平成 9 年度

筑波大学第三学群情報学類

卒業研究論文

題目：３次元物体の直接操作に関する研究

主専攻	情報科学
著者名	大芝 崇
指導教員	電子・情報工学系 田中 二郎

要旨

３次元仮想空間におけるオブジェクトに対して何らかの操作を行う場合、アイコンやメニュー等を用いた操作は、ユーザにとってその操作性が悪いという問題がある。本研究では、３次元仮想空間上のオブジェクトの例として、ルービックキューブを取り上げた。既存のルービックキューブ操作システムの操作は直感的とは言えず、分かりにくいものであった。

そこで本研究では、オブジェクトを３次元仮想空間上に表示し、直接操作により扱う手法を提案する。アイコンやメニュー等は一切用いず、マウスのみの操作で結果が即座に視覚化されるシステムを OpenGL を用いて実際に作成した。

また、ルービックキューブの特徴である関節構造を扱う手法を、より一般的なオブジェクトに応用するために、関節構造エディタの作成を進めた。

目 次

第 1 章

序論

1.1 はじめに

高機能なグラフィックスライブラリ等を用いたソフトウェアを用いて構築されたシステムの中には、システムの操作法を理解することが難解であるものが多い [16, 17]。例えば、3 次元仮想空間におけるオブジェクトに対して何らかの操作を行う場合を考えてみると、アイコンやメニュー等を用いた間接的な操作体系では、システムの操作法も難解である。

1.2 3 次元統合モデラ “Claymore”

現在、我々の研究室では、3 次元統合モデラ “Claymore” を作成している [2](図 1.1)。ここで3 次元モデラとは、3 次元物体を扱うエディタのことを指す。

様々な 3 次元画像の構築において、3 次元形状を製作するモデリング作業は必須となる基本作業であるが、従来の 3 次元モデラを使いこなすには熟練を要するという問題があったが、“Claymore” は現実世界の粘土細工を真似た部品の変形と、模型工作を真似た部品の組み合わせによって、より直感的なモデリングを行なうことによりこの問題を解決することを目指している。

“Claymore” は、従来の 3 次元モデラと違い三面図を用いず、物体を直接かつ直感的に操作・編集ができ、いわば MacDraw を操作するような感覚で簡単に構築できるモデラである。

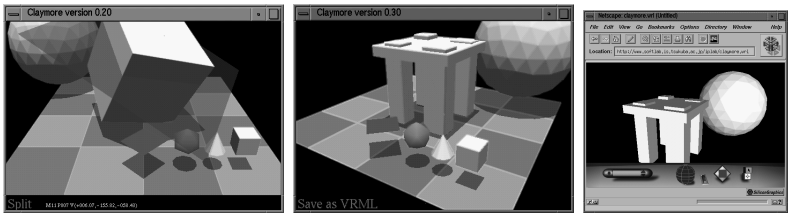

図 1.1: Claymore 実行画面

1.3　3次元オブジェクトの直接操作

本研究では、3次元仮想空間上のオブジェクトの題材として、ルービックキューブを取り上げた[5]。その理由は、既存のルービックキューブ操作システムのほとんど[8, 9, 10, 11, 12, 13, 14, 15]は、操作性が直感的とは言えず、分かりにくいものであったからである。ルービックキューブのような多関節を持ったオブジェクトへの操作には、「回転」の種類がたくさんあるために操作法が複雑になり直感的でないと言う問題がある。

そこで本研究では、3次元仮想空間上のオブジェクトの操作に直接操作手法を応用し、より直感的に扱う手法を提案する。ルービックキューブへの操作には、3次元仮想空間におけるオブジェクトへの重要な操作の1つである「回転」と言う操作が色々なバリエーションを持って含まれているため、本研究で作成したシステムの操作性を考察することは、ルービックキューブに限らず3次元仮想空間上の一般的なオブジェクトへの直接操作に対する操作性を考察するための有用な手段となることが考えられる。

1.4　研究の背景 — “Claymore” と “Cubism”

既に “Claymore” はモデラとしての基本部分であるマウスによる3次元物体の作成・操作・編集・保存等の直接操作インタフェースが完成しており、現在、多関節構造を持った物体も扱えるような機能の実装を試みている。

前述の通りルービックキューブは非常に複雑な多関節物体であり、関節構造を持った物体を扱う手法について研究することは “Claymore” の多関節インタフェースの構築の際にも、本研究の手法の応用が期待出来る。

そこで本研究では “Claymore” の多関節インタフェース構築についての基礎研究として、実際にその手法を実装した直感的なシステム “Cubism”[1] を作成した。

第 2 章

既存のルービックキューブ操作システムの問題点

既に、“Virtual Rubik's Cube” などと銘打ったルービックキューブ操作システムは WWW 上やシェアウェアとして数多く世の中に出廻っており [8, 9, 10, 11, 12, 13, 14, 15]、オブジェクトを 2 次元的に表示したもの [9, 11, 14] とオブジェクトを 3 次元的に表示したもの [8, 10, 12, 13, 15] がある。

- オブジェクトを 2 次元的に表示したものは、「単に菱形が並んでいる」といった感があり、自分が今どこのキューブを触っているのかを把握することは難しい。また、2 次元であるために、今見えている面の裏側の状況の把握が非常に難しい (図 2.1,[9])。
- 回転に伴う各キューブの動きがアニメーションによって描画されない（再描画が瞬時に行われる）ものもあり [9, 12, 14]、これはユーザへのフィードバックに問題がある (図 2.2,[12])。
- オブジェクトを 3 次元的に表示したものにも色々と問題点が挙げられる。まず、射影変換に正射影を用いているものがある [8, 12] が、この場合時々キューブの表裏が逆転してしまったような錯覚に陥ることがある (図 2.3,[8])。
- また、これはオブジェクトを 2 次元的に表示したものにも言えることであるが、回転などのオブジェクトへの操作をアイコンやメニュー、キーボードなどを用いて行うものが多く [8, 9, 10, 11, 12, 13, 14, 15] (図 2.4,[15])、こういった間接的な操作体系は、物理的な操作というよりもむしろ構文的な操作であり [7]、我々が現実世界で実際にルービックキューブを手に持っているような感覚（リアリティ）が希薄になってしまうためユーザにとってその理解が難解になってしまっている。
- オブジェクトを 3 次元的に表示したものには、キューブの回転などの操作をマウスを用いて直接行えるもの [8, 10, 13, 15] も多いが、ここにもユーザを困惑させる原因が潜んでいる。まず、ルービックキューブ全体を回転させる

操作（今見えていない面を見るために行う。ここでは単に『全体の回転』と呼ぶことにする）と、9個のキューブだけをルービックキューブ全体に対して相対的に回転させる操作（各面の色を合わせるために行う。ここでは単に『相対的な回転』と呼ぶことにする）の区別が分かりにくい。多くのシステムはアイコンを用いて対処しているが、中にはセンターキューブ（各面の真ん中にある1色しか持たない面）をつまんで動かした時は「全体の回転」で、その他のキューブをつまんで動かした時には「相対的な回転」が行われるというもの (図 2.5,[13]) もあり、このような仕組みは現実世界の適切な比喩になっておらず、ユーザを混乱させてしまう。

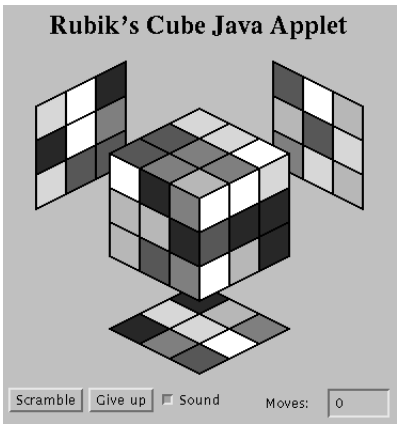

図 2.1: 既存のルービックキューブ操作システム実行画面 [9]。オブジェクトが2次元的に表示されている。

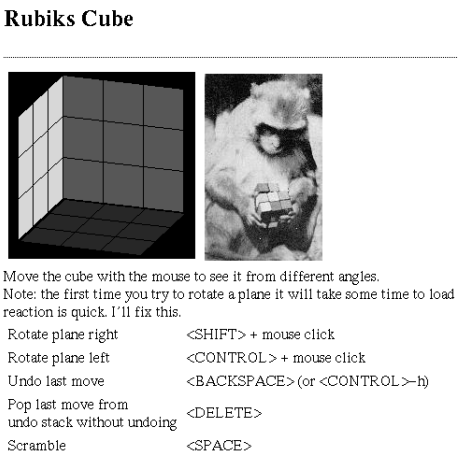

図 2.2: 既存のルービックキューブ操作システム実行画面 [12]。アニメーションによって描画されない。

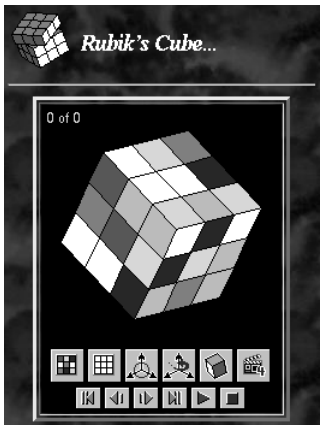

図 2.3: 既存のルービックキューブ操作システム実行画面 [8]。射影変換に正射影を用いている。

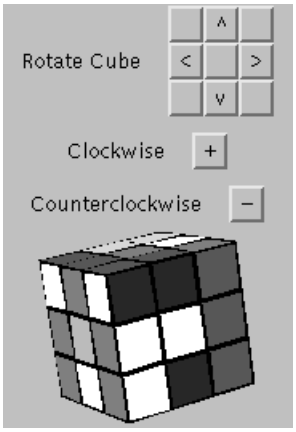

図 2.4: 既存のルービックキューブ操作システム実行画面 [15]。アイコンやメニュー、キーボードなどを用いている。

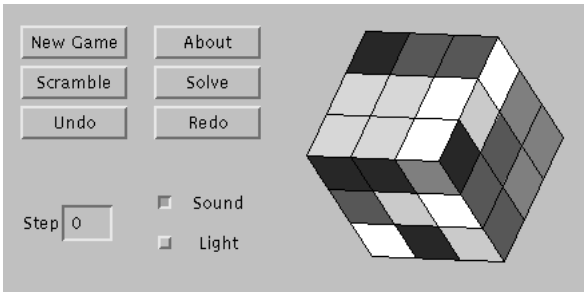

図 2.5: 既存のルービックキューブ操作システム実行画面 [13]。「全体の回転」と「相対的な回転」の区別が分かりにくい。

第 3 章

“Cubism” 作成の方針

3.1 既存のシステムの問題点の改善

上で挙げたような問題点を考慮に入れ、本研究では以下のような方針でシステムの実装を行うことにした。

- オブジェクトは 3 次元的に表示する。
- 射影変換には透視射影を用いる。
- オブジェクトへの操作には、アイコンなどの間接的な操作体系は用いず、マウスのみを用いることにより、オブジェクトを直接操作する。
- 回転の操作にアニメーションを用いることにより、フィードバックを向上させる。
- 「全体の回転」の操作と、「相対的な回転」の操作とは、それぞれを異なるマウスボタンに割り当てることにより区別する。

3.2 追加機能 — 操作性の向上

以上は既存のシステムの問題点を改善させるためのものであるが、以下のような追加機能も実装し、さらなる操作性の向上 [6] をはかった。

- 現在触っている面の黒枠の色を少し白くすることにより、フィードバックを向上させる。これにより、ユーザは今どのキューブの面を触っているのかを知ることが出来る。
- 「相対的な回転」の操作に関して、1/4 回転 ($\pm 90°$) だけでなく、2/4 回転 (半回転。$\pm 180°$) や 3/4 回転 ($\pm 270°$)、4/4 回転 (1 回転。$\pm 360°$) や、それ以上の回転を一度の操作で出来るようにした（ほぼ総ての既存のシステムでは 1/4 回転以上の回転を一度の操作で実行することは不可能である）。

- もし「相対的な回転」の途中でマウスボタンをリリース (図 3.1左) しても、リリースされた時点での適切な位置に 9 個のキューブを揃えてくれるようにする (図 3.1右)。

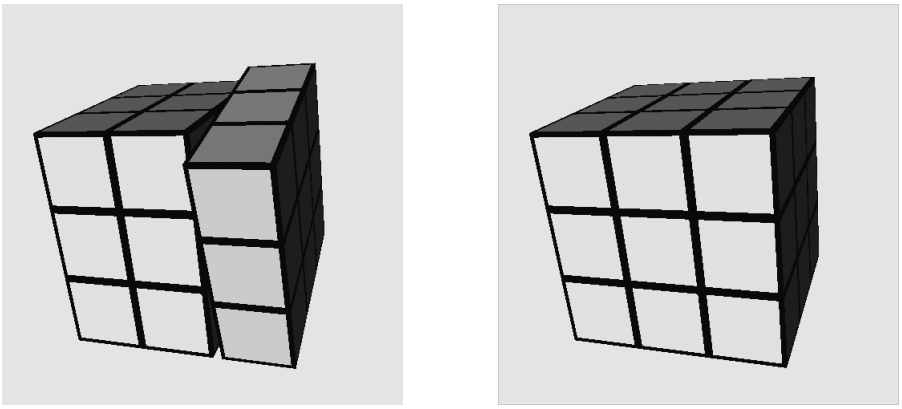

図 3.1: システム側で適切な位置に 9 個のキューブを揃える。

第 4 章

直接操作インタフェース

4.1 直接操作とは

ここで、「直接操作」というものについて、簡単に説明する。
Ben Shneiderman[7] は「直接操作」について、

- **操作の世界の視覚的な表現** — 操作の結果が即座に視覚化され、その操作の結果と表示に一致性がある。
- **構文的な操作ではなく、物理的な操作体系** — 複雑で目に見えないような操作ではなく、簡単でわかりやすい。

と定義している。 Jurgen Ziegler[4] は、

- **オブジェクトとその状態を、ユーザとシステムが共有している。**
- **操作に対して、結果が即座にフィードバックされる。**

このような操作を「直接操作」と呼んでいる。
また、 Michael Chen ら [3] は、 WYSIWYG, What You See is What You Get を意識し、直接操作について

- **WYDIWYS, What You Do is What You See**

であると提唱している。

すなわちこれらをまとめると「直接操作」とは、操作の結果が視覚的にフィードバックされ、システムの状態や操作する対象の概念を直感的に把握させることによって、ユーザの認知的な負荷を減らせるような分かり易い操作のことである。

4.2 作成したルービックキューブ操作システム “Cubism”

本研究で作成したルービックキューブ操作システム “Cubism” について述べる。“Cubism” は汎用 3 次元グラフィックスライブラリ OpenGL を用いて作成されており、現在 Unix (IRIX 6.3, Solaris 2.5.1), Windows95 上において動作する。

実装されている機能は、ルービックキューブの「全体の回転」と、触った面の黒枠の色を変化させることと、そして 9 個のキューブの「相対的な回転」である (図 4.1,4.2,4.3)。ルービックキューブは常に 9 個のキューブが一度に回転する)。回転に関する操作は、すべてマウスにより行う直接操作である。

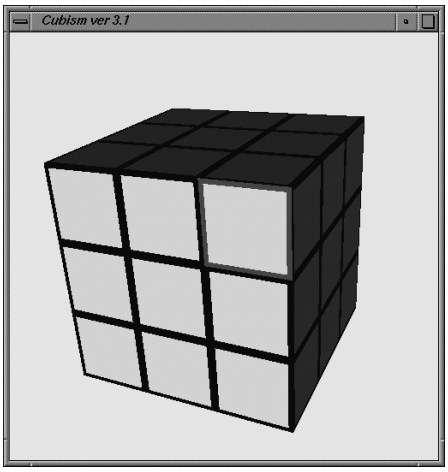

図 4.1: 初期画面。触った面の黒枠の色が変化。

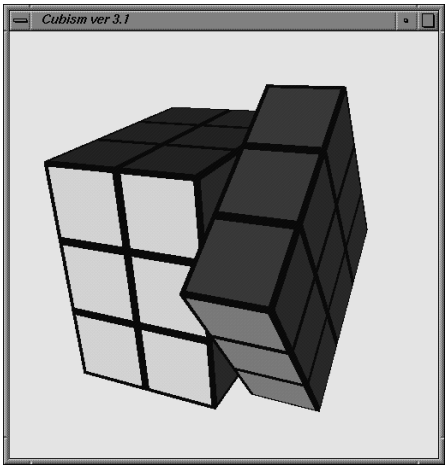

図 4.2: 9 個のキューブの相対的な回転をしているところ。

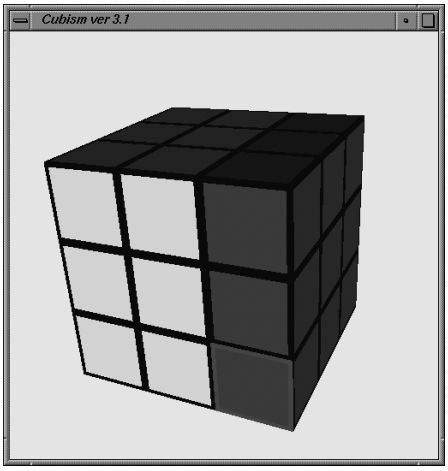

図 4.3: 1/4 回転の結果

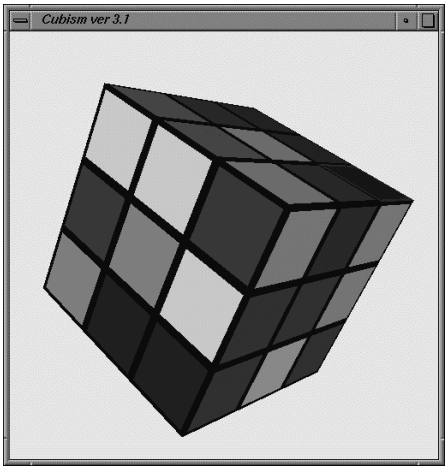

図 4.4: 何度も回転を繰り返した結果

4.3 操作法

本システムの操作はアイコンやメニュー等は一切用いず、すべてマウスのみの操作で行い、結果は即座にアニメーションにより視覚化される。

「全体の回転」の操作と、「相対的な回転」の操作とは、それぞれを異なるマウスボタンに割り当てることにより区別する。実際には、

- 「全体の回転」 · · · マウスのミドルボタン
- 「相対的な回転」 · · · マウスのレフトボタン

という割り当て方をした (図 4.5)。

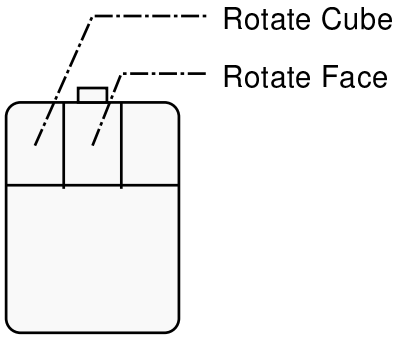

図 4.5: マウスによる操作

4.3.1 「全体の回転」について

適当なキューブ（どれでも良い）の上にマウスカーソルを合わせ、マウスのミドルボタン (図 4.5) のクリック & ドラッグによってルービックキューブの「全体の回転」を行う。その際、ユーザがマウスを動かした方向とルービックキューブが回転する方向とが一致する (図 4.6) ので、ユーザは 3 次元仮想空間に内部的に設定されている座標軸等は意識しなくてもよい。

図 4.6 は、「全体の回転」の様子を擬似的に表すものである。中央にあるウィンドウを持った少し大きな図が初期画面で、その周りにある 8 つの図は、それぞれ初期画面からマウスを各図がある方向へドラッグした時の、ルービックキューブの姿勢を示す図である。

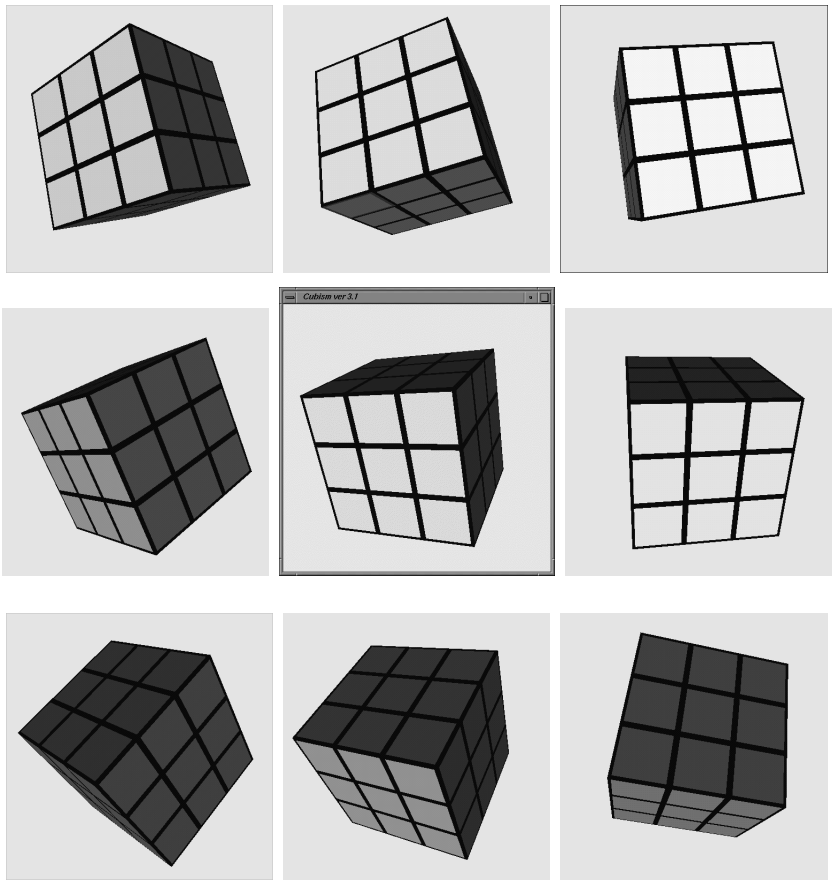

図 4.6: マウスの移動方向とルービックキューブの回転方向が一致

4.3.2 「相対的な回転」について

回転させるキューブの選択（クリック） マウスカーソルを回転させるキューブの 1 面の上に合わせると、選択された面の外側の黒枠の部分の色が少し白っぽくなり、ユーザにキューブの選択が行われたことがフィードバックされる（図 4.1）。実際に回転を始める際には、マウスのレフトボタン (図 4.5) をクリックする。

回転させる方向へドラッグ マウスをドラッグし、マウスカーソルを回転方向に動かすことにより、選択された 9 個のキューブの「相対的な回転」を行う (図 4.2, 4.3)。マウスをクリックした位置からの距離（ピクセル数）が回転角度に変換されるので、、1/4 回転以上の回転（例えば半回転）も可能である。つまり、ドラッグする距離を長くすることによって、1/4 回転を何度も繰り返さなくても 1 回でユーザの要求する操作が出来る。もしユーザが回転の途中でレフトボタンをリリースしても、システム側が、ユーザのマウスリリースの時点での適切な位置に 9 個のキューブを揃えてくれる (図 3.1)。

4.4 原理, 実装方法

4.4.1 「全体の回転」

オブジェクトをウィンドウに表示する際には、モデリング座標系 (Lx, Ly, Lz) でのオブジェクトの座標値をワールド座標系 (Wx, Wy, Wz) に変換し、さらにその座標を視点座標系 (Vx, Vy, Vz) に変換して透視変換を行うことによって、2次元の画面に投影している。

ルービックキューブの重心は常にモデリング座標系 (Lx, Ly, Lz) の原点に位置し、図 4.7のように原点から x, y, z 軸が出ている。

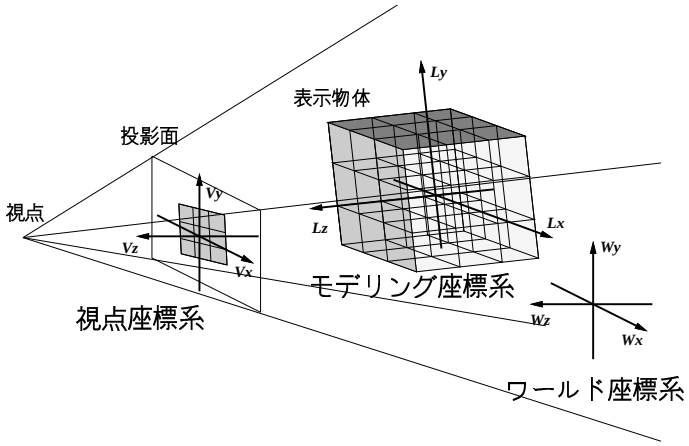

図 4.7: 各座標系とオブジェクト

本システムでは、モデリング座標系において回転変換を施すのではなく、オブジェクトをワールド座標系にモデリング変換した後に回転変換を施している (式 4.1)。

$$\boldsymbol{b}' = \boldsymbol{R_y}(m_x) \cdot \boldsymbol{R_x}(m_y) \cdot \boldsymbol{b}, \tag{4.1}$$

$$\boldsymbol{R_x}(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta\pi/180) & -\sin(\theta\pi/180) \\ 0 & \sin(\theta\pi/180) & \cos(\theta\pi/180) \end{pmatrix},$$

$$\boldsymbol{R_y}(\theta) = \begin{pmatrix} \cos(\theta\pi/180) & 0 & \sin(\theta\pi/180) \\ 0 & 1 & 0 \\ -\sin(\theta\pi/180) & 0 & \cos(\theta\pi/180) \end{pmatrix}.$$

（ここで、m_x, m_y: マウスカーソルが移動したピクセル数, $\boldsymbol{b}'$: 回転後のモデリング座標系の座標軸, $\boldsymbol{b}$: 回転の直前のモデリング座標系の座標軸）

回転変換において、マウスの移動方向に垂直になるように回転軸が設けられ、回転角度はマウスの座標の変位 (m_x, m_y) を利用しているので、ユーザはマウスの移動方向と移動距離によって、オブジェクトの回転をコントロールすることが出来る。

オブジェクトの回転に関して

Michael Chen らは、3 次元物体の直接操作手法（特に回転操作に関して）の 1 つとして、「Virtual Sphere Controller」を提案している [3]。

Virtual Sphere は、仮想的な透明の球体を、回転操作を施したいオブジェクトを覆うように表示し、この球体を操作することによって内包されているオブジェクトの回転を実現している (図 4.8)。

しかし、この手法は回転操作を施したいオブジェクトを直接的に操作しているのではなく、仮想的な球体を通して間接的に操作している点で本研究の操作手法とは本質的に異なる。

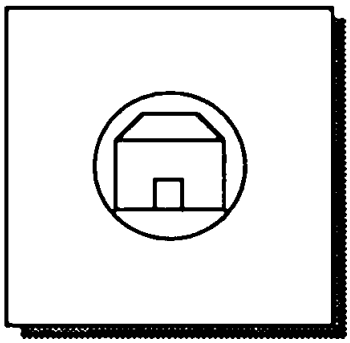

図 4.8: Virtual Sphere Controller

4.4.2 「相対的な回転」

本来、ルービックキューブの 9 個のキューブの「相対的な回転」は、モデリング座標系における 3 本の軸に関する回転がそれぞれ可能である。つまり回転の種類が 3 種類ある (図 4.9)。

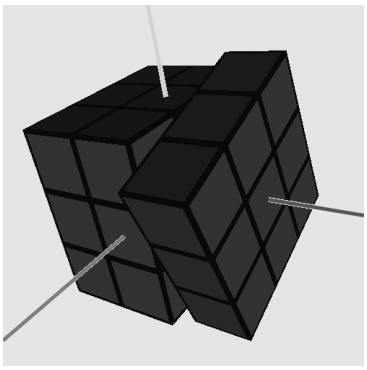

図 4.9: モデリング座標系の 3 軸

回転軸の絞り込み

本研究では、回転させるキューブの 1 面の選択をマウスのレフトボタンのクリックによって決定した時点で、3 本ある回転軸を 2 本に絞り込む手法を用いている。回転軸を 2 本に絞り込むことによって、本来 2 次元の入力デバイスであるマウスによる、ルービックキューブという関節を持つ 3 次元のオブジェクトの操作が可能になっている。

実際には、以下のような手法を用いている。

まず、マウスのレフトボタンのクリックによって選択されたキューブの 1 面 (これは 1 枚の平面である) の、法線を調べる。そして、3 本ある回転軸の中から、今得られた法線と平行であるものを除く (図 4.10)。このようにして本来は 3 本ある回転軸を 2 本に絞り込むことが出来る。

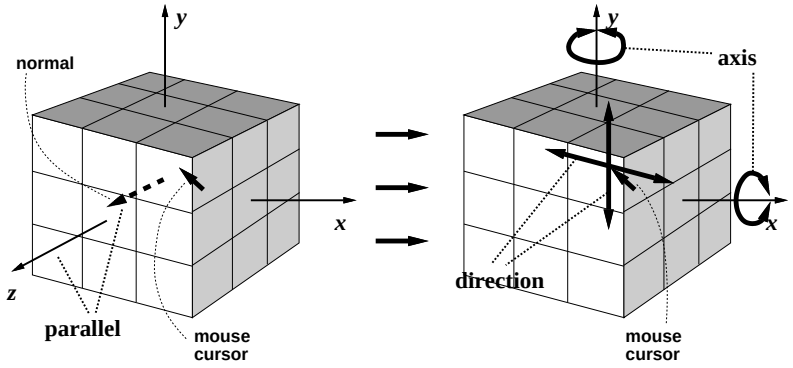

図 4.10: 回転軸の絞り込み。

このような実装方法にした理由は、クリックによって得られた法線と平行である回転軸に沿って回転を行う操作は、現実世界において指先でルービックキューブを扱う操作に当てはめてみると不自然な (実際の手ではやりにくい) 操作だったからである。

回転軸、回転方向の決定

次に、先程 2 本に絞り込まれた回転軸を、ドラッグした方向を用いて 1 本に絞り込む手法を説明する。

まず図 4.11左のように、3 次元仮想空間上に 2 本の直線を設ける。この 2 直線は、クリックされた平面上の点を交点とし、先程 2 本に絞り込まれた回転軸にそれぞれ平行な直線である。

次に、この 2 直線を透視変換し、2 次元のスクリーン (= ウィンドウ画面) に投影する。これによって、2 次元の 2 本の直線を得る (図 4.11右)。この 2 直線の方向ベクトルをそれぞれ $\boldsymbol{n1} = (n1_x, n1_y), \boldsymbol{n2} = (n2_x, n2_y)$ とする。

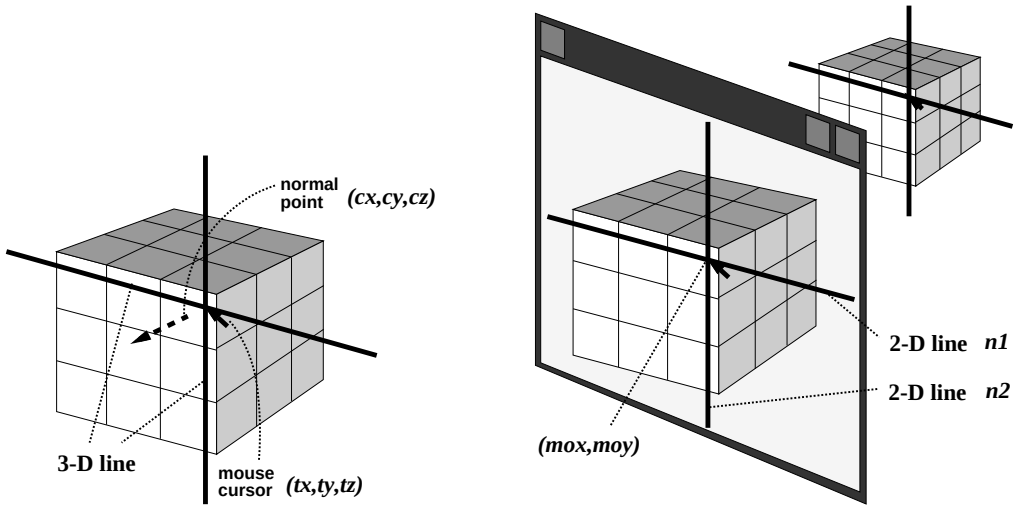

図 4.11: 選択された面に、2 本の直線を設ける。

マウスをクリックした位置を (m_{ox}, m_{oy}) とすると、この 2 直線の方程式は点 (m_{ox}, m_{oy}) を共通して通り、それぞれもう 1 点 $(m_{ox} + n1_x, m_{oy} + n1_y)$ と $(m_{ox} + n2_x, m_{oy} + n2_y)$ を通る。

マウスをドラッグした後の、現在のマウスカーソルの位置を (m_{nx}, m_{ny}) とすると、点 (m_{nx}, m_{ny}) と 2 本の直線とのそれぞれの距離 h_1, h_2 は、

$$h_1 = \frac{|n1_y \cdot (m_{nx} - m_{ox}) - n1_x \cdot (m_{ny} - m_{oy})|}{\sqrt{{n1_y}^2 + {n1_y}^2}},$$
$$h_2 = \frac{|n2_y \cdot (m_{nx} - m_{ox}) - n2_x \cdot (m_{ny} - m_{oy})|}{\sqrt{{n2_y}^2 + {n2_y}^2}}.$$

によって計算される (図 4.12左)。実際には次の関数によって計算している。

```
void CalcRotateAxis(DisplayPoint *co, DisplayPoint *a1, DisplayPoint *a2,
                    GLfloat *cursor_x, GLfloat *cursor_y)
{
  GLfloat h1, h2;

  h1 = fabs(a1->y * (*cursor_x - co->x) - a1->x * (*cursor_y - co->y))
    / sqrt(a1->y * a1->y + a1->x * a1->x);
  h2 = fabs(a2->y * (*cursor_x - co->x) - a2->x * (*cursor_y - co->y))
    / sqrt(a2->y * a2->y + a2->x * a2->x);
}
```

「相対的な回転」の回転方向は、その中から距離 h_1, h_2 の短い方の直線に平行な方向を選ぶ。その一方で「相対的な回転」の回転軸は、先程 2 本に絞り込まれた回

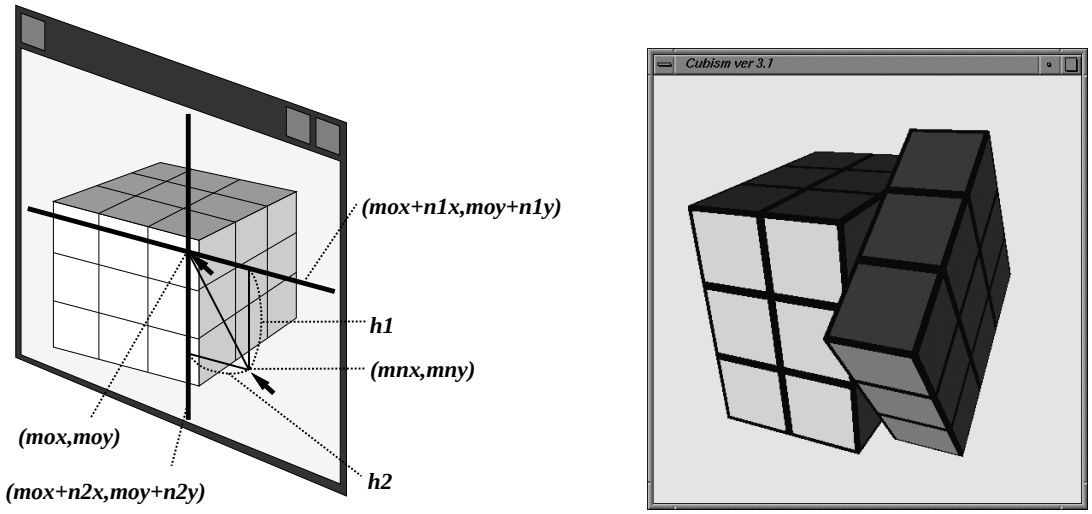

図 4.12: 回転軸、回転方向の決定。

転軸の中から、距離 h_1, h_2 の長い方の直線をワールド座標系に逆変換して得られる 3 次元の直線に平行な 1 本の回転軸に絞り込まれる。

このように、回転軸、回転方向が同時に決定される。図 4.12右では、$h_1 > h_2$ なので、回転方向は h_2 の直線に平行な方向、つまり $\boldsymbol{n2}$ の方向にオブジェクトが回転する。この時、回転軸は h_1 の直線に平行な 3 次元の 1 本の直線に決定する。

回転角度の計算

回転角度は、マウスの移動距離を利用している。回転角度 θ は、

$$\begin{aligned} \theta &= \sqrt{(m_{nx} - m_{ox})^2 + (m_{ny} - m_{oy})^2} \\ &= \sqrt{{m_x}^2 + {m_y}^2}. \end{aligned}$$

によって求めている。

マウスをクリックした位置からの距離（ピクセル数）が回転角度になるので、ユーザはマウスの移動距離によって回転角度をコントロール出来る。マウスの移動距離を多くすればするほど回転角度も増加し、1/4 回転以上の回転（例えば半回転）も可能である。

第 5 章

3次元関節構造作成エディタ “Juggler”

我々の研究室では、3次元統合モデラ “Claymore” の開発を進めている。“Claymore” に関節構造を持った物体も扱える機能を実装するにあたって、基礎研究として現在 “Cubism” とは別に、3次元関節構造作成エディタ “Juggler” を作成中である (図 5.1)。

“Juggler” は、「関節」自体を 1 つのパーツとして扱い、“Claymore” で扱うようなより一般的な物体を複数組み合わせることによって、物体同士に関節構造を設定し、直接操作によって出来るだけ簡単にルービックキューブのような複雑な関節構造を持った物体も作成することを目指している。

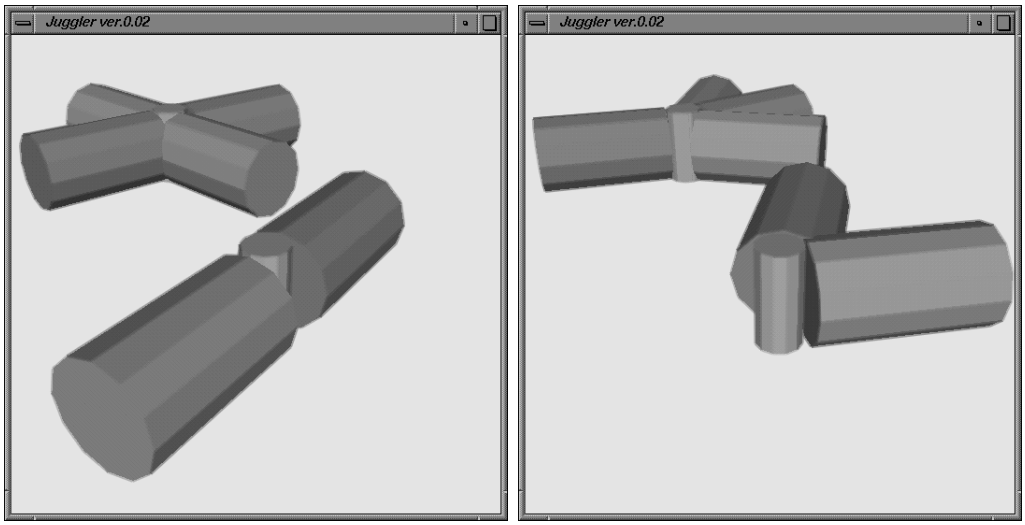

図 5.1: “Juggler” 実行画面。

“Juggler” は、「関節」自体や、関節によって結合される3次元オブジェクトのモデルデータをファイルから複数個読み込んで、それぞれを 1 つのパーツとして扱うことが出来る。

5.1　オブジェクトの移動

オブジェクトの移動には、オブジェクトをマウス操作によってマウスの移動量に合わせて動く直感的な手法を用いた。“Juggler” では現在オブジェクトを 3 次元仮想空間内の地面と水平に平行移動させることが出来る。オブジェクトの移動にはマウスの右ボタンを用いる。関節によって結合された片方のオブジェクトをマウスによって選択し移動すると、もう片方のオブジェクトもその操作に連動して移動させることが出来る。

3 次元仮想空間内での位置の指定には、x, y, z の 3 種類のパラメータの設定を必要とするが、入力される情報はマウスの移動による 2 次元の座標情報であるという問題がある。また、マウスの移動量をそのまま物体の x, y, z 座標の情報に単純に対応させるだけでは、ほとんどの場合、画面上のオブジェクトとマウスカーソルの動きが一致しないという問題がある。

マウスカーソルの位置と、選択されたオブジェクトの位置を常に一致させるために、以下のような手法を用いた。

まず、あらかじめ移動を開始する前に、オブジェクトのマウス接触位置 (p_x, p_y, p_z) を求めておく。次に、マウスの移動に合わせて地面とマウスカーソルによる指定点の交点 (p'_x, p'_y, p'_z) を、移動後のオブジェクトの、マウス接触位置とすることで、常に画面上のオブジェクトとマウスカーソルの動きを一致させることが出来る。

5.2　関節で連結されたオブジェクトの回転

関節で連結された個々のオブジェクトを選択し、関節の周りを回転させることが出来る。関節で連結されたオブジェクトの回転は “Cubism” においての「相対的な回転」と同じ原理の操作であるため、マウスの左ボタンを用いて操作を行う。

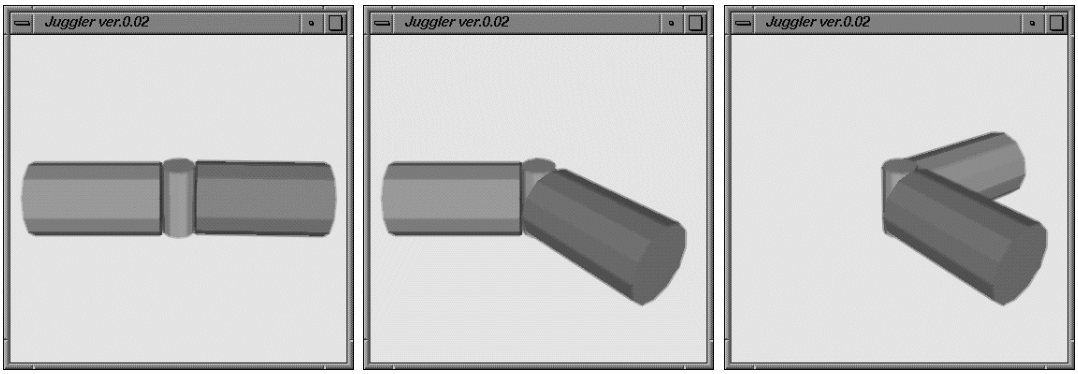

図 5.2: 関節で連結されたオブジェクトの回転。

現在のところ “Juggler” では実世界の蝶番 (ちょうつがい) をモチーフにした関節パーツをサポートしている。この関節パーツは 2 つのオブジェクトの間に挟まった形で 2 つのオブジェクトは関節パーツを回転軸として回転する (図 5.1)。

実際には以下の手順で関節の回転を実現している。

まず、選択されたオブジェクトのマウス接触位置 (p_x, p_y, p_z) を求めておく。マウスの移動後、地面とマウスカーソルによる指定点の交点を (p'_x, p'_y, p'_z) とする。関節パーツの軸と地面との交点を (a_x, a_y, a_z) とすると、3 点 $(p_x, p_y, p_z), (a_x, a_y, a_z), (p'_x, p'_y, p'_z)$ によって作られる角度 φ を回転角度として、選択されたオブジェクトを関節パーツを回転軸として回転させることが出来る。

第 6 章

まとめ

本研究では、既存のルービックキューブ操作システムの問題点を指摘し、その解決策としてまず「直接操作」について考察し、3 次元仮想空間上のオブジェクトをマウスのみを用いた直接操作により扱う手法を提案し、実際にその手法を実装したシステム “Cubism” を作成した。

また、ルービックキューブの特徴である関節構造を扱う手法を、より一般的なオブジェクトに応用するために、関節構造エディタ “Juggler” の作成を進めた。

謝辞

本研究を進めるにあたり、終始親切に指導してくださった、担当教官の田中二郎助教授に深く感謝する。

また、インタラクティブプログラミング研究室のメンバーおよび友人諸氏、特に光延 秀樹氏には、研究に関する様々なアイディア・アドバイスを提供していただいた。ここに改めて感謝する。

本研究に対して様々なコメントを下さった多くの方々に、感謝の意を述べる。

参考文献

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

[2] 光延 秀樹, 岡田 潤, 田中 二郎: 三次元インタラクティブ編集環境の構築, 並列・分散処理研究推進機構 成果概要 (1997), pp.130-131.

[3] Michael Chen, S.Joy Mountford, Abigail Sellen: A Study in Interactive 3-D Rotation Using 2-D Control Devices, Proceedings of SIGGRAPH'88 (Atlanta, August 1-5, 1988), In *Computer Graphics* 22, 4 (August 1988), pp121-129.

[4] Jurgen Ziegler: Interactive Techniques, *ACM Computing Surveys*, Vol.28, No.1, March 1996, pp.185-187.

[5] P. ジョンソン: ケーススタディ 1. ルービックキューブ, 対話システム設計に適用されるタスク分析, ヒューマンインタフェースの設計方法, pp.201-216.

[6] D.A. ノーマン: 可視性とフィードバック, 誰のためのデザイン？, 新曜社, pp.158-168.

[7] Ben Shneiderman: 直接操作の原理, ユーザインタフェースの設計方法, 日経 BP 社, pp.126-161.

[8] Glen Nakamura: Glen Nakamura has several cube-like java puzzles, http://www2.hawaii.edu/~gnakamur/puzzle/frame.html

[9] Michael Schubart: Michael Schubart's java cube, http://www.best.com/~schubart/rc/

[10] Junhao Xiong: Junhao Xiong's java cube, http://ucunix.san.uc.edu/~xiongjo/mytest2.html

[11] Karl Hornell: Karl Hornell's java cube, http://www.tdb.uu.se/~karl/java/rubik.html

[12] Geert-Jan van Opdorp: Geert-Jan van Opdorp's java cube, http://www.aie.nl/~geert/java/public/Rubik.html

[13] Song Li: Song Li's java cube,
http://www.cs.umbc.edu/~sli2/cube/cube.html

[14] Justin Campbell: Justin Campbell's java cube,
http://www.reed.edu/~jmc/project

[15] Kevin Leeds: Kevin Leeds' interactive java cube,
http://www.math.gatech.edu/~leeds/java/MoCube/MoCube.html

[16] Debbie Myers: IRIS Showcase User's Guide 日本語版, 日本シリコングラフィックス株式会社.

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

付録 A

“Cubism” のソースコード

A.1 Cubism.c

```
/*
   Cubism.c
   Program by Takashi Oshiba <ohshiba@softlab.is.tsukuba.ac.jp>

   for IRIX 6.3
*/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <math.h>
#include "Cubism.h"
#include "material.h"
#include "draw.h"
#include "rubik.h"
#include "select.h"
#include "modeldata.h"
#include "print.h"

#define M11 0
#define M12 4
#define M13 8
#define M14 12
#define M21 1
#define M22 5
#define M23 9
#define M24 13
#define M31 2
#define M32 6
#define M33 10
```

```
#define M34 14

#define K 0.363970234 // tan(20)
#define EyeDist 8.0

GLfloat whole_spin_x = 0.0;
GLfloat whole_spin_y = 0.0;
GLfloat  CubeViewMatrix[16]; //全体のキューブの姿勢
GLfloat spin_y = 0.0;
GLfloat scale = 1.0;
GLint Spin = 0;
GLint SpinAxis= 0;
GLint SpinRow= 0;
GLint Drag = 0;
GLint Scale = 0;
GLint xsize = 500;
GLint ysize = 500;
GLint MouseON = 0;
GLint last_x = 0, last_y = 0;
GLint mylast_x, mylast_y;
GLint RotateLock = 0;

extern MATERIAL_LIST Material, Material0, Material1, Material2, Material3,
  Material4, Material5, MaterialB,
  TransparentWhite;
extern LIGHT_MODEL Lmodel;
extern LIGHT Light0;

CubeInfo CubeData[3][3][3];
SurfInfo SurfOld, SurfNew;
DisplayPoint ClickOrigin, Axis1, Axis2;

//■頂点変換
void CalcMultMatVec(const GLfloat *xp, const GLfloat *sp, GLdouble *dp){
 dp[0] = xp[M11]*sp[0] + xp[M12]*sp[1] + xp[M13]*sp[2] + xp[M14];
 dp[1] = xp[M21]*sp[0] + xp[M22]*sp[1] + xp[M23]*sp[2] + xp[M24];
 dp[2] = xp[M31]*sp[0] + xp[M32]*sp[1] + xp[M33]*sp[2] + xp[M34];
}

//透視変換
void CalcScreenPosition(GLfloat x, GLfloat y, GLfloat z,
GLfloat *sx, GLfloat *sy)
{
  *sx = x*ysize/(2.0*K*(EyeDist-z));
  *sy = y*ysize/(2.0*K*(EyeDist-z));
}

void InitCubeInfo()
{
  int x, y, z;
```

```
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();

  for (x=0; x < 3; x++)
    for (y=0; y < 3; y++)
      for (z=0; z < 3; z++) {
glLoadIdentity();
glTranslatef((x-1), (y-1), (z-1));
glGetFloatv(GL_MODELVIEW_MATRIX, CubeData[x][y][z].Matrix);

  glPopMatrix();
}

void RotateCube90(int axis, int row)
{
  int p, q;
  CubeInfo CubeTemp[3][3];

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();

  switch (axis) {
  case 1 :
    for (q=0; q<3; q++)
      for (p=0; p<3; p++)
CubeTemp[p][q] = CubeData[row][p][q];
    for (q=0; q<3; q++)
      for (p=0; p<3; p++) {
CubeData[row][p][q] = CubeTemp[q][2-p];
glLoadIdentity();
glRotatef(90.0, 1.0, 0.0, 0.0);
glMultMatrixf(CubeData[row][p][q].Matrix);
glGetFloatv(GL_MODELVIEW_MATRIX, CubeData[row][p][q].Matrix);
      }
    break;
  case 2 :
    for (q=0; q<3; q++)
      for (p=0; p<3; p++)
CubeTemp[p][q] = CubeData[p][row][q];
    for (q=0; q<3; q++)
      for (p=0; p<3; p++) {
CubeData[p][row][q] = CubeTemp[2-q][p];
glLoadIdentity();
glRotatef(90.0, 0.0, 1.0, 0.0);
glMultMatrixf(CubeData[p][row][q].Matrix);
glGetFloatv(GL_MODELVIEW_MATRIX, CubeData[p][row][q].Matrix);
      }
    break;
  case 3 :
```

```
    for (q=0; q<3; q++)
      for (p=0; p<3; p++)
CubeTemp[p][q] = CubeData[p][q][row];
    for (q=0; q<3; q++)
      for (p=0; p<3; p++) {
CubeData[p][q][row] = CubeTemp[q][2-p];
glLoadIdentity();
glRotatef(90.0, 0.0, 0.0, 1.0);
glMultMatrixf(CubeData[p][q][row].Matrix);
glGetFloatv(GL_MODELVIEW_MATRIX, CubeData[p][q][row].Matrix);
      }
    break;
  }

  glPopMatrix();
}

void RotateCube(int axis, int row, float spin)
{
 while (spin > 45.0) {
   RotateCube90(axis, row);
   spin -= 90.0;
 }
}

int CalcRotateMode(SurfInfo *New, SurfInfo *Old, int *axis_p, int *row_p)
{
  // x が違う
  if (New->x != Old->x && New->y == Old->y && New->z == Old->z
      && New->color == Old->color) {
    // y 軸回転
    if (New->color == 0 || New->color == 1) {
      *axis_p = 2;
      *row_p = New->y;
      return 1;
    }
    // z 軸回転
    else if (New->color == 4 || New->color == 5) {
      *axis_p = 3;
      *row_p = New->z;
      return 1;
    }
    else return 0;
  }
  // y が違う
  else if (New->x == Old->x && New->y != Old->y && New->z == Old->z
   && New->color == Old->color) {
    // x 軸回転
    if (New->color == 0 || New->color == 1) {
      *axis_p = 1;
```

```
      *row_p = New->x;
      return 1;
    }
    // z 軸回転
    else if (New->color == 2 || New->color == 3) {
      *axis_p = 3;
      *row_p = New->z;
      return 1;
    }
    else return 0;
  }
  // z が違う
  else if (New->x == Old->x && New->y == Old->y && New->z != Old->z
   && New->color == Old->color) {
    // x 軸回転
    if (New->color == 4 || New->color == 5) {
      *axis_p = 1;
      *row_p = New->x;
      return 1;
    }
    // y 軸回転
    else if (New->color == 2 || New->color == 3) {
      *axis_p = 2;
      *row_p = New->y;
      return 1;
    }
    else return 0;
  }
  // 同じキューブ
  else if (New->x == Old->x && New->y == Old->y && New->z == Old->z
   && New->color != Old->color) {
    // x 軸回転
    if ((New->color == 0 && Old->color == 4) ||
(New->color == 4 && Old->color == 1) ||
(New->color == 1 && Old->color == 5) ||
(New->color == 5 && Old->color == 0) ||

(New->color == 4 && Old->color == 0) ||
(New->color == 1 && Old->color == 4) ||
(New->color == 5 && Old->color == 1) ||
(New->color == 0 && Old->color == 5)) {
      *axis_p = 1;
      *row_p = New->x;
      return 1;
    }
    // y 軸回転
    else if ((New->color == 0 && Old->color == 3) ||
     (New->color == 3 && Old->color == 1) ||
     (New->color == 1 && Old->color == 2) ||
     (New->color == 2 && Old->color == 0) ||
```

```
     (New->color == 3 && Old->color == 0) ||
     (New->color == 1 && Old->color == 3) ||
     (New->color == 2 && Old->color == 1) ||
     (New->color == 0 && Old->color == 2)) {
      *axis_p = 2;
      *row_p = New->y;
      return 1;
    }
    // z 軸回転
    else if ((New->color == 2 && Old->color == 4) ||
     (New->color == 4 && Old->color == 3) ||
     (New->color == 3 && Old->color == 5) ||
     (New->color == 5 && Old->color == 2) ||

     (New->color == 4 && Old->color == 2) ||
     (New->color == 3 && Old->color == 4) ||
     (New->color == 5 && Old->color == 3) ||
     (New->color == 2 && Old->color == 5)) {
      *axis_p = 3;
      *row_p = New->z;
      return 1;
    }
    else return 0;
  }
  else return 0;
}

void CalcWorldToScreen(const GLfloat *wp, GLfloat *sx, GLfloat *sy)
{
  //int i;
  // スクリーン座標に変換
  GLdouble ScreenPoint[3];

  CalcMultMatVec(CubeViewMatrix, wp, ScreenPoint);

  CalcScreenPosition(ScreenPoint[0], ScreenPoint[1], ScreenPoint[2],
   sx, sy);
}

// 触った面に矢印を表示
void DrawRotateVector(GLfloat cursor_x, GLfloat cursor_y)
{
  int i;
  GLfloat TouchSurfacePoint[3], //TouchSurfaceNormal[3],
          TouchSurfaceVector1[3], TouchSurfaceVector2[3];

  // SurfNew によりセレクトされている面を決定し
  // 点と法線を指定することにより平面の方程式を得る
  for (i=0; i<3; i++)
```

```
    TouchSurfacePoint[i] //= TouchSurfaceNormal[i]
      = TouchSurfaceVector1[i] = TouchSurfaceVector2[i] = 0.0;

  switch (SurfNew.color) {
  case 0 : TouchSurfacePoint[2] = -0.5; //TouchSurfaceNormal[2] = -1.0;
   TouchSurfaceVector1[0] = -1.0; TouchSurfaceVector2[1] = -1.0; break;
  case 1 : TouchSurfacePoint[2] =  0.5; //TouchSurfaceNormal[2] =  1.0;
   TouchSurfaceVector1[0] = 1.0; TouchSurfaceVector2[1] = 1.0; break;
  case 2 : TouchSurfacePoint[0] =  0.5; //TouchSurfaceNormal[0] =  1.0;
   TouchSurfaceVector1[1] = 1.0; TouchSurfaceVector2[2] = 1.0; break;
  case 3 : TouchSurfacePoint[0] = -0.5; //TouchSurfaceNormal[0] = -1.0;
   TouchSurfaceVector1[1] = -1.0; TouchSurfaceVector2[2] = -1.0; break;
  case 4 : TouchSurfacePoint[1] =  0.5; //TouchSurfaceNormal[1] =  1.0;
   TouchSurfaceVector1[2] = 1.0; TouchSurfaceVector2[0] = 1.0; break;
  case 5 : TouchSurfacePoint[1] = -0.5; //TouchSurfaceNormal[1] = -1.0;
   TouchSurfaceVector1[2] = -1.0; TouchSurfaceVector2[0] = -1.0; break;
 }

  // SurfNew により点を平行移動
  TouchSurfacePoint[0] += SurfNew.x - 1.0;
  TouchSurfacePoint[1] += SurfNew.y - 1.0;
  TouchSurfacePoint[2] += SurfNew.z - 1.0;

  GLfloat Axis1Temp[3], Axis2Temp[3];

#define VEC_SCALE 2.0

  for (i=0; i<3; i++) {
    Axis1Temp[i] = TouchSurfacePoint[i] + VEC_SCALE*TouchSurfaceVector1[i];
    Axis2Temp[i] = TouchSurfacePoint[i] + VEC_SCALE*TouchSurfaceVector2[i];
  }

  // スクリーン座標に変換
  CalcWorldToScreen(TouchSurfacePoint, &ClickOrigin.x, &ClickOrigin.y);
  CalcWorldToScreen(Axis1Temp, &Axis1.x, &Axis1.y);
  CalcWorldToScreen(Axis2Temp, &Axis2.x, &Axis2.y);

  // 位置ベクトル情報
  Axis1.x -= ClickOrigin.x;
  Axis1.y -= ClickOrigin.y;
  Axis2.x -= ClickOrigin.x;
  Axis2.y -= ClickOrigin.y;
}

void CalcRotateAxis(DisplayPoint *co, DisplayPoint *a1, DisplayPoint *a2,
    GLfloat *cursor_x, GLfloat *cursor_y)
{
  GLfloat h1, h2;

  h1 = fabs(a1->y * (*cursor_x - co->x) - a1->x * (*cursor_y - co->y))
```

```
    / sqrt(a1->y * a1->y + a1->x * a1->x);
  h2 = fabs(a2->y * (*cursor_x - co->x) - a2->x * (*cursor_y - co->y))
    / sqrt(a2->y * a2->y + a2->x * a2->x);
}

void CalcRotateAngle(DisplayPoint *LastClick,
     DisplayPoint *Axis1, DisplayPoint *Axis2,
     GLfloat *cursor_x, GLfloat *cursor_y,
     GLfloat *r1, GLfloat *r2)
{
  DisplayPoint a1, a2, a3;

  a1.x = Axis1->x;
  a1.y = Axis1->y;
  a2.x = Axis2->x;
  a2.y = Axis2->y;
  a3.x = *cursor_x - LastClick->x;
  a3.y = *cursor_y - LastClick->y;

  if (1) {
    *r1 = (a2.y * a3.x - a2.x * a3.y)
      / (a1.x * a2.y - a2.x * a1.y);
    *r2 = (a1.y * a3.x - a1.x * a3.y)
      / (a1.x * a2.y - a2.x * a1.y);
  }
  else {
    *r1 = 0.0;
    *r2 = 0.0;
  }
}

void DrawRotateLine(GLfloat cursor_x, GLfloat cursor_y)
{
  int i;
  GLfloat TouchSurfacePoint[3], TouchSurfaceNormal[3], tmpP[3], tmpN[3];

  // mynumsuf[4] によりセレクトされている面を決定し
  // 点と法線を指定することにより平面の方程式を得る
  for (i=0; i<3; i++) TouchSurfacePoint[i] = TouchSurfaceNormal[i] = 0.0;

  switch (SurfNew.color) {
  case 0 : TouchSurfacePoint[2] = -0.5; TouchSurfaceNormal[2] = -1.0; break;
  case 1 : TouchSurfacePoint[2] =  0.5; TouchSurfaceNormal[2] =  1.0; break;
  case 2 : TouchSurfacePoint[0] =  0.5; TouchSurfaceNormal[0] =  1.0; break;
  case 3 : TouchSurfacePoint[0] = -0.5; TouchSurfaceNormal[0] = -1.0; break;
  case 4 : TouchSurfacePoint[1] =  0.5; TouchSurfaceNormal[1] =  1.0; break;
  case 5 : TouchSurfacePoint[1] = -0.5; TouchSurfaceNormal[1] = -1.0; break;
 }

  // mynumsuf[2],mynumsuf[1],mynumsuf[0] により点を平行移動
```

```
  //TouchSurfacePoint[0] += mynumsuf[2] - 1.0;
  //TouchSurfacePoint[1] += mynumsuf[1] - 1.0;
  //TouchSurfacePoint[2] += mynumsuf[0] - 1.0;
  TouchSurfacePoint[0] += SurfNew.x - 1.0;
  TouchSurfacePoint[1] += SurfNew.y - 1.0;
  TouchSurfacePoint[2] += SurfNew.z - 1.0;

  // 点を CubeViewMatrix[] と乗算して現在の位置に変換
  tmpP[0] = CubeViewMatrix[ 0] * TouchSurfacePoint[0] +
    CubeViewMatrix[ 4] * TouchSurfacePoint[1] +
      CubeViewMatrix[ 8] * TouchSurfacePoint[2];
  tmpP[1] = CubeViewMatrix[ 1] * TouchSurfacePoint[0] +
    CubeViewMatrix[ 5] * TouchSurfacePoint[1] +
      CubeViewMatrix[ 9] * TouchSurfacePoint[2];
  tmpP[2] = CubeViewMatrix[ 2] * TouchSurfacePoint[0] +
    CubeViewMatrix[ 6] * TouchSurfacePoint[1] +
      CubeViewMatrix[10] * TouchSurfacePoint[2];
  for (i=0; i<3; i++) TouchSurfacePoint[i] = tmpP[i];

  // 法線を CubeViewMatrix[] と乗算して現在の姿勢に変換
  tmpN[0] = CubeViewMatrix[ 0] * TouchSurfaceNormal[0] +
    CubeViewMatrix[ 4] * TouchSurfaceNormal[1] +
      CubeViewMatrix[ 8] * TouchSurfaceNormal[2];
  tmpN[1] = CubeViewMatrix[ 1] * TouchSurfaceNormal[0] +
    CubeViewMatrix[ 5] * TouchSurfaceNormal[1] +
      CubeViewMatrix[ 9] * TouchSurfaceNormal[2];
  tmpN[2] = CubeViewMatrix[ 2] * TouchSurfaceNormal[0] +
    CubeViewMatrix[ 6] * TouchSurfaceNormal[1] +
      CubeViewMatrix[10] * TouchSurfaceNormal[2];
  for (i=0; i<3; i++) TouchSurfaceNormal[i] = tmpN[i];
}

void ModelDraw(void)
{
  /* Draw Model */
  /* Clear Window (color buffer and depth buffer) */
  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glMultMatrixf(CubeViewMatrix);

  glScalef(scale,scale,scale);
  //CoordXYZ(1.0);
  if (!Spin && !Drag)
    DrawRotateVector(1.0, 1.0);

  DrawRubik(1.0, DRAW_DISPLAY, SpinAxis, SpinRow, spin_y);
  if (!Spin && !Drag)
    DrawRubikPointer(1.0, DRAW_DISPLAY, SpinAxis, SpinRow, spin_y);
```

```
  glPopMatrix ();

  if (!Spin && !Drag)
    DrawRotateLine(1.0, 1.0);

  if (Spin) {
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glLoadIdentity();
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    gluOrtho2D(0.0, xsize, 0.0, ysize);
    glTranslatef(xsize/2.0, ysize/2.0, 0.0);

    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glPopMatrix();
  }

  glFlush();
  glutSwapBuffers();
}

void ModelSpin(int x, int y)
{
  if (MouseON)
    {
      if (Spin)
{
  if ((last_x != x) || (last_y != y))
    {
      last_x = x;
      last_y = y;

      // 回転軸の計算
      int axis1, axis2;
      GLfloat cursor_x = x-xsize/2.0, cursor_y = -(y-ysize/2.0);
      GLfloat r1, r2;
      // 最初にクリックした点
      DisplayPoint LastClick;
      LastClick.x = mylast_x-xsize/2.0;
      LastClick.y = -(mylast_y-ysize/2.0);

      CalcTwoAxis(&SurfNew, &axis1, &axis2);
      CalcRotateAxis(&LastClick, &Axis1, &Axis2,
     &cursor_x, &cursor_y);
      CalcRotateAngle(&LastClick, &Axis1, &Axis2,
      &cursor_x, &cursor_y, &r1, &r2);
```

```
#define ROTATE_CONST 90.0
#define ROTATE_MIN 0.05
#define LOCK_MIN 0.05
      if (RotateLock == 0) {
if (fabs(r1) < ROTATE_MIN && fabs(r2) < ROTATE_MIN)
  SpinAxis =0;
else {
  if (fabs(r1) > fabs(r2)) {
    spin_y = ROTATE_CONST * r1;
    SpinAxis = axis2 + 1;
    if (fabs(r1) > LOCK_MIN)
      RotateLock = 1;
  }
  else {
    spin_y = ROTATE_CONST * r2;
    SpinAxis = axis1 + 1;
    if (fabs(r2) > LOCK_MIN)
      RotateLock = 2;
  }
}
      }
      else {
if (RotateLock == 1)
  spin_y = ROTATE_CONST * r1;
else
  spin_y = ROTATE_CONST * r2;
      }

      switch (SpinAxis) {
      case 1 : SpinRow = SurfNew.x; break;
      case 2 : SpinRow = SurfNew.y; break;
      case 3 : SpinRow = SurfNew.z; break;
     }

      glMatrixMode(GL_MODELVIEW);
      glPushMatrix();
      glMultMatrixf(CubeViewMatrix);

      glPopMatrix ();

      glutPostRedisplay();
      glFlush();
    }
}
      else if (Drag)
{
  if ((last_x != x) || (last_y != y))
    {
      whole_spin_x =  1.0 * (x - last_x);
      whole_spin_y =  1.0 * (y - last_y);
```

```
      last_x = x;
      last_y = y;
      glMatrixMode(GL_MODELVIEW);
      glPushMatrix();
      glLoadIdentity();
      glRotatef(whole_spin_x, 0.0, 1.0, 0.0);
      glRotatef(whole_spin_y, 1.0, 0.0, 0.0);
      glMultMatrixf(CubeViewMatrix);
      glGetFloatv(GL_MODELVIEW_MATRIX, CubeViewMatrix);
      glPopMatrix();
      glutPostRedisplay();
    }
}
      else if (Scale)
{
  if (last_y != y)
    {
      scale += 3.0*2.0*(y - last_y)/ysize;
      if (scale <= 0.1) scale = 0.1;
      glutPostRedisplay();
      last_x = x;
      last_y = y;
    }
}
    }
}

void TouchCube(int x, int y)
{
  if (!MouseON && ((last_x != x) || (last_y != y))) {
    SelectCube(x, y, &SurfNew, &SurfOld);

    glutPostRedisplay();
  }
}

void mouse(int button, int state, int x, int y)
{
  if (button == GLUT_LEFT_BUTTON)
    {
      if (state == GLUT_DOWN)
{
  SelectCube(x, y, &SurfNew, &SurfOld);
  Spin = 1;
  MouseON = 1;
  mylast_x = last_x = x;
  mylast_y = last_y = y;
}
      else if (state == GLUT_UP)
{
```

```
  Spin = 0;
  MouseON = 0;
  RotateLock = 0; //回転のロック解除
  while (spin_y < 0.0) spin_y += 360;
  if (spin_y >= 0.0 && spin_y < 45.0) spin_y = 0.0;
  else if (spin_y >= 45.0 && spin_y < 135.0) spin_y = 90.0;
  else if (spin_y >= 135.0 && spin_y < 225.0) spin_y = 180.0;
  else if (spin_y >= 225.0 && spin_y < 315.0) spin_y = 270.0;
  else if (spin_y >= 360.0 && spin_y < 360.0) spin_y = 0.0;
  RotateCube(SpinAxis, SpinRow, spin_y);
  spin_y = 0.0;
  glutPostRedisplay();
}
    }
  else if (button == GLUT_MIDDLE_BUTTON)
    {
      if (state == GLUT_DOWN)
{
  Drag = 1;
  MouseON = 1;
  last_x = x;
  last_y = y;
}
      else if (state == GLUT_UP)
{
  Drag = 0;
  MouseON = 0;
}
    }
  else if (button == GLUT_RIGHT_BUTTON)
    {
    }
}

// キーボード・イベントの処理
void keyboard(unsigned char key, int x, int y)
{
  if (toupper(key) == 'Q')
    exit(0);
}

void ReshapeWindow(int width, int height)
{
  /* Event handling for Window */

  /* VIEW PORT SETUP */
  glViewport(0, 0, width, height);

  /* PROJECTION MATRIX SETUP */
  glMatrixMode(GL_PROJECTION);
```

```
  glLoadIdentity();
  gluPerspective(40.0, (GLfloat)width/(GLfloat)height, 0.1, 20.0);

  /* MODEL VIEW MATRIX SETUP */
  glMatrixMode(GL_MODELVIEW);
  /* Reset Modelview matrix */
  glLoadIdentity();
  glGetFloatv(GL_MODELVIEW_MATRIX, CubeViewMatrix);
  gluLookAt(0.0,0.0,EyeDist, 0.0,0.0,0.0, 0.0,1.0,0.0);

  /* Set Light Posision */
  glLightfv(GL_LIGHT0,GL_POSITION,Light0.posision);

  /* Save Now Window size for Mouse evnet */
  xsize = width;
  ysize = height;
}

void main(int argc, char** argv)
{
  /* Window Setup using glut */
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
  glutInitWindowPosition(0, 0);
  glutInitWindowSize(xsize,ysize);
  glutCreateWindow("Cubism ver 3.1");
  //glutCreateWindow(argv[0]);

  glHint(GL_LINE_SMOOTH_HINT,GL_FASTEST);
  glEnable(GL_LINE_SMOOTH);
  glHint(GL_POINT_SMOOTH_HINT,GL_FASTEST);
  glEnable(GL_POINT_SMOOTH);

  glClearColor (1.0, 0.9, 0.8, 1.0);
  //glClearColor (0.7, 0.7, 0.5, 1.0);

  /* Set Light model,Light,Material Parameter */
  SetLight_MaterialParameter();

  /* Set Rubik's Cube -- Material Parameter */
  SetMyMaterialParameter();

  /* Set Light model */
  SetLightModel();
  /* Set Light */
  SetLight();
  /* Set Material */
  SetMaterial();

  /* LIGHTING ON */
```

```
  glEnable(GL_LIGHTING);
  /* LIGHT NO.0 ON */
  glEnable(GL_LIGHT0);

  /* DEPTH TEST ON */
  glEnable(GL_DEPTH_TEST);
  /* NORMALIZE ON */
  glEnable(GL_NORMALIZE);

  /* Polygon mode set */
  /* Front face Fill */
  glPolygonMode(GL_FRONT,GL_FILL);
  /* Back face Line (Only wireframe mode) */
  glPolygonMode(GL_BACK,GL_LINE);

  /* BackFace Culling */
  glCullFace(GL_BACK);
  /* Culling mode set */
  glEnable(GL_CULL_FACE);

  /* alpha blending */
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
  glEnable(GL_BLEND);

  // キューブ情報の初期化
  InitCubeInfo();

  glutMotionFunc(ModelSpin);
  glutPassiveMotionFunc(TouchCube);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboard);
  glutReshapeFunc(ReshapeWindow);
  glutDisplayFunc(ModelDraw);
  glutMainLoop();
}
```

A.2 rubik.c

```
/*
   rubik.c
   Program by Takashi Oshiba <ohshiba@softlab.is.tsukuba.ac.jp>

   for IRIX 6.3
*/

#include <GL/gl.h>
#include "Cubism.h"
#include "draw.h"
#include "rubik.h"
```

```
#include "select.h"

// ルービックキューブ全体を描画
// 通常の描画モード DRAW_DISPLAY と,
// ヒットチェックモード DRAW_SELECT の２種類の動作をする。
void DrawRubik(GLfloat length, int mode, int axis, int row, float spin)
     // axis 0: 関係なし 1:x 2:y 3:z
{
  int x, y, z;

  if (mode == DRAW_DISPLAY) {
    switch (axis) {
    case 0 :
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    DrawCube(length, x, y, z);
      break;

    case 1 :
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (x != row)
      DrawCube(length, x, y, z);
      glPushMatrix ();
      glRotatef(spin, 1.0, 0.0, 0.0);
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (x == row)
      DrawCube(length, x, y, z);
      glPopMatrix ();
      break;

    case 2 :
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (y != row)
      DrawCube(length, x, y, z);
      glPushMatrix ();
      glRotatef(spin, 0.0, 1.0, 0.0);
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (y == row)
      DrawCube(length, x, y, z);
      glPopMatrix ();
      break;
```

```
    case 3 :
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (z != row)
      DrawCube(length, x, y, z);
      glPushMatrix ();
      glRotatef(spin, 0.0, 0.0, 1.0);
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (z == row)
      DrawCube(length, x, y, z);
      glPopMatrix ();
      break;
    }
  }
  else if (mode == DRAW_SELECT) {
    switch (axis) {
    case 0 :
      for (z=0; z < 3; z++) {
glPushName(z);
for (y=0; y < 3; y++) {
  glPushName(y);
  for (x=0; x < 3; x++) {
    glPushName(x);
    DrawHCCube(length, x, y, z);
    glPopName();
  }
  glPopName();
}
glPopName();
      }
      break;

    case 1 :
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (x != row)
      DrawHCCube(length, x, y, z);
      glPushMatrix ();
      glRotatef(spin, 1.0, 0.0, 0.0);
      for (z=0; z < 3; z++) {
glPushName(z);
for (y=0; y < 3; y++) {
  glPushName(y);
  for (x=0; x < 3; x++) {
    if (x == row) {
```

```
      glPushName(x);
      DrawHCCube(length, x, y, z);
      glPopName();
    }
  }
  glPopName();
}
glPopName();
      }
      glPopMatrix ();
      break;

    case 2 :
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (y != row)
      DrawHCCube(length, x, y, z);
      glPushMatrix ();
      glRotatef(spin, 0.0, 1.0, 0.0);
      for (z=0; z < 3; z++) {
glPushName(z);
for (y=0; y < 3; y++) {
  glPushName(y);
  for (x=0; x < 3; x++) {
    if (y == row) {
      glPushName(x);
      DrawHCCube(length, x, y, z);
      glPopName();
    }
  }
  glPopName();
}
glPopName();
      }
      glPopMatrix ();
      break;

    case 3 :
      for (z=0; z < 3; z++)
for (y=0; y < 3; y++)
  for (x=0; x < 3; x++)
    if (z != row)
      DrawHCCube(length, x, y, z);
      glPushMatrix ();
      glRotatef(spin, 0.0, 0.0, 1.0);
      for (z=0; z < 3; z++) {
glPushName(z);
for (y=0; y < 3; y++) {
  glPushName(y);
```

```
  for (x=0; x < 3; x++) {
    if (z == row) {
      glPushName(x);
      DrawHCCube(length, x, y, z);
      glPopName();
    }
  }
  glPopName();
}
glPopName();
      }
      glPopMatrix ();
      break;
    }
  }
}
```

A.3 select.c

```
/*
   select.c
   Program by Takashi Oshiba <ohshiba@softlab.is.tsukuba.ac.jp>

   for IRIX 6.3
*/

#include <GL/gl.h>
#include <GL/glu.h>
#include "Cubism.h"
#include "draw.h"
#include "rubik.h"
#include "select.h"
#include "modeldata.h"

void DrawHCCube(GLfloat length, int x, int y, int z)
{
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glTranslatef((x-1)*length, (y-1)*length, (z-1)*length);

  HCCube(length);
  glPopMatrix();
}

void CalcTwoAxis(SurfInfo *Surf, int *axis1, int *axis2)
{
  switch (Surf->color) {
  case 0 : *axis1 = 0; *axis2 = 1; break;
  case 1 : *axis1 = 0; *axis2 = 1; break;
```

```
  case 2 : *axis1 = 1; *axis2 = 2; break;
  case 3 : *axis1 = 1; *axis2 = 2; break;
  case 4 : *axis1 = 2; *axis2 = 0; break;
  case 5 : *axis1 = 2; *axis2 = 0; break;
 }
}

// マウスによるピック
void SelectCube(int x, int y, SurfInfo *New, SurfInfo *Old)
{
  GLuint selbuf[BUFFERSIZE]; // セレクションバッファ
  GLint hits;
  GLint viewport[4];
  GLuint *p, zmin;
  int i, j;
  GLint numsuf[4]; // 面の番号（面のコード）

  glGetIntegerv(GL_VIEWPORT, viewport); // ビューポート情報の取得

  glSelectBuffer(BUFFERSIZE, selbuf);
  glRenderMode(GL_SELECT);
  glInitNames(); // ネームスタックを空にする

  glMatrixMode(GL_PROJECTION);
  glPushMatrix();

  glLoadIdentity();

  // ピックの処理
  gluPickMatrix((GLdouble) x, (GLdouble) (viewport[3] - y),
2.0, 2.0, viewport);
  gluPerspective(40.0, (GLfloat) viewport[2]/(GLfloat) viewport[3], 0.1, 20.0);

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glMultMatrixf(CubeViewMatrix);

  /* Change modelview matrix */
  glScalef(scale,scale,scale);

  //DrawRubik(1.0, DRAW_SELECT);
  DrawRubik(1.0, DRAW_SELECT, 0, 0, 0.0);

  glPopMatrix();

  glMatrixMode(GL_PROJECTION);
  glPopMatrix();
  glFlush();

  // 戻り値としてセレクションヒットの数が返ってくる
```

```
  hits = glRenderMode(GL_RENDER);
  //printf("hits= %d\n", hits);

  zmin = 4294967295;
  p = (GLuint *) selbuf;

  numsuf[0] = -1;
  for (j=0; j<hits; j++) {
    GLuint /*num,*/ z0/*, z1*/;
    /*num =*/ *p++;
    z0 = *p++;
    /*z1 =*/ *p++;

    if (z0 < zmin) {
      zmin = z0;
      for (i=0; i<4; i++)
numsuf[i] = p[i];
    }
    p += 4;
  }

  // ヒットがあったら.....
  int axis1, axis2;
  if (numsuf[0] >= 0) {
    CalcTwoAxis(New, &axis1, &axis2);
    //printf("axis1=%d, axis2=%d\n", axis1, axis2);
    if (New->x != numsuf[2] || New->y != numsuf[1] ||
New->z != numsuf[0] || New->color != numsuf[3]) {
      Old->x = New->x;
      Old->y = New->y;
      Old->z = New->z;
      Old->color = New->color;

      New->x = numsuf[2];
      New->y = numsuf[1];
      New->z = numsuf[0];
      New->color = numsuf[3];
    }
  }
}
```

A.4 draw.c

```
/*
   draw.c
   Program by Takashi Oshiba <ohshiba@softlab.is.tsukuba.ac.jp>

   for IRIX 6.3
*/
```

```
#include <GL/gl.h>
#include <math.h>
#include "Cubism.h"
#include "material.h"
#include "draw.h"
#include "rubik.h"
#include "modeldata.h"

extern CubeInfo CubeData[3][3][3];
extern SurfInfo SurfOld, SurfNew;

// キューブを 1 つだけ描画
void DrawCube(GLfloat length, int x, int y, int z)
{
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glMultMatrixf(CubeData[x][y][z].Matrix);

  Cube(length);

  glPopMatrix();
}

// マウスカーソルが触った面の外側の枠を少し白くする
void DrawPointer(int length, int x, int y, int z)
{
  GLfloat half = fabs(length) / 2.0;

  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glTranslatef((x-1)*length, (y-1)*length, (z-1)*length);

  switch (SurfNew.color) {
  case 0 :
    // orange
    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,0.0,-1.0);
    glVertex3f(half,half,-half);
    glVertex3f(half,-half,-half);
    glVertex3f(8.0*half/9.0,-8.0*half/9.0,-half);
    glVertex3f(8.0*half/9.0,8.0*half/9.0,-half);
    glEnd();

    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,0.0,-1.0);
    glVertex3f(half,-half,-half);
    glVertex3f(-half,-half,-half);
```

```
    glVertex3f(-8.0*half/9.0,-8.0*half/9.0,-half);
    glVertex3f(8.0*half/9.0,-8.0*half/9.0,-half);
    glEnd();

    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,0.0,-1.0);
    glVertex3f(-half,-half,-half);
    glVertex3f(-half,half,-half);
    glVertex3f(-8.0*half/9.0,8.0*half/9.0,-half);
    glVertex3f(-8.0*half/9.0,-8.0*half/9.0,-half);
    glEnd();

    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,0.0,-1.0);
    glVertex3f(-half,half,-half);
    glVertex3f(half,half,-half);
    glVertex3f(8.0*half/9.0,8.0*half/9.0,-half);
    glVertex3f(-8.0*half/9.0,8.0*half/9.0,-half);
    glEnd();
    break;

  case 1 :
    // red
    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,0.0,1.0);
    glVertex3f(-half,half,half);
    glVertex3f(-half,-half,half);
    glVertex3f(-8.0*half/9.0,-8.0*half/9.0,half);
    glVertex3f(-8.0*half/9.0,8.0*half/9.0,half);
    glEnd();

    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,0.0,1.0);
    glVertex3f(-half,-half,half);
    glVertex3f(half,-half,half);
    glVertex3f(8.0*half/9.0,-8.0*half/9.0,half);
    glVertex3f(-8.0*half/9.0,-8.0*half/9.0,half);
    glEnd();

    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,0.0,1.0);
    glVertex3f(half,-half,half);
    glVertex3f(half,half,half);
    glVertex3f(8.0*half/9.0,8.0*half/9.0,half);
    glVertex3f(8.0*half/9.0,-8.0*half/9.0,half);
```

```
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(0.0,0.0,1.0);
  glVertex3f(half,half,half);
  glVertex3f(-half,half,half);
  glVertex3f(-8.0*half/9.0,8.0*half/9.0,half);
  glVertex3f(8.0*half/9.0,8.0*half/9.0,half);
  glEnd();
  break;

case 2 :
  // green
  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(1.0,0.0,0.0);
  glVertex3f(half,half,half);
  glVertex3f(half,-half,half);
  glVertex3f(half,-8.0*half/9.0,8.0*half/9.0);
  glVertex3f(half,8.0*half/9.0,8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(1.0,0.0,0.0);
  glVertex3f(half,-half,half);
  glVertex3f(half,-half,-half);
  glVertex3f(half,-8.0*half/9.0,-8.0*half/9.0);
  glVertex3f(half,-8.0*half/9.0,8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(1.0,0.0,0.0);
  glVertex3f(half,-half,-half);
  glVertex3f(half,half,-half);
  glVertex3f(half,8.0*half/9.0,-8.0*half/9.0);
  glVertex3f(half,-8.0*half/9.0,-8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(1.0,0.0,0.0);
  glVertex3f(half,half,-half);
  glVertex3f(half,half,half);
  glVertex3f(half,8.0*half/9.0,8.0*half/9.0);
  glVertex3f(half,8.0*half/9.0,-8.0*half/9.0);
  glEnd();
  break;
```

```
case 3 :
  // yellow
  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(-1.0,0.0,0.0);
  glVertex3f(-half,half,-half);
  glVertex3f(-half,-half,-half);
  glVertex3f(-half,-8.0*half/9.0,-8.0*half/9.0);
  glVertex3f(-half,8.0*half/9.0,-8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(-1.0,0.0,0.0);
  glVertex3f(-half,-half,-half);
  glVertex3f(-half,-half,half);
  glVertex3f(-half,-8.0*half/9.0,8.0*half/9.0);
  glVertex3f(-half,-8.0*half/9.0,-8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(-1.0,0.0,0.0);
  glVertex3f(-half,-half,half);
  glVertex3f(-half,half,half);
  glVertex3f(-half,8.0*half/9.0,8.0*half/9.0);
  glVertex3f(-half,-8.0*half/9.0,8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(-1.0,0.0,0.0);
  glVertex3f(-half,half,half);
  glVertex3f(-half,half,-half);
  glVertex3f(-half,8.0*half/9.0,-8.0*half/9.0);
  glVertex3f(-half,8.0*half/9.0,8.0*half/9.0);
  glEnd();
  break;

case 4 :
  // blue
  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(0.0,1.0,0.0);
  glVertex3f(-half,half,-half);
  glVertex3f(-half,half,half);
  glVertex3f(-8.0*half/9.0,half,8.0*half/9.0);
  glVertex3f(-8.0*half/9.0,half,-8.0*half/9.0);
  glEnd();
```

```
  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(0.0,1.0,0.0);
  glVertex3f(-half,half,half);
  glVertex3f(half,half,half);
  glVertex3f(8.0*half/9.0,half,8.0*half/9.0);
  glVertex3f(-8.0*half/9.0,half,8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(0.0,1.0,0.0);
  glVertex3f(half,half,half);
  glVertex3f(half,half,-half);
  glVertex3f(8.0*half/9.0,half,-8.0*half/9.0);
  glVertex3f(8.0*half/9.0,half,8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(0.0,1.0,0.0);
  glVertex3f(half,half,-half);
  glVertex3f(-half,half,-half);
  glVertex3f(-8.0*half/9.0,half,-8.0*half/9.0);
  glVertex3f(8.0*half/9.0,half,-8.0*half/9.0);
  glEnd();
  break;

case 5 :
  // white
  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(0.0,-1.0,0.0);
  glVertex3f(-half,-half,half);
  glVertex3f(-half,-half,-half);
  glVertex3f(-8.0*half/9.0,-half,-8.0*half/9.0);
  glVertex3f(-8.0*half/9.0,-half,8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
  glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
  glNormal3f(0.0,-1.0,0.0);
  glVertex3f(-half,-half,-half);
  glVertex3f(half,-half,-half);
  glVertex3f(8.0*half/9.0,-half,-8.0*half/9.0);
  glVertex3f(-8.0*half/9.0,-half,-8.0*half/9.0);
  glEnd();

  glBegin(GL_POLYGON);
```

```
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,-1.0,0.0);
    glVertex3f(half,-half,-half);
    glVertex3f(half,-half,half);
    glVertex3f(8.0*half/9.0,-half,8.0*half/9.0);
    glVertex3f(8.0*half/9.0,-half,-8.0*half/9.0);
    glEnd();

    glBegin(GL_POLYGON);
    glMaterialfv(GL_FRONT_AND_BACK,GL_DIFFUSE,TransparentWhite.diffuse);
    glNormal3f(0.0,-1.0,0.0);
    glVertex3f(half,-half,half);
    glVertex3f(-half,-half,half);
    glVertex3f(-8.0*half/9.0,-half,8.0*half/9.0);
    glVertex3f(8.0*half/9.0,-half,8.0*half/9.0);
    glEnd();
    break;
  }

  glPopMatrix();
}

// マウスカーソルが触った面の外側の枠を少し白くする
void DrawRubikPointer(GLfloat length, int mode, int axis, int row, float spin)
{
  int x, y, z;

  glDisable(GL_DEPTH_TEST);
  if (mode == DRAW_DISPLAY) {
    for (z=0; z < 3; z++)
      for (y=0; y < 3; y++)
for (x=0; x < 3; x++)
  if (x == SurfNew.x && y == SurfNew.y && z == SurfNew.z) {
    DrawPointer(length, x, y, z);
  }
  }
  glEnable(GL_DEPTH_TEST);
}
```