無重力状態での剛体の回転運動と可視化

川畑　茂徳, 緒方　隆一, 古賀　俊崇
福岡工業大学　電子情報工学科

平成 15 年 4 月 28 日

はしがき

1992 年に放映された NHK の「毛利衛さんの宇宙報告」という番組で，スペースシャトルの中でペンチを回す実験の映像を見た人もいるでしょう．ペンチは始め図 1 のように回転しながら浮かんでいるが，数秒たつと上下が入れ替わって図 2 のようになり，また数秒たつと元の図 1 に戻るという運動を繰り返す．

本稿ではこのような運動を剛体の回転運動のコンピュータシミュレーションで再現してみる．ペンチに固定された座標系 a,b,c 軸を導入しよう．ペンチの中心軸を c 軸とする．ペンチの面内に，重心を通り中心軸に直交する軸を考え，これを b 軸とする．これらに直交するもうひとつの軸を a 軸とする．中心軸 c の周りを回転しながら上下反転するの仕組みは，a 軸，b 軸，c 軸まわりの慣性モーメントを I_a, I_b, I_c とするとき $I_b < I_c < I_a$ の関係になっていると，回転は不安定になることが知られている．実際，回転軸の方向と回転の周期を与えると回転は決まり，大きな軌道を描いてペンチは反転し始める．

構成としては，第 1 章では，はじめに，剛体の回転運動の文献を解説する．次に，回転運動をシミュレートするプログラミング言語の選定について述べる．第 2 章ではプログラミングにおける問題点とその対策について述べる．無重力状態での剛体の回転を記述する Vector クラス，Matrix クラスの導入，ベクトル微分方程式に対する Runge–Kutta 法，数値シミュレーション結果の可視化について述べる．

この記事は 2000 年度に行なわれた緒方隆一，古賀俊崇君の卒業研究に一部手を加えて再構成したものである．しかし，両君の卒業研究を可能な限り尊重し，修正は最小限にとどめた．

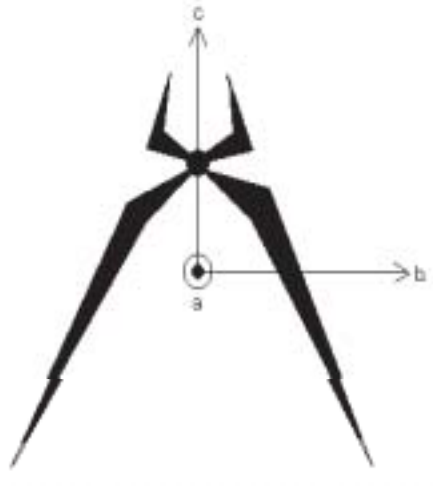

図 1: ペンチの上下反転

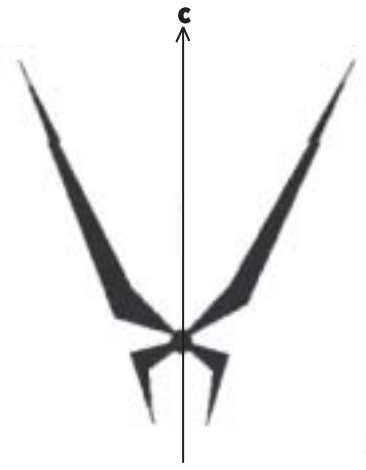

図 2: ペンチの上下反転

目 次

第1章　準備

1.1　現象のモデル化の文献案内

研究室では，現象のモデルとモデリングに関する卒業研究を継続しておこなっているが，2000年度のテーマの一つが「無重力状態での剛体の回転運動」であった．現実の問題をモデル化しシミュレートするために，われわれは単純化した仮定をたくさん立て，現実の問題を数学の問題に書き換える．現実の問題をなんらかの方法でモデル化し，コンピュータ上で計算できる形に翻訳しなければならない．モデルがなければ一歩も進めないし，プログラムも書きようがない．したがって，モデルを考え，これを理解することがシミュレーションの第一歩となる．シミュレーションにとって重要なのは，対象のモデリングであり，正確なモデリングとは対象の理解そのものである．ところが，十分な時間を割いて対象を理解することなく，直ぐにコンピュータの前に座ろうとする学生が毎年見られる．当然のことながら，コンピュータの画面を前に途方にくれるばかりでプログラムのコードを書くどころではない．

そこでこのシミュレーションを試みるための基礎的な文献の案内から始めよう．スペースシャトルの中のペンチの運動を分かりやすく解説した本は

科学シュミレーション研究会: パソコンで見る複雑系 · カオス · 量子，BLUE BACKS B1160, 講談社 (1995)

である．Turbo C++ Version 4 でコンパイル可能なソースファイルと，MSDOS 環境下で実行可能な EXE ファイルも収録されている．我々はペンチの形状データ，ペンチの慣性モーメント等はこのソースファイルを参考にして決めた．ゴールドスタイン

ゴールドスタイン: 古典力学 (上)，吉岡書店 (1983)

は古典力学の定番教科書として著名で，第 4 章「剛体の運動」で無重力状態での剛体の運動方程式が導かれている．一方，コマの運動を丁寧にそしてやさしく解説し本は

安井久一: コマはなぜ倒れないか，共立出版株式会社 (1998)

が最良なテキストである．文献 [1] に収録されているソースファイルを理解するには，Euler-Olinde-Rodorigues のパラメタ，Cayley-Klein のパラメタを理解しておくほうがいい．これについては

山内恭彦: 回転群とその表現，岩波書店，(1957)

の第 3 章「回転群の表現」が詳しい．最後に本稿と直接関係しないが

牧野淳一郎: パソコン物理実地指導，共立出版株式会社 (1999)

は，物理系の振る舞いを数値的に調べるときに，どうやって求まった答えが「正しい」かどうかを判断するという議論，すなわちシミュレーション結果の検証にかなりのスペースを割いている．プログラムリストもついており，セミナーの教材として最適であると考え，採用した．

1.2 運動方程式とプログラミング言語の選択

モデルを記述するプログラミング言語をどうするか．これはモデル自体がある程度まで最適な言語を決定してしまう．今回は，常微分方程式で記述されるモデルであるから，使われている数学の抽象性を素直に表現できる言語が望ましい．どの程度の抽象能力が要求されるかを見てみよう．

力学のラグランジアン形式を用いて剛体の運動を記述するためには，剛体の方向を指定する 3 つの独立なパラメータが必要である．文献には，このようなパラメータの組がいくつも述べられているが，もっと普通に用いられ，しかも有用なのは Euler の角 α, β, γ である．Euler の角を用いた剛体の自由運動，すなわち無重力状態での回転運動の方程式を，以下に述べる [2].

$$\frac{d\alpha}{dt} = \frac{\sin\gamma}{\sin\beta}\omega_a + \frac{\cos\gamma}{\sin\beta}\omega_b \tag{1.1}$$

$$\frac{d\beta}{dt} = \omega_a \cos\gamma - \omega_b \sin\gamma \tag{1.2}$$

$$\frac{d\gamma}{dt} = -\frac{\cos\beta\sin\gamma}{\sin\beta}\omega_a - \frac{\cos\beta\sin\gamma}{\sin\beta}\omega_b + \omega_c \tag{1.3}$$

次に，静止直交座標系から回転座標系への変換行列が Euler の角でどうのように表されるか示しておこう．

$$\mathbf{A} = \begin{pmatrix} \cos\alpha\cos\gamma - \sin\alpha\cos\beta\sin\gamma & \sin\alpha\cos\gamma + \cos\alpha\cos\beta\sin\gamma & \sin\beta\sin\gamma \\ -\cos\alpha\sin\gamma - \sin\alpha\cos\beta\cos\gamma & -\sin\alpha\sin\gamma + \cos\alpha\cos\beta\cos\gamma & \sin\beta\cos\gamma \\ \sin\alpha\sin\beta & -\cos\alpha\sin\beta & \cos\beta \end{pmatrix}$$

主軸回りの慣性モーメントは

$$\begin{pmatrix} I_a \\ I_b \\ I_c \end{pmatrix} = \begin{pmatrix} 2(I + M\ell^2\sin^2\delta) \\ 2(J\sin^2\delta + K\cos^2\delta) \\ 2(J\cos^2\delta + (K + M\ell^2)\sin^2\delta) \end{pmatrix}$$

である．ペンチは剛体の二つのパーツからなり，これが要で固定されている．M は各パーツの質量，ℓ は各パーツの重心と要との距離である．また I, J, K は各パーツの慣性モーメントを表す．角度 δ はペンチの開き角の半分で 46° に固定してある．

角速度ベクトル $\boldsymbol{\omega}$ と角運動量ベクトル $\mathbf{L}$ との関係は $\mathbf{L} = \mathbf{I}\cdot\boldsymbol{\omega}$ である．ここで $\mathbf{I}$ は慣性テンソルである．したがって

$$\mathbf{L} = \begin{pmatrix} L_x \\ L_y \\ L_z \end{pmatrix}$$

とおけば，$\boldsymbol{\omega} = \mathbf{I}^{-1}\cdot\mathbf{A}\cdot\mathbf{L}$ と表される．ここで $\mathbf{L}$ は運動のあいだじゅう初期条件から決まる一定の値をもつ．

運動方程式をベクトル形式で表現するためにベクトルとその内積が必要である．勿論座標系の変換行列のためには行列とその逆行列が定義されなければならない．このためにはオブジェクト指向言語を使ったほうが，そうでない言語を使うよりずっと簡潔にプログラムができる．そこで C++ で書くことにした．特に C++の演算子の多重定義を使えばベクトルや，行列を表現しやすく，使いやすいものにすることができる．

第2章　プログラミングにおける問題点と対策

2.1　Vectorクラスと Matrixクラス

就職試験などのために卒業研究の時間が確保しにくいので，Matrix クラスは手元にあった本

Bruce Eckel: Using C++，Osborne McGraw-Hill(1989)

のソースコードを使うことにした．しかしながら，C++ は発展途上の言語であるからバージョンによる変化も激しく Bruce のテキストは今や古い感じを与える．GNU の gcc は 3.0 から ISO C++(文献 [7] に相当) に準拠したので，gcc でコンパイルできなければならない．しかし，Bruce のソースコードには Visual C++ v.6 でコンパイル可能であっても Linux の gcc v.3.0 以上でエラーになる箇所がある．そこで必要な箇所のみ採用し，C++ のバイブルである

Bjarne Stroustrup: プログラミング言語 C++ 第 3 版，アジソン・ウェスレイ (1998)

を参考にして書き換えた．

ベクトルは行列の特殊であるから継承関係にある．Vector クラスを Matrix クラスのサブクラスとして定義するのが数学から見ても素直な考え方である．しかし内積の定義に問題が生じる．内積は行列の積を使って ${}^t\mathbf{ab}$ で定義される．ここで ${}^t\mathbf{a}$ は $\mathbf{a}$ の転置行列である．しかし転置は C++ の演算子でないので，プログラムの中では内積を関数形で

```
const int VLEN = 3;
vector a, b;
double x;
x = transpose(a)* b
```

と書かざるをえない．ここで “*” は行列の積を表す．内積はしばしば現れる演算なので，もっと簡潔に表したい．そこで Vector クラスと Matrix クラスをおのおの独立なクラスとして定義した．行列の四則演算は Matrix クラスの定義の中で

```
matrix operator+(const matrix& rval);  // matrix addition
matrix operator-(const matrix& rval);  // matrix subtraction
matrix operator-();                    // unary minus
matrix operator*(const matrix& rval);  // matrix multiplication
matrix operator*(const double  rval);  // scalar multiplication

//引数の値に基づいて新しい値を生成するだけの演算子
matrix operator+(const matrix& a, const matrix& b);
```

```
    matrix operator-(const matrix& a, const matrix& b);
    matrix operator*(const matrix& a, double b);
    matrix operator*(double a, const matrix& b);
    matrix operator/(const matrix &a, double b);
    matrix operator/(double a, const matrix& b);
```

とプロトタイプ宣言してある．一方ベクトルの四則演算は Vector クラスの中で

```
// Unary -
    const vector operator-(){
        vector v;
        for(int i=0;i<VLEN;i++) v.element[i] = -element[i];
             return v;
    }
//Binary operator  +,-,*,/
    friend  const vector operator+(const  vector&, const vector&);
    friend  const vector operator-(const  vector&, const vector&);
    friend  vector operator*(double, const vector&);
    friend  double operator*(const  vector&, const vector&);
    friend  vector operator*(const matrix&, const vector&);
    friend  vector operator*(const  vector&, double);
    friend  vector operator/(const  vector&, double);

// Binary operator +=, -=, *=, /=
   vector& operator+=(const vector& b){
       for(int i=0;i<VLEN;i++)element[i] += b.element[i];
           return *this;
       }
   vector& operator-=(const vector& b){
       for(int i=0;i<VLEN;i++)element[i] -= b.element[i];
           return *this;
       }
   vector& operator*=(double b){
       for(int i=0;i<VLEN;i++)element[i] *= b;
           return *this;
       }
   vector& operator/=(double  b){
       for(int i=0;i<VLEN;i++)element[i] /= b;
           return *this;
```

```
}
```

と定義している．行列 A とベクトル b の積のプロトタイプ宣言は

```
friend double operator*(const matrix&  A, const vector& b);
```

である，これにより，ベクトルの内積と行列の積に同じシンボル”*” が使え，またベクトルと行列の積も A*b と表現できる．

勿論演算子の多重定義を用いることはいいことばかりでなく，一時オブジェクトの生成と消滅にコストが掛かり，効率を損ねる．今回は処理能力はそれほど要求されないと考え，一時オブジェクトは問題にならないと判断した．

2.2 4次の Runge-Kutta 法

Vctor クラスと Matrix クラスを導入したことにより，連立常微分方程式

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}) \tag{2.1}$$

$$\mathbf{x}(0) = \mathbf{x}_0 \tag{2.2}$$

を解く4次の Runge-Kutta 法を次のように簡潔に表すことができる．

```
typedef const vector (vfunc)(const vector&);


// the fourth order Runge-Kutta method
void rk4(vector& x, vector& w, vfunc f, double h)
{
    static  vector kx1, kx2, kx3, kx4;


    kx1 = f(x)*h;
    kx2 = f(x+kx1/2)*h;
    kx3 = f(x+kx2/2)*h;
    kx4 = f(x+kx3)*h;
    x += (kx1+2*(kx2+kx3)+kx4)/6;
}
```

一方，剛体の無重力状態での運動方程式は右辺に角速度ベクトル $\boldsymbol{\omega} = {}^t(\omega_a, \omega_b, \omega_c)$ を含んでいる．すなわち，運動方程式の右辺は C++で

```
// system equations
 const vector dxdt(const  vector&  x, const vector& omega)
{
    vector d;
    d[0] = sin(x[2])/sin(x[1])*omega[0] + cos(x[2])/sin(x[1])*omega[1];
```

```
    d[1] = cos(x[2])*omega[0] -sin(x[2])*omega[1];
    d[2] = -cos(x[1])*sin(x[2])/sin(x[1])*omega[0] - cos(x[1])*cos(x[2])/sin(x[1])*omega[1] +
         omega[2];
    return d;
}
```

と表されるので Runge–Kutta 法も次のように修正されなければならない.

```
typedef const vector (vfunc)(const vector&, const vector&);

// the fourth order Runge-Kutta method
void rk4(vector& x, vector& w, vfunc f, double h)
{
    static   vector kx1, kx2, kx3, kx4;

    kx1 = f(x, w)*h;
    w= omega(x+kx1/2);
    kx2 = f(x+kx1/2,w)*h;
    w=omega(x+kx2/2);
    kx3 = f(x+kx2/2,w)*h;
    w=omega(x+kx3);
    kx4 = f(x+kx3,w)*h;
    x += (kx1+2*(kx2+kx3)+kx4)/6;
}
```

Runge–Kutta 法が実現されれば数値シュミレーションに取り掛かれる.

2.3 gnuplot，MATLABを使ったグラフの作成

プログラミングの方法として，可能な限り単純な仕様を満たすものをまず書くべきである．今回は数値シミュレーションの正当性をまずチェックすることにした．文献 [1] の結果を参考にした．特に回転エネルギーが時間に関して不変であることを確かめた．一般論として，実時間アニメーションを行なうための複雑な GUI(Grahical user interface) は最初は避けるのがよい．とりあえずシミュレーションの計算結果をちょっと見てみよう，というとき，グラフ作成ツール gnuplot を使うのが良い考えである．我々は MATALAB が使える環境にあるので MATLAB を使った．次の図 2.1∼2.4 は計算結果を MATLAB でグラフ化し PostScript(EPS) ファイルに変換したものである.

2.4 pgplotでグラフを書く

数値シミュレーションをおこないながら実時間シュミレーションをしたい場合は，グラフ作成ツールでは無理がある．C++ から呼び出せる，デバイスに依存しないグラフィックスパッケージ

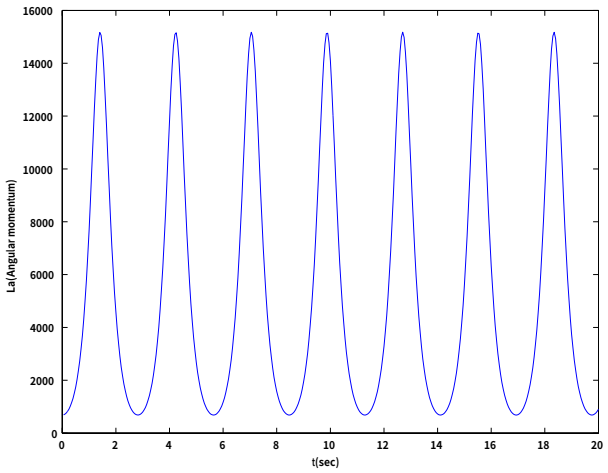

図 2.1: 角運動量ベクトルの成分 La の符号は変化しない

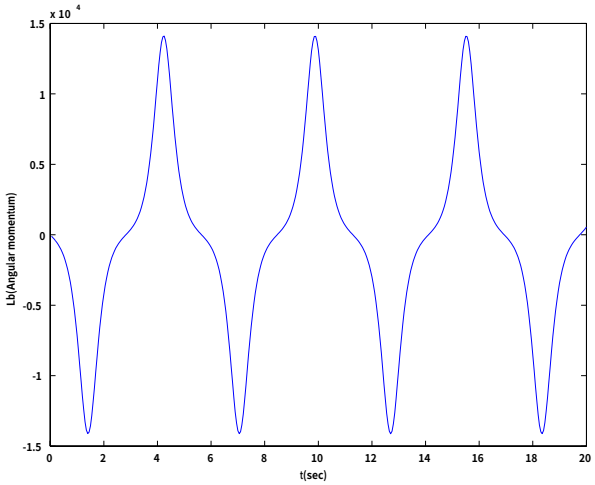

図 2.2: 角運動量ベクトルの成分 Lb の符号は変わる.

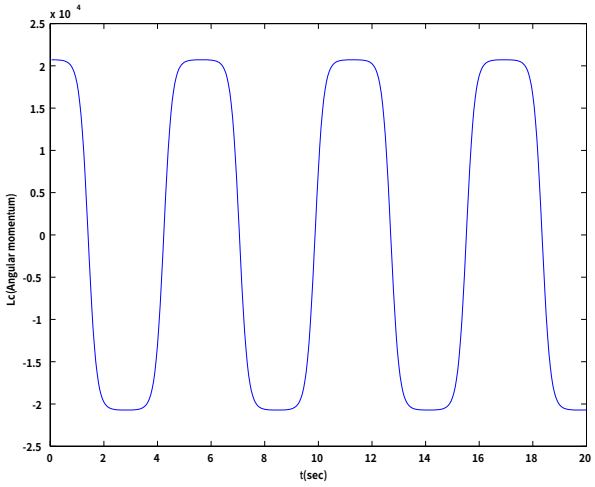

図 2.3: 角運動量ベクトルの成分 Lc の時間変化

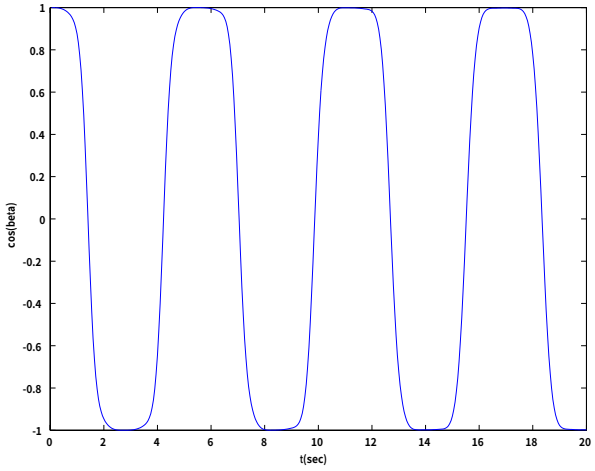

図 2.4: c 軸が z 軸となす角の余弦の時間変化. -1 から 1 までで変化しているからペンチが上下反転している.

ライブラリが必要になってくる．プログラムを Linux で書くことが多いので，Linux 環境との親和性がよく Windows 2000 でも使えるものが望ましい．Visual Basic は Windows の特徴である GUI を極めて簡単に作ることができるが，Mastrix クラスや Vector クラスの構築には不向きである．また我々にとって Linux 環境で使えないのは問題点の一つである．研究室の卒業研究では，これまで，ユーザーインターフェイスを Visual Basic で作り，VC++ でアルゴリズムを実現した DLL(Dynamic Link Library) を Visual Basic 側からコールする方法をとってきた．今回はプログラムから直接図を作ることにしよう．そこで，Pgplot Graphics subroutine library を使うことに決めた．Pgplot はもともと Fortran 用のサブルーチン群で，C++ から使うこともできます．Pgplot を用いたプログラムを Linux 上で開発し，同じソースファイルを Cross Compile 環境でコンパイルすれば Windows 2000 で走る EXE ファイルを作れるので一挙両得である．Pgplot は比較的使いやすく，最初に作成した数値シュミレーションプログラムを修正するだけよい．図 2.1 と同じグラフを出力する main() 関数を示します．cpg で始まる関数が Pgplot のサブルーチンである．このプログラムはひとつの画面に一つの図を出力するものであるが，複数の図を入れることも容易にできる．

```
int main(int argc, char* argv[]){

vector x; // Eulerian angles x=(alpha, beta ,gamma)
vector w; // 回転系角速度
vector rL; // 回転系角運動量

double t,h;
int nsteps, print_interval;

std::ifstream fin(infile, std::ios::in);
if (!fin.is_open())
    error(argv[0], infile , ":Cannot open file for reading");

    // 回転軸の天頂角の初期値 p(degree)，回転軸の方位角の初期値 q(degree)，回転周期の初期値 T(second)
    // 天頂角の初期値のずれ u(degree)
fin  >> nsteps >> print_interval >> T >> p >> q >> u;

// ........ Initialization ............................
initial_condition(x,w);

h = T/200.0;

cout.precision(16);
        std::ofstream fout(outfile, std::ios::out);
```

```
if(!fout.is_open())
    error(argv[0], outfile , ":Cannot open file for writing");
fout.precision(16);
fout.setf(std::ios_base::fixed, std::ios_base::floatfield);
// graphics
double told=0.0,yold=0.0;
char *devname="?";
if (argc == 2)
    devname =  argv[1];
if (cpgopen(devname) != 1)   // open a graphics device
    exit(1);
 // change the size of the view surface
cpgpap(6.0,1);
     // set window and viewpoint and draw labeled frame
double tmin=0.0,tmax=20.0,ymin=0.0,ymax=16000.0;
cpgenv(tmin,tmax,ymin,ymax,0,0);
cpglab("t(sec)","Lx(angular momentum)"," ");// write labels
x-axis,y-axis and top of plot

for(int i=0; i < nsteps; i++){
    t= h*(i+1)
    rk4(x,w,dxdt,h);
    w=omega(x);    // w の再計算
    rL= A*L;
    double E=(I[0]*w[0]*w[0]+I[1]*w[1]*w[1]+I[2]*w[2]*w[2])/2.0; // 回転エネルギー

    if (((i+1) % print_interval ) == 0){
         //fout << t << "  " << rL[0] << " " << rL[1]<<  " " << rL[2] << " " <<
         //E << "  " << cos(x[1]) << "\n";
            cpgsci(2);                  // set color index
            cpgmove(told,yold);         // move pen
             // draw a line from the current position to a point
            cpgdraw(t,rL[0]);
            told=t;
            yold=rL[0];
    }
}
cpgclos(); close a selected graphics device
return 0;
```

}

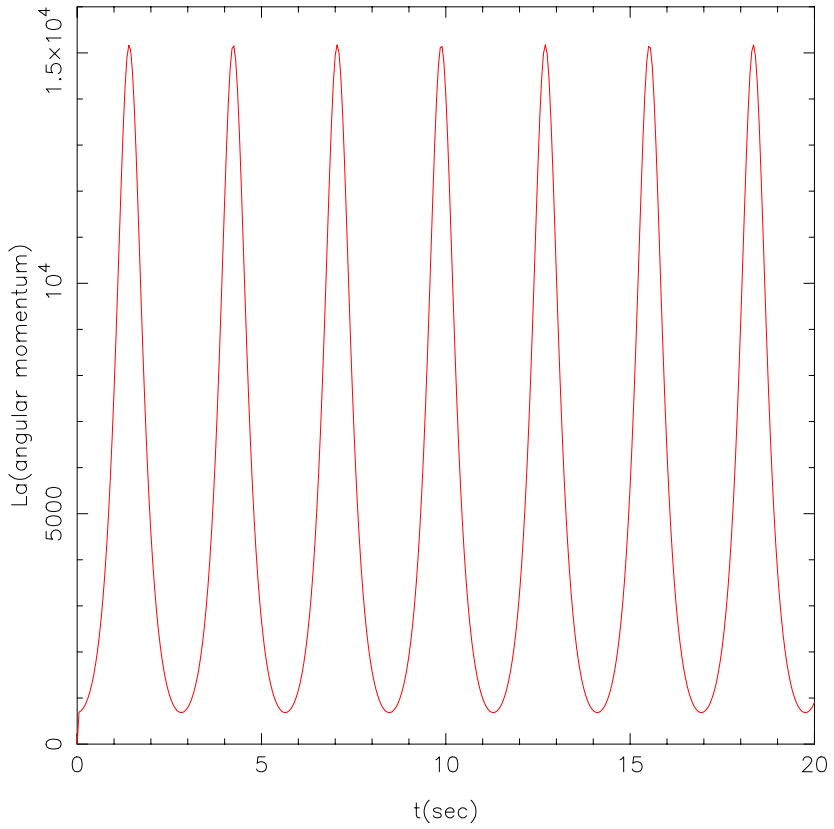

図 2.5: 角運動量ベクトルの成分 La の符号は変化しない

2.5 剛体の回転運動の可視化

ペンチが回転しながら上下反転することは分かった．今度は，シュミレーションを始めて，ペンチがどんな風に回転しながら傾いて，上下反転してゆくかを画面上で立体的に見ることができなければならない．これには多くの選択肢がある．リアルタイム 3D プログラミングで C++言語を使いたければ MFC(Microsoft Fundation Class) と OpenGL の組み合わせが考えられる．しかし，この場合，学習曲線が急峻で卒業研究としては難しい．一方，Visual Basic と OpenGL の組み合わせは

酒井幸市: OpenGL 3D プログラミング，CQ 出版社 (2000)

で詳しく解説されている．Visual C++ で作成した DLL を Visual Basic でコールする形のプログラムになる．この方法がより分かりやすい，と思う．しかし，我々の希望は Linux と Windows 2000 で相互に移植性の高いプログラムである．Linux 環境でアプリケーションを開発し，同じソースファイルをクロスコンパイルするだけで Windows 2000 でも走るプログラムを作りたい．この場合，選択肢は VTK(Visualization Toolkit) と FOX(free objects for X) の二つがある．VTK にはテキスト [9] がある．アニメーションはあまり得意でないようだ．ローレベル API として OpenGL をサポートしているので，手軽に OpenGL プログラムが書けるという．一方，FOX には残念ながら

よいドキュメントがない． FOX ライブラリーを使ってかかれたプログラムはコンパイルするだけでさまざまなプラットホームで動作する (Write once, Compile anywhere)．したがって，Linux と Windows 2000 のクロスプラットホーム開発が可能である．また，FOX Library は OpenGL と Mesa に対する Widget を提供しているので 3D アプリケーションをかくのに便利である．

次の問題は，3D オブジェクトのモデルの制作である．物体 (ペンチ) をコンピュータ内部に数値化した幾何情報 (形状モデル) として定義しなければならない．いくつかの可能性が考えられる．

1. 直方体や円柱，球などの単純なプリミティブを組み合わせて物体を作成する．多くのテキストの例題はこの方式である． しかしある程度複雑になると手に負えなくなるのは明らかである．3D オブジェクトをテキストエディターで編集するのは原理的に無理がある．

2. ファイルから幾何学データを読み込んで 3D オブジェクトを作成する．この方式を採用しているテキストもある．しかし，ファイルに書き込まれたデータをどのように作成したかが問題である．そもそも OpenGL は DirectX と違って，決まったファイルフォーマットが無い．
   これによりプログラム側でデータ形式や描画ルーチンを自由に設計できる．その代償として汎用性は失われる． この欠点は避けられない．

3. ある程度複雑な 3D オブジェクトは対話型モデリングツールを利用して作成する．これが王道である． OpenGL のソースコードを吐き出すモデリングツールはあるのか？

   3D 図形の表現に広く使われている CAD 系の標準フォーマット DXF 形式をほとんどのモデリングツールがサポートしているので，テキストファイルである DXF ファイルを OpenGL のファイルに変換するプログラムを書けば目的は達成される．しかし DXF 形式にもバージョンの違いによる互換性の問題があり，目的に応じて仕様を決めなければならない．いずれにしてもコンバータが必要である．

4. 新藤・安部 [10] は PiasGL4 という 3DCG エンジンを開発している．PiasGL4 は高水準ライブラリで，OpenGL の上位に位置している．またモデリングツール PiasArtist2000 もこのエンジンと同等のものを内蔵して動作している． 複雑な形状のモデリングは PiasAtist2000 で作成したデータファイルをプログラムに読み込んでアニメーションを動かしている．Windows2000 環境で動作するので，マルチプラットホーム環境を指向する我々の目的に沿わない点もあるが，今のところ最も優れた現実的な解決策を提供している．

図 2.6 は PiasArtist2000 で作成したペンチの 3D オブジェクトである．ペンチはそれほど複雑でないが，この程度の複雑な物体でもモデリングツールなしでは作成は難しい．PiasArtist2000 は市販の CAD ソフトに比べると機能が落ちるとはいえ，OpenGL でプログラミングする際のツールとしては十分な機能を持っている．

2.6 PiasArtist2000で作成した形状情報の読み込みとシミュレーション

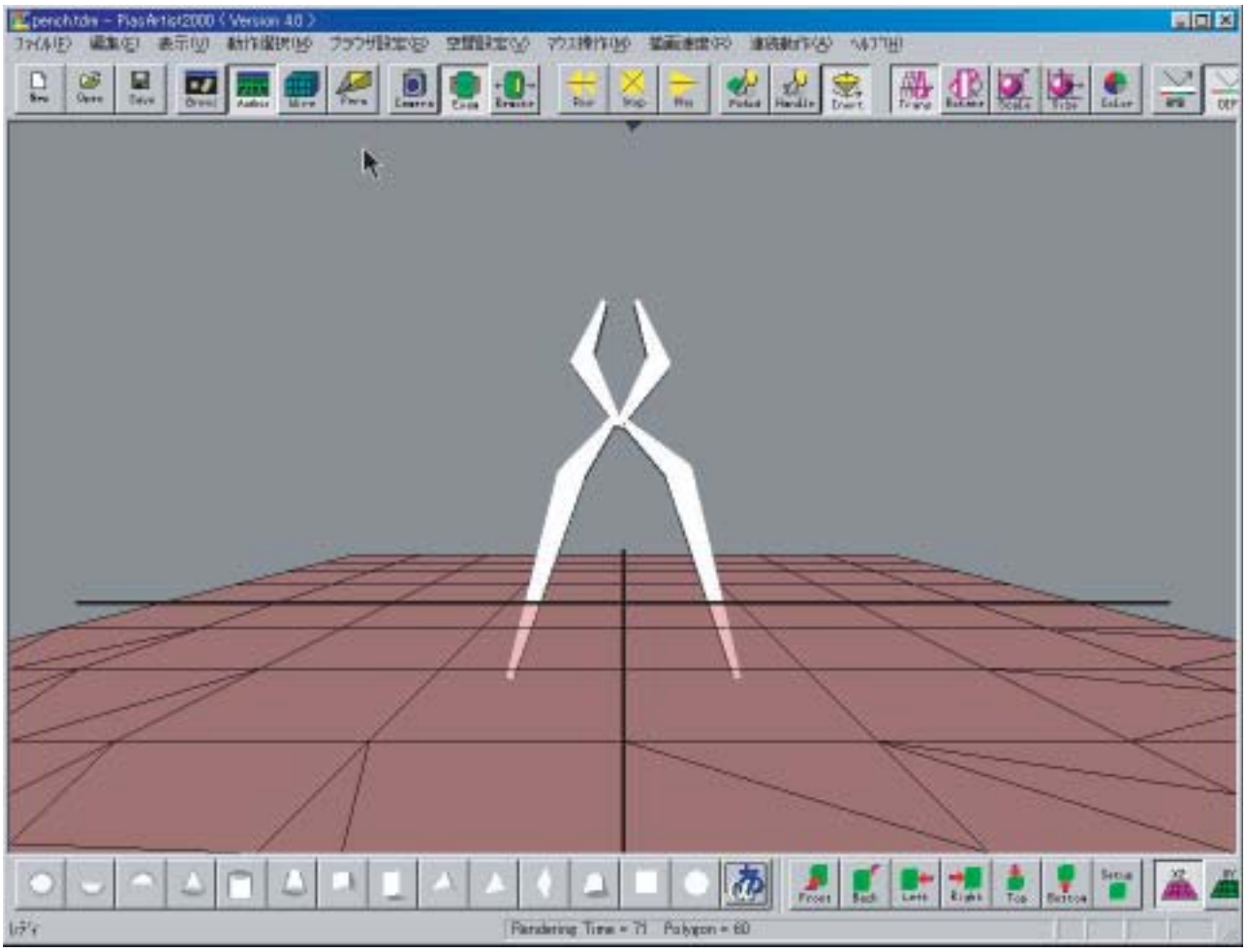

図 2.6: PiasArtist2000 で作成したベンチ

関連図書

[1] 科学シュミレーション研究会: パソコンで見る複雑系・カオス・量子，BLUE BACKS　B1160, 講談社 (1995)

[2] ゴールド スタイン: 古典力学 (上)，吉岡書店 (1983).

[3] 安井久一: コマはなぜ倒れないか，共立出版株式会社 (1998).

[4] 山内恭彦: 回転群とその表現，岩波書店 (1957).

[5] 牧野淳一郎: パソコン物理実地指導，共立出版株式会社 (1999).

[6] Bruce Eckel: Using C++， Osborne McGraw-Hil l(1989).

[7] Bjarne Stroustrup: プログラミング言語 C++ 第 3 版，アジソン・ウェスレイ (1998).

[8] 酒井幸市: OpenGL 3D プログラミング，CQ 出版社 (2000).

[9] W.Schroeder, K.Martin, and B.Lorensen: The Visualization Toolkit; An Object-Oriented Approch to 3D Graphics, Prentice–Hall(1998).

[10] 新藤義昭，阿部正平: OpenGL リアルタイム 3D プログラミング，秀和システム (2000).