /* pointer to storage for points */
                          POINT3D** variances_p)        /* pointer to storage for var
iances */
{
  /* open the hand gesture points data */

  if(gFp)
    {
      fclose(gFp);
    }
  if((gFp = fopen(fname, "r")) == NULL)
    {
      fprintf(stderr,"Failed to open range data file %s\n",fname);
      perror(fname);
      return 0;
    }

  gN_data_points = 7000;
  fprintf(stderr, "collect_hand_gesture_data: sorry, I can read only %05d data points
.\n");

  /* temporal codes */
  if(*points_p)
    {
      free(*points_p);
    }
  if((*points_p = (POINT3D*)calloc(gN_data_points, sizeof(POINT3D))) == NULL)
```



```
    {
      perror("Failed to calloc points_p");
      return 0;
    }
  if(*variances_p)
    {
      free(*variances_p);
    }
  if((*variances_p = (POINT3D*)calloc(gN_data_points, sizeof(POINT3D))) == NULL)
    {
      perror("Failed to calloc variances_p");
      return 0;
    }

#if 1 /* temporal codes */
  gN_data_points = readFromObjFile(fname, *points_p, *variances_p);
  fprintf(stderr, "collect_hand_gesture_data: read %05d points.\n",
          gN_data_points);
#endif

  /* added by M.Yoshida 11Jan1996 */
  if((gFittedPoints = (int**)calloc(gN_data_points, sizeof(int*))) == NULL)
    {
      perror("malloc gFittedPoints");
      fprintf(stderr, "collect_gesture_data_points: Failed to malloc for gFittedPoint
s\n");
      exit(1);
    }
  return 1;
}

int
readFromObjFile(char* fname, POINT3D* points, POINT3D* variances)
{
  char  s1[256];
  char  s2[256];
  char  s3[256];
  char  s4[256];
  int           num = gN_data_points;
  int           sum;
  POINT3D       max, min, center;
  POINT3D       p;
  POINT3D*      sp = points;
  double        max_width, w;
  register int  k;
  int           n = 0;

  min.x = min.y = min.z = REALLY_BIG_NUMBER;
  max.x = max.y = max.z  = -REALLY_BIG_NUMBER;
  sum = 0;
  while(fscanf(gFp, "%s %s %s %s", s1, s2, s3, s4) == 4 &&
        num > 0)
    {
      sscanf(s2, "%lf", &p.x);
      sscanf(s3, "%lf", &p.y);
      sscanf(s4, "%lf", &p.z);

      points->x = -p.z;
      points->y = p.x;
      points->z = -p.y;

      MINMAX(min.x, max.x, points->x);
      MINMAX(min.y, max.y, points->y);
      MINMAX(min.z, max.z, points->z);

      variances->x = variances->y = variances->z = 1.0;
```

```
      num--;
      ++variances;
      points++;
      n++;
    }
  fprintf(stderr, "MIN (% -012.5lf % -012.5lf % -012.5lf) \n",
          min.x, min.y, min.z);
  fprintf(stderr, "MAX (% -012.5lf % -012.5lf % -012.5lf) \n",
          max.x, max.y, max.z);

  center.x = (min.x + max.x) / 2.0e0;
  center.y = (min.y + max.y) / 2.0e0;
  center.z = (min.z + max.z) / 2.0e0;

  max_width = (max.x - min.x) / 2.0e0;
  w = (max.y - min.y) / 2.0e0;
  max_width = MAX(max_width, w);
  w = (max.z - min.z) / 2.0e0;
  max_width = MAX(max_width, w);
  for(k = 0; k < n; k++)
    {
      sp[k].x -= center.x;
      sp[k].y -= center.y;
      sp[k].z -= center.z;
      sp[k].x *= 250.0e0/max_width;
      sp[k].y *= 250.0e0/max_width;
      sp[k].z *= 250.0e0/max_width;
      sp[k].x += 250.0e0;
      sp[k].y += 250.0e0;
    }
  gMaxWidth = 250.0e0/max_width;
  return n;
}
/*---*/
```

```
/*
 *
 * Name:                my_fit_multi.c
 *
 * Descriptions:        my_phiで得られた微分値から最適な変形パラメータを求める
 *
 * ----------------------------------------------------------------------------
 *
 */

#include "sq.h"

/* Prototype Declarations
 */
int my_fit_model_multi(SUPERQUAD*       sq,
                       POINT3D*         fit_points,     /* give by get_sq_points(sq)
*/
                       POINT3D*         weights,        /* weights[sq->n_points][3] *
/
                       int              n_iterations,   /* n_iterations = 1 */
                       int              use_svd,        /* use_svd + 0 see solve_weig
hted_lsq */
                       int              solution_technique, /* FIT_MODES(=1) */
                       FLOAT_TYPE       lambda);        /* 1.0e-7  */

/*---*/
/*----------------------------------------------------------------------------
 *
 * Name          : my_fit_model_multi
 *
 * Description   : 曲面が離散点集合に近似するような変形パラメータを求める
 *                 最小自乗法による.
 *
 * Arguments     : sq : 超２次関数に関するデータのレコード
 *
 * Returns       : On failure 0 return.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *----------------------------------------------------------------------------*/
int
my_fit_model_multi(SUPERQUAD*   sq,
                   POINT3D*     fit_points,     /* give by get_sq_points(sq) */
                   POINT3D*     weights,        /* weights[sq->n_points][3] */
                   int          n_iterations,   /* n_iterations = 1 */
                   int          use_svd,        /* use_svd + 0 see solve_weighted_lsq
 */
                   int          solution_technique, /* FIT_MODES(=1) */
                   FLOAT_TYPE   lambda)         /* 1.0e-7  */
{
  int           i,j,k,l;
  matrix_type   *rotated_phi,*phi,*phi_transpose;
  FLOAT_TYPE    *mode_deltas;
  POINT3D       *point_deltas;


  /* allocate matrices
   *
   * phi_transpose = sq->phi
   */
  phi_transpose = sq->phi;
```



```
  /*
   * transpose
   * phi -  phi_transpose
   ****************************************
   * phi_transpose = sq->phi
   *--------------------------------------
   * w = n_points * 3
   * h = n_modes
   *======================================
   * phi
   *--------------------------------------
   * w = n_modes = phi->w = phi_transpose->h
   * h = n_poitns * 3
   ****************************************
   */
  phi = make_matrix_type(phi_transpose->h,
                         phi_transpose->w,
                         "phi");

  for(k = i = 0; i < phi->w; ++i)
    {
      for(j=0, l=i; j < phi->h; ++j,++k,l += phi->w)
        {
          phi->m[l] = phi_transpose->m[k];
        }
    }

  /*
   * rotated_phi
   *
   * w : phi->w, h : phi->h             dx/du
   *
   * mode_deltas
   * w : 1, h : phi->w
   *
   * point_deltas
   * w : 1, h = sq->npoints * 3 or w : 3, h: sq->n_points
   *
   */
  rotated_phi = make_matrix_type(phi->w, phi->h, "rotated phi");
  mode_deltas = (FLOAT_TYPE *)calloc(phi->w, sizeof(FLOAT_TYPE));
  point_deltas = (POINT3D *)calloc(sq->n_points, sizeof(POINT3D));

  /*
   * n_iterations = 1
   *
   */
  for(i = 0; i < n_iterations; ++i)
    {
      /* calculate deltas between points
       *
       * deformed_points: SCALE & ROTATE
       *
       *
       *
       */
      for(j = 0; j < sq->n_points; ++j)
        {
          point_deltas[j].x = fit_points[j].x - sq->x_trans - sq->deformed_points[j].
x;
          point_deltas[j].y = fit_points[j].y - sq->y_trans - sq->deformed_points[j].
y;
          point_deltas[j].z = fit_points[j].z - sq->z_trans - sq->deformed_points[j].
z;
        }
```

```
    /*
     * rotate the phi matrix into the sq orientation
     *
     * phi rotated_phi
     *
     * sq->R
     *
     *  phi
     *-----------------------------------------
     *  w = n_modes = phi->w = phi_transpose->h
     *  h = n_poitns * 3
     *
     *  phi
     *  [X]...
     *  [Y]...
     *  [Z]...
     *  [X]...
     *
     * sq->R : [3][3]
     *
     *
     * Rotate dx/du du = rotate scale dx
     *
     *
     *
     */
    rotate_phi_matrix(phi, sq->R, rotated_phi);

    /* solve weighted least squares
     *
     * r_phi diag(weights) mode_deltas = diag(weights) point_deltas
     *
     * unknow vector : mode_deltas
     *
     * sq->mode_covariance :
     *
     * use_svd
     *
     * cov <- Inv(T(A_w) A_w) : [n_modes][n_modes]
     *
     * A_w = A diag(w)
     *-----------------------------------------
     * considering weights (but weights = 1)
     *
     */
    solve_weighted_lsq(rotated_phi,    /* rotated phi */
                       mode_deltas,    /* mode_deltas[n_modes] */
                       point_deltas,   /* point_deltas[sq->n_points][3] */
                       weights,        /* weight is argument of this function
                                        * weight[sq->npoints] = 1
                                        */
                       sq->mode_covariance, /* [n_modes][n_modes] */
                       use_svd); /*
                                  * singular value decomposition
                                  */


    /*
     * update the modes -- position is handled specially
     *
     * ------------------------------------------------------------
     * mode[0,1,2]   (MODE_30_X_TRANS, MODE_30_Y_TRANS, MODE_30_Z_TRANS)
     *
     * sequence : R -> T so that modes[j]
     *
     *  All modes are considering rotation.
     *
```



```
     * translation
     *
     */
    for(j = 0; j < 3; ++j)
      {
        /*   mode_deltas : in object coordinates */
        sq->modes[j] += (mode_deltas[0] * sq->R[j][0] +
          mode_deltas[1] * sq->R[j][1] + mode_deltas[2] * sq->R[j][2]);
        fprintf(stderr, "mode_deltas[%02d] : % -12.5f\n", j, mode_deltas[j]);
      }

    for(j = 3; j < sq->n_modes; ++j)
      {
        sq->modes[j] += mode_deltas[j];
        fprintf(stderr, "mode_deltas[%02d] : % -12.5f\n", j, mode_deltas[j]);
      }

    /* update the polygons
     */
    sq->translate(sq,sq->modes[MODE_30_X_TRANS],sq->modes[MODE_30_Y_TRANS],
                  sq->modes[MODE_30_Z_TRANS]);
    sq->rotate(sq,sq->modes[MODE_30_X_ANGLE],sq->modes[MODE_30_Y_ANGLE],
               sq->modes[MODE_30_Z_ANGLE]);
    sq->scale(sq,sq->modes[MODE_30_X_SCALE],sq->modes[MODE_30_Y_SCALE],
              sq->modes[MODE_30_Z_SCALE]);

#if 0
    fprintf(stderr, "mode_delatas % -12.5f, % -12.5f\n",
            mode_deltas[9], mode_deltas[11]);
#endif
    sq->deform(sq);
    sq->temp_draw(sq);

#if 0
    {
      char info_string[256];

      sprintf(info_string,"%02d / %02d iteration to fit model...  Click mouse to ex
it",
              i, n_iterations);
      fprintf(stderr, "%s\n", info_string);
      prompt_and_wait(info_string);
    }
#endif

  }
/* free up matrices */
free_matrix_type(phi);
free_matrix_type(rotated_phi);
free(mode_deltas);
free(point_deltas);
}
/*---*/
```

```
/*
 *
 * my_mode.c :
 *
 * Descriptions: このプログラムでは各ドメインに対応するような変形形態
 *               を定義するような変形マトリックスを生成する関数群を定
 *               義している.
 *
 * Author:      M. Yoshida
 *
 * ------------------------------------------------------------------------
 * ------------------------------------------------------------------------
 * relating source files:
 *      my_sq_multi.c           -               22Nov1995
 *
 */
#include "sq.h"
#include "my_main.h"
#include "my_mode.h"

/* Prototype Declaration
 */
int     set_mode(long domain);
int     get_deformation_matrix(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                               FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int     getDeformationMpdeNum(int domain);

int getDM_RBF(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
              FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_Slant(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_Car(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
              FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_1_2(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
              FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_Apple_Special(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                        FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_1_Apple(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                  FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_2_Apple(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                  FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_2_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                  FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_2_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_3_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                  FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_3_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
          FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
int getDM_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
              FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);

/* Static Functions Declarations
 */
static int (*getDeformationMatrix)(FLOAT_TYPE def[3][3], FLOAT_TYPE* mode_values,
                                   FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z) = NULL;

/*---*/
/*------------------------------------------------------------------------
 *
 * Name         : set_mode
 *
```



```
 * Description  : ドメインに対応する変形形態関数をセットする.
 *
 * Arguments    : domain : ドメインＩＤ
 *
 * Returns      : On failure 0 return.
 *
 * Author       : M.Yoshida
 *
 * Date         :
 *
 * Changes      :
 *
 *------------------------------------------------------------------------*/
int
set_mode(long domain)
{

  /*   if((domain + 1) * sizeof(void*) > sizeof(getDMFunctions)) */
  if((domain < 0) || (domain >= gDomainInfoNum))
    {
      fprintf(stderr, "set_mode: Failed to set_mode\n");
      return 0;
    }

#if 0
  getDeformationMatrix = (int(*)(FLOAT_TYPE[][3], FLOAT_TYPE*, FLOAT_TYPE, FLOAT_TYPE
, FLOAT_TYPE))
    getDMFunctions[domain];
#else
  getDeformationMatrix = gDomainInfo[domain].func;
#endif

  fprintf(stderr, "set_mode: getDeformationMatrix was changed to %02d domain\n",
          domain);

  return 1;
}

/*------------------------------------------------------------------------
 *
 * Name         : set_mode
 *
 * Description  : 現在のドメインと位置に対応する変形形態から変形マトリックスを作成す
る
 *
 * Arguments    : def[][3]      : 変形マトリックス
 *                mode_values   : 変形形態に対応するパラメータの値
 *                x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns      : On failure returns 0.
 *
 * Author       : M.Yoshida
 *
 * Date         :
 *
 * Changes      :
 *
 *------------------------------------------------------------------------*/
int
get_deformation_matrix(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                       FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  if(getDeformationMatrix)
    return (*getDeformationMatrix)(def, mode_values, x, y, z);
  fprintf(stderr, "get_deformation_matrix: failed to exec getDeformationMatrix()\n");
  return 0;
```

```
}

/*---*/
/*-------------------------------------------------------------------------------
 *
 * Name          : getDM_RBF
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ジェネラルスプラインによる補間（１０分割）
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------------*/
int getDM_RBF(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
              FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  int           i;
  double        m = 0.0e0;
#define SEGMENT 10
  def[0][0] = 0.0e0;
  for(i = 0; i < SEGMENT+1; i++)
    {
      double    t = (double)i/(double)(SEGMENT);
      double    _m;
      _m = exp(-5.0e0 *  pow((z - (2.0e0 * t - 1)), 2.0e0));
      m += _m;
      def[0][0] += mode_values[34 + i] * _m;
    }
  def[0][0] /= m;
  def[0][0] += mode_values[6];

  def[0][1] = 0.0e0;
  def[0][2] = 0.0e0; /* (SGN(x) * mode_values[17]) * z * 2.0e0; */
  def[1][0] = 0.0e0;

  def[1][1] = def[0][0];
#if 0
  + mode_values[15] * z - mode_values[17] +
    mode_values[30] * pow(z, 2.0e0) +   mode_values[31] * pow(z, 3.0e0);
#endif
  def[1][2] = 0.0e0; /* (SGN(y) * mode_values[17]) * z  * 2.0e0; */
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] = mode_values[8]  - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z);
  return 1;
}

/*-------------------------------------------------------------------------------
 *
 * Name          : getDM_Slant
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 xy方向にノンプロポーショナルな形状を生成する.
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
```



```
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------------*/
int getDM_Slant(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_3_Sym\n"); */
  def[0][0] = mode_values[6]  + mode_values[6] * z;
  def[0][1] = 0.0e0;
  def[0][2] = 0.0e0; /* (SGN(x) * mode_values[17]) * z * 2.0e0; */
  def[1][0] = 0.0e0;

  def[1][1] = mode_values[6] * 1.2;
#if 0
  + mode_values[15] * z - mode_values[17] +
    mode_values[30] * pow(z, 2.0e0) +   mode_values[31] * pow(z, 3.0e0);
#endif
  def[1][2] = 0.0e0; /* (SGN(y) * mode_values[17]) * z  * 2.0e0; */
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] = mode_values[8]  - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z);
  return 1;
}

/*-------------------------------------------------------------------------------
 *
 * Name          : getDM_Car
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 xの負方向側が変化しない非軸対称な形状を生成する
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       : Carに特別な意味はない
 *
 *-------------------------------------------------------------------------------*/
int getDM_Car(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
              FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_3_Sym\n"); */
  if(x > 0)
    {
      def[0][0] = mode_values[6]  + mode_values[15] * z -
        mode_values[17] + mode_values[30] * pow(z, 2.0e0) + mode_values[31] * pow(z,
3.0e0);
    }
  else
    {
      def[0][0] = mode_values[6];
    }
```

```
  def[0][1] = 0.0e0;
  def[0][2] = 0.0e0; /* (SGN(x) * mode_values[17]) * z * 2.0e0; */
  def[1][0] = 0.0e0;

  def[1][1] = mode_values[6] * 1.2;
#if 0
  + mode_values[15] * z - mode_values[17] +
    mode_values[30] * pow(z, 2.0e0) +   mode_values[31] * pow(z, 3.0e0);
#endif
  def[1][2] = 0.0e0; /* (SGN(y) * mode_values[17]) * z  * 2.0e0; */
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] = mode_values[8]  - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z);
  return 1;
}

/*----------------------------------------------------------------------------
 *
 * Name          : getDM_1_2
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 xy方向に非対象な形状を生成する
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *--------------------------------------------------------------------------*/
int
getDM_1_2(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
          FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_1\n"); */
  def[0][0] = mode_values[6] + mode_values[11] * z;
  def[0][1] = 0.0e0;
  def[0][2] = 0.0e0;
  def[1][0] = 0.0e0;
  def[1][1] = mode_values[6] * 2.0 + mode_values[11] * z;
  def[1][2] = 0.0e0;
  def[2][0] = 0.0e0;
  def[2][1] = 0.0e0;
  def[2][2] = mode_values[8];
  return 1;
}

/*----------------------------------------------------------------------------
 *
 * Name          : getDM_NoSym
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ペントランドらの手法で用いたれたマトリックスを生成
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
```



```
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *--------------------------------------------------------------------------*/
int
getDM_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
            FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_2_NoSym\n"); */
  def[0][0] = mode_values[6]  + mode_values[12] * y + mode_values[15] * z -
    (mode_values[13] + mode_values[16]) * SGN(x) -
      mode_values[14] - mode_values[17];
  def[0][1] = mode_values[11] + (mode_values[13] + SGN(x) * mode_values[14]) * y * 2.
0e0;
  def[0][2] = mode_values[10] + (mode_values[16] + SGN(x) * mode_values[17]) * z * 2.
0e0;
  def[1][0] = mode_values[11] + (mode_values[19] + SGN(y) * mode_values[20]) * x * 2.
0e0;

#if 1
  def[1][1] = mode_values[7]  + mode_values[18] * x + mode_values[21] * z -
    (mode_values[19] + mode_values[22]) * SGN(y) -
      mode_values[20] - mode_values[23];
#endif

  def[1][2] = mode_values[9] + (mode_values[22] + SGN(y) * mode_values[23]) * z  * 2.
0e0;
  def[2][0] = mode_values[10] + (mode_values[25] + SGN(z) * mode_values[26]) * x  * 2
.0e0;
  def[2][1] = mode_values[9] + (mode_values[28] + SGN(z) * mode_values[29]) * y  * 2.
0e0 ;
  def[2][2] = mode_values[8] + mode_values[24] * x + mode_values[27] * y
              - ( mode_values[25] + mode_values[28] ) * 0.5 * SGN(z) -
                mode_values[26] - mode_values[29];
  return 1;
}

/*----------------------------------------------------------------------------
 *
 * Name          : getDM_Apple_Special
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 林檎の形状を生成する
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *--------------------------------------------------------------------------*/
int
getDM_Apple_Special(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                    FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_2_Sym\n"); */
  def[0][0] = mode_values[6]  + mode_values[15] * z -
```

```
    mode_values[17];

  def[0][1] = 0.0e0;
  def[0][2] = (SGN(x) * mode_values[17]) * z * 2.0e0;
  def[1][0] = 0.0e0;

  def[1][1] = mode_values[6]  + mode_values[15] * z -
    mode_values[17];

  def[1][2] = (SGN(y) * mode_values[17]) * z  * 2.0e0;
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] = (sin(sqrt(x*x + y*y) * M_PI / 2.0e0) + 2.0e0 ) / 3.0e0 * mode_values[8]
;
/*
  def[2][2] =   - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z)+ mode_values[8
] *
    (1.0 - exp(-10.0e0 * pow((SGN(x) * x + 0.385987551e0), 2.0e0))) *
      (1.0 - exp(-10.0e0 * pow((SGN(y) * y + 0.385987551e0), 2.0e0)));
*/
  return 1;
}

/*--------------------------------------------------------------------------
 *
 * Name          : getDM_1_Apple
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 やはり林檎の形状を生成する
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *------------------------------------------------------------------------*/
int
getDM_1_Apple(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_2_Sym\n"); */
  def[0][0] = mode_values[6]  + mode_values[15] * z -
    mode_values[17];

  def[0][1] = 0.0e0;
  def[0][2] = (SGN(x) * mode_values[17]) * z * 2.0e0;
  def[1][0] = 0.0e0;

  def[1][1] = mode_values[6]  + mode_values[15] * z -
    mode_values[17];

  def[1][2] = (SGN(y) * mode_values[17]) * z  * 2.0e0;
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] =   - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z)+ mode_values[8
] *
    (1.0 - exp(-10.0e0 * pow((SGN(x) * x + 0.385987551e0), 2.0e0))) *
      (1.0 - exp(-10.0e0 * pow((SGN(y) * y + 0.385987551e0), 2.0e0)));
  return 1;
```



```
}

/*--------------------------------------------------------------------------
 *
 * Name          : getDM_2_Apple
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 林檎の形状を生成する．ただし，２次関数を使用
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *------------------------------------------------------------------------*/
int
getDM_2_Apple(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_2_Sym\n"); */
  def[0][0] = mode_values[6]  + mode_values[15] * z -
    mode_values[17] + mode_values[30] * pow(z, 2.0e0);

  def[0][1] = 0.0e0;
  def[0][2] = (SGN(x) * mode_values[17]) * z * 2.0e0;
  def[1][0] = 0.0e0;

  def[1][1] = mode_values[6]  + mode_values[15] * z - mode_values[17] +
      mode_values[32] * pow(z, 2.0e0);

#if 1
  def[1][2] = (SGN(y) * mode_values[17]) * z  * 2.0e0;
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] = - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z) + mode_values[8]
 *
    (1.0 - exp(-10.0e0 * pow((SGN(x) * x + 0.385987551e0), 2.0e0))) *
      (1.0 - exp(-10.0e0 * pow((SGN(y) * y + 0.385987551e0), 2.0e0)));
#else
  def[2][0] = (mode_values[25]*0) * x  * 2.0e0;
  def[2][1] = (mode_values[25]*0) * y  * 2.0e0 ;
  def[2][2] = -(mode_values[25] + mode_values[25]) * 0 * 0.5 * SGN(z) +  mode_values[
8] *
      (1.0 - exp(-10.0e0 * pow((SGN(x) * x + 0.385987551e0), 2.0e0))) *
        (1.0 - exp(-10.0e0 * pow((SGN(y) * y + 0.385987551e0), 2.0e0)));
#endif
  return 1;
}

/*--------------------------------------------------------------------------
 *
 * Name          : getDM_2_NoSym
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ペントランドらが紹介した変形形態に2次の項を加えたもの
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
```

```
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/
int
getDM_2_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
              FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_2_NoSym\n"); */
  def[0][0] = mode_values[6]  + mode_values[12] * y + mode_values[15] * z -
    (mode_values[13] + mode_values[16]) * SGN(x) -
      mode_values[14] - mode_values[17] +
      mode_values[30] * pow(z, 2.0e0);
  def[0][1] = mode_values[11] + (mode_values[13] + SGN(x) * mode_values[14]) * y * 2.
0e0;
  def[0][2] = mode_values[10] + (mode_values[16] + SGN(x) * mode_values[17]) * z * 2.
0e0;
  def[1][0] = mode_values[11] + (mode_values[19] + SGN(y) * mode_values[20]) * x * 2.
0e0;

#if 1
  def[1][1] = mode_values[7]  + mode_values[18] * x + mode_values[21] * z -
    (mode_values[19] + mode_values[22]) * SGN(y) -
      mode_values[20] - mode_values[23] +
      mode_values[32] * pow(z, 2.0e0);
#endif

  def[1][2] = mode_values[9] + (mode_values[22] + SGN(y) * mode_values[23]) * z  * 2.
0e0;
  def[2][0] = mode_values[10] + (mode_values[25] + SGN(z) * mode_values[26]) * x  * 2
.0e0;
  def[2][1] = mode_values[9] + (mode_values[28] + SGN(z) * mode_values[29]) * y  * 2.
0e0 ;
  def[2][2] = mode_values[8] + mode_values[24] * x + mode_values[27] * y
            - ( mode_values[25] + mode_values[28] ) * 0.5 * SGN(z) -
              mode_values[26] - mode_values[29];
  return 1;
}

/*-------------------------------------------------------------------------
 *
 * Name          : getDM_2_Sym
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ベントランドらが紹介した変形形態に2次の項を加え，軸
 *                 対称な形状を生成するようにしたもの
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/
```



```
int
getDM_2_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_2_Sym\n"); */
  def[0][0] = mode_values[6]  + mode_values[15] * z -
    mode_values[17] + mode_values[30] * pow(z, 2.0e0);

  def[0][1] = 0.0e0;
  def[0][2] = (SGN(x) * mode_values[17]) * z * 2.0e0;
  def[1][0] = 0.0e0;

  def[1][1] = mode_values[6]  + mode_values[15] * z - mode_values[17] +
      mode_values[32] * pow(z, 2.0e0);

  def[1][2] = (SGN(y) * mode_values[17]) * z  * 2.0e0;
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] = mode_values[8]  - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z);
  return 1;
}

/*-------------------------------------------------------------------------
 *
 * Name          : getDM_3_NoSym
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ベントランドらが紹介した変形形態に3次の項を加えたも
 *                 の．
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/
int
getDM_3_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_3_NoSym\n"); */
  def[0][0] = mode_values[6]  + mode_values[12] * y + mode_values[15] * z -
    (mode_values[13] + mode_values[16]) * SGN(x) -
      mode_values[14] - mode_values[17] +
      mode_values[30] * pow(z, 2.0e0) +
        mode_values[31] * pow(z, 3.0e0);
  def[0][1] = mode_values[11] + (mode_values[13] + SGN(x) * mode_values[14]) * y * 2.
0e0;
  def[0][2] = mode_values[10] + (mode_values[16] + SGN(x) * mode_values[17]) * z * 2.
0e0;
  def[1][0] = mode_values[11] + (mode_values[19] + SGN(y) * mode_values[20]) * x * 2.
0e0;

#if 1
  def[1][1] = mode_values[7]  + mode_values[18] * x + mode_values[21] * z -
    (mode_values[19] + mode_values[22]) * SGN(y) -
      mode_values[20] - mode_values[23] +
      mode_values[32] * pow(z, 2.0e0) +
        mode_values[33] * pow(z, 3.0e0);
```

```
#endif

  def[1][2] = mode_values[9] + (mode_values[22] + SGN(y) * mode_values[23]) * z  * 2.
0e0;
  def[2][0] = mode_values[10] + (mode_values[25] + SGN(z) * mode_values[26]) * x  * 2
.0e0;
  def[2][1] = mode_values[9] + (mode_values[28] + SGN(z) * mode_values[29]) * y  * 2.
0e0 ;
  def[2][2] = mode_values[8] + mode_values[24] * x + mode_values[27] * y
              - ( mode_values[25] + mode_values[28] ) * 0.5 * SGN(z) -
                mode_values[26] - mode_values[29];
  return 1;
}

/*----------------------------------------------------------------------------
 *
 * Name          : getDM_3_Sym
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ペントランドらが紹介した変形形態に3次の項を加え，軸
 *                 対称な形状を生成するようにしたもの
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *--------------------------------------------------------------------------*/
int
getDM_3_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_3_Sym\n"); */
  def[0][0] = mode_values[6]  + mode_values[15] * z -
    mode_values[17] + mode_values[30] * pow(z, 2.0e0) + mode_values[31] * pow(z, 3.0e
0);

  def[0][1] = 0.0e0;
  def[0][2] = (SGN(x) * mode_values[17]) * z * 2.0e0;
  def[1][0] = 0.0e0;

  def[1][1] = mode_values[6]  + mode_values[15] * z - mode_values[17] +
      mode_values[30] * pow(z, 2.0e0) + mode_values[31] * pow(z, 3.0e0);

  def[1][2] = (SGN(y) * mode_values[17]) * z  * 2.0e0;
  def[2][0] = (mode_values[25]) * x  * 2.0e0;
  def[2][1] = (mode_values[25]) * y  * 2.0e0 ;
  def[2][2] = mode_values[8]  - ( mode_values[25] + mode_values[25] ) * 0.5 * SGN(z);
  return 1;
}

/*----------------------------------------------------------------------------
 *
 * Name          : getDM
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ３次曲線で表現される形状を生成
 *
 * Arguments     : def[][3]      : 変形マトリックス
```



```
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *--------------------------------------------------------------------------*/
int
getDM(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_1\n"); */
#if 0
  def[0][0] = mode_values[6]  + mode_values[12] * y + mode_values[15] * z
    - ( mode_values[14] + mode_values[17] ) * 0.5;
#else
  def[0][0] = mode_values[6]  + mode_values[12] * y + mode_values[15] * z -
    ( mode_values[14] + mode_values[17] ) * 0.5 +
      mode_values[31] * pow((1.0e0-z), 3.0e0) +
        mode_values[32] * 3.0e0 * z * pow((1.0e0 - z), 2.0e0) +
          mode_values[33] * 3.0e0 * z * z * (1.0e0 - z) +
            mode_values[34] * z * z * z;
#endif

  def[0][1] = mode_values[11] + (mode_values[13] + SGN(x) * mode_values[14]) * y ;
  def[0][2] = mode_values[10] + (mode_values[16] + SGN(x) * mode_values[17]) * z ;
  def[1][0] = mode_values[11] + (mode_values[19] + SGN(y) * mode_values[20]) * x ;

#if 0
  def[1][1] = mode_values[7] + mode_values[18] * x + mode_values[21] * z
    - ( mode_values[20] + mode_values[23] ) * 0.5 ;
#else
  def[1][1] = mode_values[7] + mode_values[18] * x + mode_values[21] * z -
    ( mode_values[20] + mode_values[23] ) * 0.5+
      mode_values[35] * pow((1.0e0-z), 3.0e0) +
        mode_values[36] * 3.0e0 * z * pow((1.0e0 - z), 2.0e0) +
          mode_values[37] * 3.0e0 * z * z * (1.0e0 - z) +
            mode_values[38] * z * z * z;
#endif

  def[1][2] = mode_values[9] + (mode_values[22] + SGN(y) * mode_values[23]) * z  ;
  def[2][0] = mode_values[10] + (mode_values[25] + SGN(z) * mode_values[26]) * x ;
  def[2][1] = mode_values[9] + (mode_values[28] + SGN(z) * mode_values[29]) * y  ;
  def[2][2] = mode_values[8] + mode_values[24] * x + mode_values[27] * y
              - ( mode_values[26] + mode_values[29] ) * 0.5;

  return 1;
}

/*----------------------------------------------------------------------------
 *
 * Name          : getDM_Sym
 *
 * Description   : 変形形態パラメータと位置から変形マトリックスを求める
 *                 ３次曲線で表現される形状を生成，軸対称な形状を生成
 *
 * Arguments     : def[][3]      : 変形マトリックス
 *                 mode_values   : 変形形態に対応するパラメータの値
 *                 x,y,z         : 変形マトリックスを作用させたい曲面上の点の位置
 *
 * Returns       : On failure returns 0.
```

```
 *
 * Author       : M.Yoshida
 *
 * Date         :
 *
 * Changes      :
 *
 *-----------------------------------------------------------------------------*/
int
getDM_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
          FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z)
{
  /*   fprintf(stderr, "my_mode: getDM_1\n"); */
#if 0
  def[0][0] = mode_values[6]  - mode_values[10] / 2.0e0 +
    mode_values[11] * pow((1.0e0-z), 3.0e0) +
      mode_values[12] * 3.0e0 * z * pow((1.0e0 - z), 2.0e0) +
        mode_values[13] * 3.0e0 * z * z * (1.0e0 - z) +
          mode_values[14] * z * z * z;
#else
  def[0][0] = mode_values[6]  - mode_values[10] / 2.0e0 +
    mode_values[11] * z;
#endif
  def[0][1] = 0.0e0;
  def[0][2] = SGN(x) * mode_values[10] * z ;
  def[1][0] = 0.0e0;
#if 0
  def[1][1] = mode_values[6] - mode_values[10] / 2.0e0 +
    mode_values[11] * pow((1.0e0-z), 3.0e0) +
      mode_values[12] * 3.0e0 * z * pow((1.0e0 - z), 2.0e0) +
        mode_values[13] * 3.0e0 * z * z * (1.0e0 - z) +
          mode_values[14] * z * z * z;
#else
  def[1][1] = mode_values[6] - mode_values[10] / 2.0e0 +
    mode_values[11] * z;
#endif
  def[1][2] = SGN(y) * mode_values[10] * z  ;
  def[2][0] = mode_values[9] * x ;
  def[2][1] = mode_values[9] * y  ;
  def[2][2] = mode_values[8];

  return 1;
}

/*-----------------------------------------------------------------------------
 *
 * Name          : getDeformationModeNum
 *
 * Description   : return the number of modes required.
 *
 * Arguments     : domain : to specify the id of domain
 *
 * Returns       : Normally returns the number of mode of domain.
 *
 * Author        : M.Yoshida
 *
 * Date          : 21:40:00 @ 01Dec1995
 *
 * Changes       :
 *
 *-----------------------------------------------------------------------------*/
int
getDeformationModeNum(int domain)
{
  return gDomainInfo[domain].nmodes;
}
```



```
/*---*/
```

```
/*
 * my_mode.h - include for my_main
 *
 * File names and variables for OVO Genesis,
 *  Copyright (C) 1994 ATR Communication Systems Research Laboratories.
 *
 */
#ifndef _H_MY_MODE
#define _H_MY_MODE

/* Prototype Definitions
 */
extern int getDM_RBF(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                     FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_Slant(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                       FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_Car(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                     FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_1_2(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                     FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                       FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_Apple_Special(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                               FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_1_Apple(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                    FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_2_Apple(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                    FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_2_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                    FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_2_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                    FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_3_NoSym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                    FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int getDM_3_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                    FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int      getDM(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);
extern int      getDM_Sym(FLOAT_TYPE def[][3], FLOAT_TYPE* mode_values,
                      FLOAT_TYPE x, FLOAT_TYPE y, FLOAT_TYPE z);

#endif /* !_H_MY_MODE */
```

```
/*
 *
 * NAME:        my_phi.c
 *
 * Descriptions: 変形パラメータに対する曲面上の点の変化を求める
 *
 * ---------------------------------------------------------------------------
 *
 *
 */
#include "sq.h"

/* Prototype Declaration
 */
int     compute_phi_multi(SUPERQUAD* sq);

/*---*/
/*-------------------------------------------------------------------------------
 *
 * Name          : set_mode
 *
 * Description   : ドメインに対応する変形形態関数をセットする.
 *
 * Arguments     : domain : ドメインＩＤ
 *
 * Returns       : On failure 0 return.
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------------*/
int
compute_phi_multi(SUPERQUAD* sq)
{
  POINT3D*      original_points;
  FLOAT_TYPE    angles[3];      /* to store sq->angle */
  register      int             mode_number;
  register      int             i;
  FLOAT_TYPE    scale = 100.0;
  register      FLOAT_TYPE*     phi;
  register      FLOAT_TYPE*     p0;
  register      FLOAT_TYPE*     p1;

  /* phi matrixis is computed in unrotated coordinate frame */
  angles[0] = sq->x_angle;
  angles[1] = sq->y_angle;
  angles[2] = sq->z_angle;
  sq->rotate(sq, 0.0e0, 0.0e0, 0.0e0);

  /* set Deformed points (to get unrotated positions of points) */
  sq->deform(sq);

  /* allocate data segment for phi matrix */
  if(sq->phi == NULL)
    {
      sq->phi = make_matrix_type(sq->n_points * 3, sq->n_modes,
                                 "multi mode phi matrix");
    }
  else if(sq->phi->w != sq->n_points*3)
    {
      free_matrix_type(sq->phi);
      sq->phi = make_matrix_type(sq->n_points * 3,
                                 sq->n_modes,
```



```
                                 "multi mode phi matrix");
    }

  /* set originapo points coordinates */
  original_points = (POINT3D *)calloc(sq->n_points, sizeof(POINT3D));
  (void)memcpy((char*)original_points, (char*)sq->deformed_points, sizeof(POINT3D)*sq
->n_points);

  phi = sq->phi->m;
  matrix_zero(sq->phi);
  for(i = 0; i < sq->n_points; i++)
    {
      phi[0] = 1.0e0;                   /* phi[0][0] */
      phi[1 + sq->phi->w] = 1.0e0;      /* phi[1][1] */
      phi[2 + 2 * sq->phi->w] = 1.0e0;  /* phi[2][2] */
      phi += 3;
    }

  /* phi += 2 * sq->phi->w;     /* phi -> sq->phi->m[3][0] */
  phi = sq->phi->m;
  for(mode_number = 0; mode_number < sq->n_modes; mode_number++)
    {
      if(mode_number == 0 ||
         mode_number == 1 ||
         mode_number == 2 ||
         mode_number == 3 ||
         mode_number == 4 ||
         mode_number == 5 ||
         mode_number == 8
         )
        {
          /* フィッティングの過程では，z軸方向のサイズを変更しないようにしている.
           *  mode_number 8 はz軸方向のサイズに対応している.
           */
          ;
        }
      else
        {
          /*
           *  du = 1/100
           *  有限差分により微分値を得る.
           *  変形パラメータに与える差分は0.01に固定している.
           */
          sq->modes[mode_number] += 1.0 / scale;
        }

      if(mode_number < 6)
        {
          sq->rotate(sq, sq->modes[3], sq->modes[4], sq->modes[5]);
        }

      sq->deform(sq);

      p0 = (FLOAT_TYPE *)original_points;
      p1 = (FLOAT_TYPE *)sq->deformed_points;
      for(i = 0; i < sq->n_points; i++)
        {
          /*
           * p0 = Duf(x)
           * p1 = (Du + dDu) f(x)
           * p1 - p0 = (Du + du)f(x) - Du f(x) = dDu f(x)
           * (p1-p0)/du = dDuf(x) / du = phi f(x)
           */
          phi[0] = (p1[0] - p0[0]) * scale;
          phi[1] = (p1[1] - p0[1]) * scale;
          phi[2] = (p1[2] - p0[2]) * scale;
```

```
          phi += 3;
          p1  += 3;
          p0  += 3;
        }
      if(mode_number == 0 ||
         mode_number == 1 ||
         mode_number == 2 ||
         mode_number == 3 ||
         mode_number == 4 ||
         mode_number == 5 ||
         mode_number == 8)
        {
          ;
        }
      else
        {
          sq->modes[mode_number] -= 1.0 / scale;
        }
      /* rotation is special
       */
      if(mode_number < 6)
        {
          sq->rotate(sq, sq->modes[3],sq->modes[4],sq->modes[5]);
        }
      sq->deform(sq);
    }

  /* restore superquadrics angles */
  sq->rotate(sq, angles[0], angles[1], angles[2]);
  sq->deform(sq);

  /* free up some memory */
  free(original_points);

  return 1;
}
/*---*/
```

```
/*
 *
 * NAME:        my_phi.c
 *
 * Descriptions: 変形パラメータに対する曲面上の点の変化を求める
 *
 * ----------------------------------------------------------------------------
 *
 *
 */
#include "sq.h"

/* Prototype Declaration
 */
int     compute_phi_multi(SUPERQUAD* sq);

/*---*/
/*----------------------------------------------------------------------------
 *
 * Name         : set_mode
 *
 * Description  : ドメインに対応する変形形態関数をセットする.
 *
 * Arguments    : sq : 超２次関数に関するデータのレコード
 *
 * Returns      : On failure 0 return.
 *
 * Author       : M.Yoshida
 *
 * Date         :
 *
 * Changes      :
 *
 *----------------------------------------------------------------------------*/
int
compute_phi_multi(SUPERQUAD* sq)
{
  POINT3D*      original_points;
  FLOAT_TYPE    angles[3];      /* to store sq->angle */
  register      int             mode_number;
  register      int             i;
  FLOAT_TYPE    scale = 100.0;
  register      FLOAT_TYPE*     phi;
  register      FLOAT_TYPE*     p0;
  register      FLOAT_TYPE*     p1;

  /* phi matrixis is computed in unrotated coordinate frame */
  angles[0] = sq->x_angle;
  angles[1] = sq->y_angle;
  angles[2] = sq->z_angle;
  sq->rotate(sq, 0.0e0, 0.0e0, 0.0e0);

  /* set Deformed points (to get unrotated positions of points) */
  sq->deform(sq);

  /* allocate data segment for phi matrix */
  if(sq->phi == NULL)
    {
      sq->phi = make_matrix_type(sq->n_points * 3, sq->n_modes,
                                 "multi mode phi matrix");
    }
  else if(sq->phi->w != sq->n_points*3)
    {
      free_matrix_type(sq->phi);
      sq->phi = make_matrix_type(sq->n_points * 3,
                                 sq->n_modes,
```



```
                                 "multi mode phi matrix");
    }

  /* set originapo points coordinates */
  original_points = (POINT3D *)calloc(sq->n_points, sizeof(POINT3D));
  (void)memcpy((char*)original_points, (char*)sq->deformed_points, sizeof(POINT3D)*sq
->n_points);

  phi = sq->phi->m;
  matrix_zero(sq->phi);
  for(i = 0; i < sq->n_points; i++)
    {
      phi[0] = 1.0e0;                   /* phi[0][0] */
      phi[1 + sq->phi->w] = 1.0e0;      /* phi[1][1] */
      phi[2 + 2 * sq->phi->w] = 1.0e0;  /* phi[2][2] */
      phi += 3;
    }

  /* phi += 2 * sq->phi->w;     /* phi -> sq->phi->m[3][0] */
  phi = sq->phi->m;
  for(mode_number = 0; mode_number < sq->n_modes; mode_number++)
    {
      if(mode_number == 0 ||
         mode_number == 1 ||
         mode_number == 2 ||
         mode_number == 3 ||
         mode_number == 4 ||
         mode_number == 5 ||
         mode_number == 8
         )
        {
          /* フィッティングの過程では，z軸方向のサイズを変更しないようにしている.
           *  mode_number 8 はz軸方向のサイズに対応している.
           */
          ;
        }
      else
        {
          /*
           *  du = 1/100
           *  有限差分により微分値を得る.
           *  変形パラメータに与える差分は0.01に固定している.
           */
          sq->modes[mode_number] += 1.0 / scale;
        }

      if(mode_number < 6)
        {
          sq->rotate(sq, sq->modes[3], sq->modes[4], sq->modes[5]);
        }

      sq->deform(sq);

      p0 = (FLOAT_TYPE *)original_points;
      p1 = (FLOAT_TYPE *)sq->deformed_points;
      for(i = 0; i < sq->n_points; i++)
        {
          /*
           * p0 = Duf(x)
           * p1 = (Du + dDu) f(x)
           * p1 - p0 = (Du + du)f(x) - Du f(x) = dDu f(x)
           * (p1-p0)/du = dDuf(x) / du = phi f(x)
           */
          phi[0] = (p1[0] - p0[0]) * scale;
          phi[1] = (p1[1] - p0[1]) * scale;
          phi[2] = (p1[2] - p0[2]) * scale;
```

```
          phi += 3;
          p1  += 3;
          p0  += 3;
        }
      if(mode_number == 0 ||
         mode_number == 1 ||
         mode_number == 2 ||
         mode_number == 3 ||
         mode_number == 4 ||
         mode_number == 5 ||
         mode_number == 8)
        {
          ;
        }
      else
        {
          sq->modes[mode_number] -= 1.0 / scale;
        }
      /* rotation is special
       */
      if(mode_number < 6)
        {
          sq->rotate(sq, sq->modes[3],sq->modes[4],sq->modes[5]);
        }
      sq->deform(sq);
    }

  /* restore superquadrics angles */
  sq->rotate(sq, angles[0], angles[1], angles[2]);
  sq->deform(sq);

  /* free up some memory */
  free(original_points);

  return 1;
}
/*---*/
```

```
/*
 *
 * Name:                my_fit_ellipsoid.c
 * Descriptions:        離散点集合に曲面を近づけるための関数群
 *
 * ------------------------------------------------------------------------------
 */

#include "sq.h"

/* Prototype Declaration
 */
int     find_sq_ellipsoid_xy(SUPERQUAD* sq, POINT3D* data_points,
                             int n_data_points);

int     find_ellipsoid_xy(POINT3D* points,              /* data_points */
                          int n_points,         /* the number of points int */
                          POINT3D* center_of_gravity,/* *center of gravity float cent
er[3]*/
                          FLOAT_TYPE R[3][3],   /* *Rotation float matrix[3][3] */
                          FLOAT_TYPE* angles,   /* *angles float angles[3] */
                          FLOAT_TYPE* bounds,   /* *bounds? POINT3D bounds */
                          FLOAT_TYPE* det);     /* determinant */
int     compute_principal_axes_xy(POINT3D* points, int n_points,
                                  POINT3D* origin, FLOAT_TYPE R[3][3]);

/* static variables */
static POINT3D  sMin, sMax; /* local coordinate */
static POINT3D  sWMin, sWMax; /* world coordinate */
static POINT3D  sBounds;

/*---*/
/*------------------------------------------------------------------------------
 *
 * Name         : find_sq_ellipsoid_xy
 *
 * Description  :
 *
 * Arguments    : sq            : 超２次関数レコード
 *                angles        : 回転角(Rad)
 *                data_points   : 離散点集合
 *                n_data_points : 離散点集合の点の個数
 *
 * Returns      :
 *
 * Author       : M.Yoshida
 *
 * Date         :
 *
 * Changes      :
 *
 *----------------------------------------------------------------------------*/
int
find_sq_ellipsoid_xy(SUPERQUAD* sq, POINT3D* data_points, int n_data_points)
{
  FLOAT_TYPE    r[3][3];
  FLOAT_TYPE    angles[3];
  FLOAT_TYPE    center[3];
  FLOAT_TYPE    det;
  FLOAT_TYPE    _w;
  /* find the initial ellipsoid which fits the point data */
  find_ellipsoid_xy(data_points,
                    n_data_points,
                    (POINT3D*)center,   /* center of gravity */
                    r,          /* rotation matrix form "principal axis"
                                 * compute_privipal_axes(points, n_points, center, R)
```



```
                                 */
                    angles,     /* decode_rotation_matrix
                                 *      yaw, pitch, roll
                                 */
                    (FLOAT_TYPE*)&sBounds,      /* ax, ay, az (radius?) az=sBounds[2]
 = max
                                                    */
                    &sq->determinant);

  /* copy results into the local sq data structure    */
  /* transはワールド座標 */
  /* 14/Dec/1995
   * 重心をtranslateとするとをやめる。分布が偏っているため、
   * 中心と重心が大きくずれてしまう.
   */
  {
    FLOAT_TYPE  _center[3];
    FLOAT_TYPE  _dir[3];
    double      _w;
    FLOAT_TYPE  _l;
    int         ax;
    _center[0] = (sWMin.x + sWMax.x) / 2.0e0;
    _center[1] = (sWMin.y + sWMax.y) / 2.0e0;
    _center[2] = (sWMin.z + sWMax.z) / 2.0e0;
    /* supposing r[0], r[1] has been normalized already */
    if(fabs(r[0][1]) <= fabs(r[1][1]))  ax = 1;
    else                                ax = 0;
    sq->hand_axis = 1;
    _dir[0] = r[0][ax],    _dir[1] = r[1][ax],    _dir[2] = r[2][ax];
    if(_dir[1] > 0.0e0)
      {
        _dir[0] *= -1.0e0;
        _dir[1] *= -1.0e0;
        _dir[2] *= -1.0e0;
      }
    if(ax == 0)         _l = (sMax.x - sMin.x) / 2.0e0;
    else                _l = (sMax.y - sMin.y) / 2.0e0;
    _dir[0] *=  _l;
    _dir[1] *=  _l;
    _dir[2] *=  _l;
    _center[0] += _dir[0];
    _center[1] += _dir[1];
    _center[2] += _dir[2];
    /* 長軸:ワールドX軸:近 */
    fprintf(stderr, "r00 >= r01 X%f x%f Y%f y%f\n",
            sMax.x, sMin.x, sMax.y, sMin.y);
    sq->translate(sq, _center[0], _center[1], _center[2]);
  }

  decode_rotation_matrix(r, angles);
  sq->rotate(sq, angles[0], angles[1], angles[2]);
  sq->scale(sq, sBounds.x, sBounds.y, sBounds.z);

  /* update the deformed polygons on the sq */
  sq->deform(sq);
  return 1;
}

/*------------------------------------------------------------------------------
 *
 * Name         : find_ellipsoid_xy
 *
 * Description  : 慣性主軸から変換のためのマトリックスを求める.
 *
 * Arguments    : points        : 離散点集合
 *                n_points      : 離散点集合の点の個数
```

```
 *                center_of_gravity :
 *                                  離散点集合の単純重心座標
 *                angles            : 回転角度(Rad)
 *                sBounds           : 離散点集合の空間における範囲
 *                sq_det            : 主軸からなる座標系からワールド座標系
 *                                    への変換マトリックスのデターミナント
 * Returns      :
 *
 * Author       : M.Yoshida
 *
 * Date         :
 *
 * Changes      :
 *
 *-------------------------------------------------------------------------------*/
int
find_ellipsoid_xy(POINT3D* points,              /* data_points */
                  int n_points,         /* the number of points int */
                  POINT3D* center_of_gravity,/* *center of gravity float center[3]*/
                  FLOAT_TYPE R[3][3],   /* *Rotation float matrix[3][3] */
                  FLOAT_TYPE* angles,   /* *angles float angles[3] */
                  FLOAT_TYPE* sBounds,  /* *sBounds? POINT3D sBounds */
                  FLOAT_TYPE* sq_det)   /* determinant */
{
  register int          i, j;
  register FLOAT_TYPE   x, y, z, a;
  FLOAT_TYPE            IR[3][3];
  FLOAT_TYPE            det;

  /*
   * Find the center of gravity (simple averaging)
   */
  center_of_gravity->x = center_of_gravity->y = center_of_gravity->z = 0.0e0;
  sWMin.x = sWMax.x = points[0].x;
  sWMin.y = sWMax.y = points[0].y;
  sWMin.z = sWMax.z = points[0].z;
  for(i = 0; i < n_points; i++)
    {
      center_of_gravity->x += points[i].x;
      center_of_gravity->y += points[i].y;
      center_of_gravity->z += points[i].z;
      MINMAX(sWMin.x,sWMax.x,points[i].x);
      MINMAX(sWMin.y,sWMax.y,points[i].y);
      MINMAX(sWMin.z,sWMax.z,points[i].z);
    }
  center_of_gravity->x /= (FLOAT_TYPE)n_points;
  center_of_gravity->y /= (FLOAT_TYPE)n_points;
  center_of_gravity->z /= (FLOAT_TYPE)n_points;

  /*
   * Compute the principal axes (rotation matrix)
   */
  compute_principal_axes_xy(points, n_points,
                            center_of_gravity,
                            R); /* R indeicate axis vectors in columns */

  /*
   * Check the sign of the determinant...if negative, then negate the matrix entries
   * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   */
#if 1
  {
    double det;

    /* det(R) = +(-)1.0 ? */
    if((det = DETERMINANT_3x3(R)) < 0.0e0)
```



```
      {
        SCALE_MATRIX_3x3(R,-1.0);
      }
    *sq_det = det;
    fprintf(stderr, "find_ellipsiod @ fit_ellipsoid.c: %lf\n", det);
  }
#endif

  /* Use the rotation matrix to determine the x,y,z sBounds
   *
   */
#if USE_ROT
  UNROTATE_POINT(points[0].x, points[0].y, points[0].z,
                 R, sMin.x, sMin.y, sMin.z);
#else /* USE_ROT */

  INVERSE_MATRIX_3x3(IR, R, det);
  ROTATE_POINT(points[0].x,
               points[0].y,
               points[0].z,
               IR,                          /* By M.Yoshida */
               sMin.x, sMin.y, sMin.z);
#endif /* USE_ROT */

  sMax.x = sMin.x;
  sMax.y = sMin.y;
  sMax.z = sMin.z;

  for(i = 1 ; i < n_points; ++i)
    {
#if USE_ROT
      UNROTATE_POINT(points[i].x, points[i].y, points[i].z,
                     R, sMin.x, sMin.y, sMin.z);
#else /* USE_ROT */
      ROTATE_POINT(points[i].x,
                   points[i].y,
                   points[i].z,
                   IR,                      /*  By M.Yoshida */
                   x, y, z);
#endif /* USE_ROT */
      MINMAX(sMin.x,sMax.x,x);
      MINMAX(sMin.y,sMax.y,y);
      MINMAX(sMin.z,sMax.z,z);
    }
  sBounds[0] = 0.5e0 * (sMax.x - sMin.x);
  sBounds[1] = 0.5e0 * (sMax.y - sMin.y);
  sBounds[2] = 0.5e0 * (sMax.z - sMin.z);

#ifdef TEST
  fprintf(stderr, "Range: % -012.5f, % -012.5f, % -012.5f\n",
          sBounds[0], sBounds[1], sBounds[2]);
#endif /* TEST */

  /* Choose largest bound for Z axis
   *
   * The Shortest bound -> X axis
   *
   */
  /* original */
  j = 0 ;
  if(sBounds[0] > sBounds[1])
    {
      fprintf(stderr, "0 has max bouds\n");
    }
  else
    {
```

```
      fprintf(stderr, "2 has max bouds\n");
      j = 1;
    }

  if(j != 0)
    {
      for(i = 0 ; i < 3 ; ++i)
        {
          a = R[i][j];
          R[i][j] = R[i][0];
          R[i][0] = a;
        }
      a = sBounds[j];
      sBounds[j] = sBounds[0];
      sBounds[0] = a;
    }

  /* Get the rotation angles from the principal axes
   * R->angles */
  decode_rotation_matrix(R, angles);
}

/*-------------------------------------------------------------------------
 *
 * Name          : find_ellipsoid_xy
 *
 * Description   : 慣性主軸を求める
 *
 * Arguments     : points        : 離散点集合
 *                 n_points      : 離散点集合の点の個数
 *                 origin        : 離散点集合の単純重心座標
 *                 R             : 固有ベクトル
 *                                 慣性主軸方向のベクトル
 * Returns       :
 *
 * Author        : M.Yoshida
 *
 * Date          :
 *
 * Changes       :
 *
 *-------------------------------------------------------------------------*/
int
compute_principal_axes_xy(POINT3D* points, int n_points,
                          POINT3D* origin, FLOAT_TYPE R[3][3])
{
  register int  i;
  register int  j;
  FLOAT_TYPE    x;
  FLOAT_TYPE    y;
  FLOAT_TYPE    z;
  FLOAT_TYPE    x2;
  FLOAT_TYPE    y2;
  FLOAT_TYPE    z2;
  FLOAT_TYPE    a00;
  FLOAT_TYPE    a11;
  FLOAT_TYPE    a22;
  FLOAT_TYPE    a01;
  FLOAT_TYPE    a02;
  FLOAT_TYPE    a12; /* U */
  FLOAT_TYPE    lambdas[3];
  FLOAT_TYPE    a[3][3];
  matrix_type*  A;
  matrix_type*  Axes;

  /* Given a cloud of points, find the principal axes and lambdas */
```



```
  /* Done by finding the mass moments of inertia */
  /*
   * Task is simplified by the assumption that each point has equal mass
   * ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   *
   * Axes->m = R[3][3]  R = not determined.
   *
   */
  Axes = make_matrix_type_w_data(3, 3, /* w = 3, h = 3 */
                                 "ellipsoid rotation matrix",
                                 R);    /* data R: 3 x 3 */
  /*
   * A->m = a[3][3]
   */
  A = make_matrix_type_w_data(3, 3,
                              "moment matrix for ellispoid fitting",
                              a);       /* a : 3 x 3 */
  /* A->m = {0}
   */
  matrix_zero(A);

  /*
   * A: axis of interialを求めるためのmatrix
   *
   */
  for(i = 0; i < n_points; ++i)
    {
      x = points[i].x - origin->x;
      y = points[i].y - origin->y;
      /*       z = (i % 2) ? 0.1e0 : -0.1e0; */
      z = 0.0e0;
      x2 = x*x;
      y2 = y*y;
      z2 = z*z;
      a[0][0] += (y2 + z2);     /* 断面極2次モーメント*/
      a[1][1] += (x2 + z2);
      a[2][2] += (x2 + y2);
      a[0][1] -= x * y;         /* 断面相乗モーメント */
      a[0][2] -= x * z;
      a[1][2] -= y * z;
    }

  /* fill in moment matrix
   *
   * A->m is a symmetric matrix
   */
  a[1][0] = a[0][1];
  a[0][2] = a[2][0];
  a[1][2] = a[2][1];

  /* obtain eigenvectors and eigenvalues
   *
   * A->m 's eigenvectors is principal axes.
   * EispackRealSym is only for real number symmetrical matrix
   *
   */
  EispackRealSym(A, lambdas, Axes ); /* matrix_eispack.c */
#ifdef TEST
  {
    register int        i;
    register int        j;
    FLOAT_TYPE          a;

    for(i = 0; i < 3; i++)
      {
        for(j = 0; j < 3; j++)
```

```
          {
            fprintf(stderr, "% -012.5f\t", R[i][j]);
          }
        fprintf(stderr, "\n");
      }
    fprintf(stderr, "inner: % -012.5f\n", R[0][0] * R[0][1] + R[1][0] * R[1][1]);
    fprintf(stderr, "size 1: % -012.5f \t % -012.5f\n",
            R[0][0] * R[0][0] + R[1][0] * R[1][0],
            R[0][1] * R[0][1] + R[1][1] * R[1][1]);
#if 0
    if(R[0][1] > EPSILON)
      {
        if(R[0][1] * R[1][0] < 0.0e0)
          {
            for(i = 0; i < 3; i++)
              {
                a = R[i][0];
                R[i][0] = R[i][1];
                R[i][1] = a;
              }
          }
      }
#endif

  }
#endif /* TEST */
  free_matrix_type(A);
  free_matrix_type(Axes);
}
/*---*/
```

```
#include <stdio.h>
#include "y.tab.h"
#include "hand.h"

extern int yynerrs;
extern FILE* yyin;
int dmpflag = 0;
#ifdef YYDEBUG
extern int yydebug;
#endif
/*
 *      interpret the expression
 */
main(int argc, char** argv)
{
  int   goon;
  if(argc >= 2 && !strcmp(argv[1], "-d"))
    {
      dmpflag = 1;
#ifdef YYDEBUG
      yydebug  = 1;
#endif /* YYDEBUG */
    }
  goon = 1;
  while(goon)
    {
      if(yyparse() || yynerrs)
        {
         /*  fatal("syntax error"); */
          fprintf(stderr, "syntax error");
          myfprintf(stdout, NOGOOD);
        }
      else
        {
          goon = 0;
        }
    }
}

/*
 *      cause the fatal error
 */
fatal(char* s)
{
  fprintf(stderr, "%s!!!.\n", s);
  exit(0);
}
```

```
#ifndef _H_HAND
#define _H_HAND

#include "../SRC/my_main.h"

#define NOGOOD          "ERROR\n"
#define GOOD            "OK\n"
#define myfprintf(ARG1,ARG2)    {fprintf((ARG1),(ARG2));fflush(stdout);}

#endif /* !_H_HAND */
```

```
%{
/**
 **      Shape Generator Lexical Analyzer
 **/
#include "y.tab.h"
#include "hand.h"
#define STRSIZE         (512)
int     strcnt = 0;
char    string[STRSIZE + 1] = {0};
%}
%start STR COMM
ND      [a-zA-Z_]
DG      [0-9]
NZ      [1-9]
SN      [+-]
FS      [flFL]
EP      ([eE]{SN}?{DG}+)
FC      (({DG}*"."{DG}+)|({DG}+"."))
%%
<INITIAL>L?\"           {strcnt = 0;
                         BEGIN STR;
/*                       fprintf(stderr, "Begin STR\n");*/
                        }
<INITIAL>"/*"           {BEGIN COMM;
/*                       fprintf(stderr, "Begin COMM\n"); */
                        }
<INITIAL>[ \t]          ;
<INITIAL>\n             {return NL;}
<INITIAL>{ND}({ND}|{DG})* {  int token;
/*                           fprintf(stderr, "lex:%s", yytext); */
                             if(token = reserve(yytext, &yylval.type))
                               {
/*                               fprintf(stderr, "<%03d>\n", token); */
                                 return token;
                               }
/*                           fprintf(stderr, "<fail>\n"); */
                            }
<INITIAL>{SN}{DG}{DG}* |
<INITIAL>{DG}{DG}*      {yylval.name = yytext;/* !!! */
                         return INTEGER;}
<INITIAL>{SN}?{FC}{EP}?{FS}? |
<INITIAL>{SN}?{DG}+{EP}{FS}? {yylval.name = yytext;
                         return CONSTANT;}
<INITIAL>";"            {return SM;}
<STR>\"                 {BEGIN INITIAL;
                         if(strcnt >= STRSIZE)
                           fprintf(stderr, "string too long");
                         string[strcnt] = '\0';
                         yylval.name = string;
/*                       fprintf(stderr, "STRING: %s\n", yylval.name); */
                         BEGIN INITIAL;
                         return STRING;
                         }
<STR>\\\"               {if(strcnt  < STRSIZE)
                           string[strcnt++] = '"';
                        }
<STR>\\n                {if(strcnt < STRSIZE)
                           string[strcnt++] = '\n';
                        }
<STR>\n                 {fprintf(stderr, "string not closed");
                         string[strcnt] = '\0';
                         yylval.name = string;
                         BEGIN INITIAL;
                         return STRING;}
<STR>.                  {if(strcnt < STRSIZE)
                           string[strcnt++] = yytext[0];
                        }
<COMM>"*/"              {BEGIN INITIAL;}
<COMM>\n                ;
<COMM>.                 ;
.                       ;
%%
struct rsvword {
  char*         name;
  int           token;
  int           type;
} rsvtable[]  = {
  "computephi",         COMPUTEPHI,     0,
  "directexponents",    DIRECTEXPONENTS,0,
  "directposition",     DIRECTPOSITION, 0,
  "directrotate",       DIRECTROTATE,   0,
  "directscale",        DIRECTSCALE,    0,
  "drawsq",             DRAWSQ,         0,
  "echo",               ECHOL,          0,
  "end",                END,            0,
  "fitellipsoid",       FITELLIPSOID,   0,
  "fitmodel",           FITMODEL,       0,
  "genpolygons",        GENPOLYGONS,    0,
  "help",               HELP,           0,
  "init",               INIT,           0,
  "initglobal",         INITGLOBAL,     0,
  "inquireexponents",   INQEXPONENTS,   0,
  "inquirescale",       INQSCALE,       0,
  "mappoints",          MAPPOINTS,      0,
  "projectpoints",      PROJECTPOINTS,  0,
  "promptsq",           PROMPTSQ,       0,
  "read",               READ,           0,
  "report",             REPORT,         0,
  "setdomain",          SETDOMAIN,      0,
  "setexponents",       EXPONENTS,      0,
  "setmode",            SETMODE,        0,
  "setnormals",         SETNORMALS,     0,
  "setradius",          SETRADIUS,      0,
  "settopshape",        SETTOPSHAPE,    0,
};

#define RSVTABSIZE      sizeof(rsvtable) / sizeof(struct rsvword)

/*
 *      find the reserved word
 */

int
reserve(char* text, int*type)
{
  int   low;
  int   high;
  int   mid;
  int   ret;

  for(low = 0, high = RSVTABSIZE; low < high; )
    {
      mid = (low + high) / 2;
      ret = strcmp(rsvtable[mid].name, text);
      if(ret < 0)
        {
          low = mid + 1;
        }
      else if (ret > 0)
        {
          high = mid;
        }
```

```
      else
        {
/*        fprintf(stderr, "lex: %s\n", text); */
          *type = rsvtable[mid].type;

          return rsvtable[mid].token;
        }
    }
  return 0;
}

/*---*/
```

```
%{
/**
 **             Shape Generator Syntax Analyzer
 **
 **             !!!Caution!!!
 ** I have mistaken to develop this yacc file. This faunctionality which is realized
by this source
 ** must be realized with more easy source, e.g., using FVALUE, INTEGER, ... etc.,
 ** I am nearly new in yacc programing, so I wanted to try manything without regardin
g costs.
 ** Please modify this source largely, not locally! if you have to use this yacc sour
ce file.
 ** I don't have any power to repair this source, although I know how to repair this
basically.
 ** Have nice programing.
 **/
#include <stdio.h>
#include <string.h>
#include "hand.h"
#define NODESIZE        (64)
struct nlist {
  struct nlist* last;
  struct nlist* next;
  char*         str;
};
%}
%union {
  int   type;
  char* name;
  struct nlist* listp;
}
%type  <type> command
%token <type> READ INIT FITELLIPSOID INITGLOBAL SETDOMAIN SETTOPSHAPE COMPUTEPHI
%token <type> DRAWSQ GENPOLYGONS REPORT END EXPONENTS FITMODEL MAPPOINTS PROMPTSQ HEL
P
%token <type> DIRECTSCALE DIRECTEXPONENTS INQEXPONENTS INQSCALE SETMODE ECHOL
%token <type> DIRECTPOSITION DIRECTROTATE SETNORMALS SETRADIUS PROJECTPOINTS
%token <type> SM NL
%type  <listp> expression.list
%type  <name> expr
%token <name> CONSTANT STRING INTEGER
%%
/**
 **     statements
 **/
s       : command.list                  { fprintf(stderr, "bye-bye\n"); }
        ;
command.list
        : command NL                    {if(nodearea)
                                           {
                                             (void)memset((char*)nodearea, '\0',
                                                          nodesize * sizeof(struct nl
ist));
                                             free(nodearea);
                                           }
                                         nodearea = NULL; nodesize = 0; nodecnt = 0;
                                         fprintf(stderr, "FREE\n");
                                        }
        | command.list command  NL
        ;
command
        : expression.list               {fprintf(stderr, "");}
        | END                           {fprintf(stderr, "bye-bye\n");
                                         YYACCEPT;}
        | ECHOL expression.list         {if($2 == NULL || $2->str == NULL)
                                           {
```



```
                                             fprintf(stdout, "\n");
                                           }
                                         else
                                           {
                                             fprintf(stderr, "%s\n", $2->str);
                                           }
                                         $$ = $1;
                                        }
        | READ expression.list          {if($2 == NULL || $2->str == NULL)
                                           {
                                             yyerror("read syntax error\n");
                                             myfprintf(stdout, NOGOOD);
                                             /* YYERROR; */
                                           }
                                         else
                                           {
                                             if(!collectGestureData($2->str))
                                               {
                                                 fprintf(stderr,
                                                         "Failed to collectGestureDa
ta\n");
                                                 myfprintf(stdout, NOGOOD);
                                               }
                                             else
                                               myfprintf(stdout, GOOD);
                                           }
                                         $$ = $1;
                                        }
        | INIT expression.list          { int n;
                                          struct nlist* p;
                                          char* args[NODESIZE];
                                          fprintf(stderr, "initMyMain:");
                                          for(n=0,p=$2; p; p=p->next,n++)
                                            {
                                              args[n] = p->str;
                                              fprintf(stderr, " %s", n, $2->str);
                                            }
                                          fprintf(stderr, "\n");
                                          if(!initMyMain(n, args))
                                            {
                                              fprintf(stderr, "Failed to initMyMain\n
");
                                              myfprintf(stdout, NOGOOD);
                                            }
                                          else
                                            myfprintf(stdout, GOOD);
                                          $$ = $1;
                                        }
        | DIRECTEXPONENTS expression.list   {int n;
                                             struct nlist* p;
                                             FLOAT_TYPE e[4];
                                             fprintf(stderr, "setDirectExponents:");
                                             for(n = 0, p=$2; p && n < 4; p = p->next
, n++)
                                               {
                                                 if(p->str)
                                                   {
                                                     if(sscanf(p->str, "%lf", &e[n])
!= 1)
                                                       break;
                                                   }
                                                 fprintf(stderr, " %s(%lf)", p->str,
e[n]);
                                               }
                                             fprintf(stderr, "\n");
                                             if(n < 2)
```

```
                                        {
                                          myfprintf(stdout, NOGOOD);
                                        }
                                      else
                                        {
                                          if(!setDirectExponents(e[0], e[1], e
[2], e[3])!=0)
                                            {
                                              fprintf(stderr, "Failed to setDi
rectExponents\n");
                                              myfprintf(stdout, NOGOOD);
                                            }
                                          else
                                            {
                                              fprintf(stderr, "OK\n");
                                              myfprintf(stdout, GOOD);
                                            }
                                        }
                                          $$ = $1;
                                      }
        | DIRECTSCALE expression.list     {int n;
                                          struct nlist* p;
                                          FLOAT_TYPE    s[3];
                                          fprintf(stderr, "setDirectScale:");
                                          for(n = 0, p=$2; p && n < 3; p = p->
next, n++)
                                            {
                                              if(p->str)
                                                {
                                                  if(sscanf(p->str, "%lf", &s[
n]) != 1)
                                                    break;
                                                }
                                              fprintf(stderr, " %s(%lf)", p->s
tr, s[n]);
                                            }
                                          fprintf(stderr, "\n");
                                          if(n < 3)
                                            {
                                              fprintf(stderr, "syntax error\n"
);
                                              myfprintf(stdout, NOGOOD);
                                            }
                                          else
                                            {
                                              if(!setDirectScale(s[0], s[1], s
[2])!=0)
                                                {
                                                  fprintf(stderr, "Failed to s
etDirectScale\n");
                                                  myfprintf(stdout, NOGOOD);
                                                }
                                              else
                                                {
                                                  fprintf(stderr, "OK\n");
                                                  myfprintf(stdout, GOOD);
                                                }
                                            }
                                          $$ = $1;
                                        }
        | DIRECTPOSITION expression.list {int n;
                                        struct nlist* p;
                                        FLOAT_TYPE    s[3];
                                        fprintf(stderr, "setDirectPosition:");
                                        for(n = 0, p=$2; p && n < 3; p = p->next, n
++)
```



```
                                        {
                                          if(p->str)
                                            {
                                              if(sscanf(p->str, "%lf", &s[n]) !=
1)
                                                break;
                                            }
                                          fprintf(stderr, " %s(%lf)", p->str, s[n
]);
                                        }
                                      fprintf(stderr, "\n");
                                      if(n < 3)
                                        {
                                          fprintf(stderr, "syntax error\n");
                                          myfprintf(stdout, NOGOOD);
                                        }
                                      else
                                        {
                                          if(!setDirectPosition(s[0], s[1], s[2])
!=0)
                                            {
                                              fprintf(stderr, "Failed to setDirec
tPosition\n");
                                              myfprintf(stdout, NOGOOD);
                                            }
                                          else
                                            {
                                              fprintf(stderr, "OK\n");
                                              myfprintf(stdout, GOOD);
                                            }
                                        }
                                       $$ = $1;
                                      }
        | DIRECTROTATE expression.list {int n;
                                       struct nlist* p;
                                       FLOAT_TYPE     s[3];
                                       fprintf(stderr, "setDirectRotate:");
                                       for(n = 0, p=$2; p && n < 3; p = p->next, n+
+)
                                         {
                                           if(p->str)
                                             {
                                               if(sscanf(p->str, "%lf", &s[n]) != 1
)
                                                 break;
                                             }
                                           fprintf(stderr, " %s(%lf)", p->str, s[n]
);
                                         }
                                       fprintf(stderr, "\n");
                                       if(n < 3)
                                         {
                                           fprintf(stderr, "syntax error\n");
                                           myfprintf(stdout, NOGOOD);
                                         }
                                       else
                                         {
                                           if(!setDirectRotate(s[0], s[1], s[2])!=0
)
                                             {
                                               fprintf(stderr, "Failed to setDirect
Rotate\n");
                                               myfprintf(stdout, NOGOOD);
                                             }
                                           else
                                             {
```

```
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                   }
                                  $$ = $1;
                                 }
        | FITELLIPSOID            { if(fitEllipsoidToData() == 0)
                                     {
                                       myfprintf(stdout, NOGOOD);
                                     }
                                   else
                                     {
                                       myfprintf(stdout, GOOD);
                                     }
                                   $$ = $1;
                                 }
        | SETMODE expression.list {int n;
                                   struct nlist* p;
                                   FLOAT_TYPE v[2];
                                   fprintf(stderr, "setMode:");
                                   for(n = 0, p = $2; p && n < 2; p = p->next,
n++)
                                     {
                                       if(p->str)
                                         {
                                           if(sscanf(p->str, "%lf", &v[n]) != 1
)
                                             break;
                                         }
                                       fprintf(stderr, " %s(%lf)", p->str, v[n]
);
                                     }
                                   fprintf(stderr, "\n");
                                   if(n < 2)
                                      {
                                        inquireModes(NULL);
                                        myfprintf(stdout, GOOD);
                                      }
                                   else
                                     {
                                       if(!setMode((int)v[0], v[1]) != 0)
                                         {
                                           fprintf(stderr, "Failed to setMode\
n");
                                           myfprintf(stdout, NOGOOD);
                                         }
                                       else
                                         {
                                           fprintf(stderr, "OK\n");
                                           myfprintf(stdout, GOOD);
                                         }
                                     }
                                  }
        | SETDOMAIN expression.list { if(!$2 || !$2->str)
                                        {
                                          fprintf(stderr, "setdomain 1 : syntax e
rror\n");
                                          myfprintf(stdout, NOGOOD);
                                          /* YYERROR; */
                                        }
                                      else
                                        {
                                          int       domain;
                                          if(sscanf($2->str, "%d", &domain) != 1)
                                            {
                                              fprintf(stderr, "setdomain 2: synta
```



```
x error\n");
                                              myfprintf(stdout, NOGOOD);
                                              /* YYERROR;  */
                                            }
                                          else
                                            {
                                              fprintf(stderr, "setDomain: %02d\n"
, domain);
                                              if(!setDomain(domain))
                                                {
                                                  fprintf(stderr, "Failed to setD
omain\n");
                                                  myfprintf(stdout, NOGOOD);
                                                }
                                              else
                                                {
                                                  fprintf(stderr, "OK\n");
                                                  myfprintf(stdout, GOOD);
                                                }
                                            }
                                        }
                                      $$ = $1;
                                    }
        | HELP                      { fprintf(stderr, "\t<<<<< commands table >>>
>>\n");
                                      fprintf(stderr, "compultephi:\t \n");
                                      fprintf(stderr, "directexponents:\t directe
xponents e11 e12 e21 e22\n");
                                      fprintf(stderr, "directposition:\t directpo
sition x y z\n");
                                      fprintf(stderr, "directscale:\t directscale
 x y z\n");
                                      fprintf(stderr, "drawsq:\t\n");
                                      fprintf(stderr, "echo: \"...\"\n");
                                      fprintf(stderr, "end to finish\n");
                                      fprintf(stderr, "fitellipsoid:\t \n");
                                      fprintf(stderr, "fitmodel:\t\n");
                                      fprintf(stderr, "genpolygons:\t genpolygons
 sampling\n");
                                      fprintf(stderr, "help:\t show this list\n")
;
                                      fprintf(stderr, "init:\t initialize environ
ment.\n");
                                      fprintf(stderr, "initglobal:\t ... has to b
e called after read command\n");
                                      fprintf(stderr, "inquireexponents:\t\n");
                                      fprintf(stderr, "inquirescale:\t\n");
                                      fprintf(stderr, "mappoints:\t\n");
                                      fprintf(stderr, "projectpoints:\t\n");
                                      fprintf(stderr, "promptsq:\t\n");
                                      fprintf(stderr, "read:\t read datafile \n")
;
                                      fprintf(stderr, "report:\t\n");
                                      fprintf(stderr, "setdomain:\t setdomain dom
ainid\n");
                                      fprintf(stderr, "setmode: setmode modeid va
lue\n");
                                      fprintf(stderr, "setnormals:\t correct norm
al vectors\n");
                                      fprintf(stderr, "setradius:\t setradius mod
e(0-top,1-kubire,2-bottom) radius-value\n");
                                      fprintf(stderr, "settopshape:\t \n");
                                      fprintf(stderr, "\n");
                                    }
        | INITGLOBAL                { if(!initGlobalBuffer())
                                        {
```

```
                                        fprintf(stderr, "Failed to initGlobalBu
ffer\n");
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    $$ = $1;}
        | SETTOPSHAPE               { if(!setTopShape())
                                      {
                                        fprintf(stderr, "Failed to setTopShape\
n");
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    $$ = $1;}
        | COMPUTEPHI                { if(!computePhiMatrix())
                                      {
                                        fprintf(stderr, "Failed to computePhiMa
trix\n");
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    $$ = $1;}
        | MAPPOINTS                 { if(!mapClosestPoints())
                                      {
                                        fprintf(stderr, "Failed to mapClosestsP
oints\n");
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    $$ = $1;}
        | PROJECTPOINTS             { if(!mapProjectPoints())
                                      {
                                        fprintf(stderr, "Failed to mapProjectPo
ints\n");
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    $$ = $1;}
        | FITMODEL                  { if(!fitModel())
                                      {
                                        fprintf(stderr, "Failed to fitModel\n")
;
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
```



```
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    $$ = $1;}
        | EXPONENTS                 { if(!setExponents())
                                      {
                                        fprintf(stderr, "Failed to setExponents
\n");
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    $$ = $1;}
        | REPORT                    { report(); $$ = $1;}
        | PROMPTSQ expression.list  { int ret;
                                    if(!$2)
                                      ret = promptSQ(NULL);
                                    else
                                      ret = promptSQ($2->str);
                                    /* if(ret)
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "Failed to drawSQ\n");
                                        myfprintf(stdout, NOGOOD);
                                      }*/
                                     $$ = $1;
                                    }
        | DRAWSQ expression.list    { int ret;
                                    if(!$2)
                                      ret = drawSQ(NULL);
                                    else
                                      ret = drawSQ($2->str);
                                    if(!ret)
                                      {
                                        fprintf(stderr, "Failed to drawSQ\n");
                                        myfprintf(stdout, NOGOOD);
                                      }
                                    else
                                      {
                                        fprintf(stderr, "OK\n");
                                        myfprintf(stdout, GOOD);
                                      }
                                     $$ = $1;
                                    }
        | GENPOLYGONS expression.list {fprintf(stderr, "genpolygons: debug message\
n");
                                     if(!$2 || !$2->str)
                                       {
                                         fprintf(stderr, "No argument has been fo
und.\n");
                                         fprintf(stderr, "You can genpolygon with
out sampling value,\n");
                                         fprintf(stderr, "with argument negative
value(e.g. -1)\n");
                                         fprintf(stderr, "gePolygons 1 : syntax e
rror\n");
                                         myfprintf(stdout, NOGOOD);
                                         /* YYERROR; */
                                       }
```

```
                                        else
                                          {
                                            FLOAT_TYPE sampling;
                                            if(sscanf($2->str, "%lf", &sampling) !=
1)

                                              {
                                                fprintf(stderr, "genPolygons 2: synt
ax error\n");

                                                myfprintf(stdout, NOGOOD);
                                                /* YYERROR; */
                                              }
                                            else
                                              {
                                                fprintf(stderr, "genPolygons: % -12.
5lf\n", sampling);

                                                if(!genPolygons(sampling))
                                                  {
                                                    fprintf(stderr, "Failed to genPo
lygons\n");

                                                    myfprintf(stdout, NOGOOD);
                                                  }
                                                else
                                                  {
                                                    fprintf(stderr, "OK\n");
                                                    myfprintf(stdout, GOOD);
                                                  }
                                              }
                                          }
                                        $$ = $1;
                                      }
        | INQEXPONENTS                {FLOAT_TYPE e11, e12, e21, e22;
                                       if(!inquireExponents(&e11, &e12, &e21, &e22)
)

                                         {
                                           fprintf(stdout, NOGOOD);
                                         }
                                       else
                                         {
                                           myfprintf(stdout, GOOD);
                                           fprintf(stdout, "%lf %lf %lf %lf\n",
                                                   e11, e12, e21, e22);
                                           fflush(stdout);
                                         }
                                       $$ = $1;
                                      }
        | INQSCALE                    {FLOAT_TYPE x, y, z;
                                       if(!inquireScale(&x, &y, &z))
                                         {
                                           myfprintf(stdout, NOGOOD);
                                         }
                                       else
                                         {
                                           myfprintf(stdout, GOOD);
                                           fprintf(stdout, "%lf %lf %lf\n", x, y, z
);

                                           fflush(stdout);
                                         }
                                       $$ = $1;
                                      }
        | SETNORMALS                  { int ret;
                                        ret = setNormals();
                                        if(!ret)
                                          {
                                            fprintf(stderr, "Failed to setNormals\n
");

                                            myfprintf(stdout, NOGOOD);
```



```
                                          }
                                        else
                                          {
                                            fprintf(stderr, "OK\n");
                                            myfprintf(stdout, GOOD);
                                          }
                                        $$ = $1;
                                      }
        | SETRADIUS expression.list   {int n;
                                       struct nlist* p;
                                       FLOAT_TYPE v[2];
                                       fprintf(stderr, "setRadius:");
                                       for(n = 0, p = $2; p && n < 2; p = p->next, n+
+)

                                         {
                                           if(p->str)
                                             {
                                               if(sscanf(p->str, "%lf", &v[n]) != 1)
                                                 break;
                                             }
                                           fprintf(stderr, " %s(%lf)", p->str, v[n]);
                                         }
                                       fprintf(stderr, "\n");
                                       if(n < 2)
                                         {
                                           fprintf(stderr, "setRadius\n");
                                           myfprintf(stdout, NOGOOD);
                                         }
                                       else
                                         {
                                           if(!setRadius((int)v[0], v[1]) != 0)
                                             {
                                               fprintf(stderr, "Failed to setRadius\n
");

                                               myfprintf(stdout, NOGOOD);
                                             }
                                           else
                                             {
                                               fprintf(stderr, "OK\n");
                                               myfprintf(stdout, GOOD);
                                             }
                                         }
                                      }
        ;
expression.list
        :                             {$$ = NULL;}
        | expression.list expr        {/* fprintf(stderr, "$1 %08X, $2 %08X(%s)\n",

                                              $1, $2, $2); */
                                       $$ = makelist($1, $2);}
        ;
expr
        : STRING                      { /* fprintf(stderr, "STRING: %s\n", $1); */
                                        $$ = $1;
                                      }
        | INTEGER                     {$$ = $1;}
        | CONSTANT                    {$$ = $1;}
        ;
%%
int             nodecnt = 0;
int             nodesize = 0;
struct nlist*   nodearea = NULL;
/*
 * get the memory for the node
 */
struct nlist*
```

```
getnode(void)
{
  fprintf(stderr, "getnode: nodecnt  %02d, nodesize %02d, nodearea %p\n",
          nodecnt, nodesize, nodearea);
  if(nodesize <= nodecnt)
    {
      if((nodearea = (struct nlist*)recalloc(nodearea, (nodesize + NODESIZE), sizeof(
struct nlist)))
         == NULL)
        {
          fatal("nodearea overflow");
        }
      else
        {
          nodesize += NODESIZE;
        }
    }
  return &nodearea[nodecnt++];
}
/*
 * make the list
 */
struct nlist*
makelist(struct nlist* top, char* name)
{
  struct nlist*         q;

  if((q = getnode()) == NULL)
    {
      perror("getnode");
      fprintf(stderr, "makelist: Failed to getnode.\n");
      return NULL;
    }
  if(top == NULL)
    {
      q->last = q;
      q->next = NULL;
      q->str = strdup(name);
      return q;
    }
  else
    {
      q->last = NULL;
      q->next = NULL;
      q->str = strdup(name);
      top->last->next = q;
      top->last = q;
      return top;
    }
}

/*
 * display an error message
 */
yyerror(char* s)
{
  extern int yylineno;
  fprintf(stderr, "%s:%d\n", s, yylineno);
}
```