内積と外積を計算する関数

01ca0125 鈴木 藍
2002 年 5 月 16 日

目 次

概 要

内積と外積の概念について調べ、これを計算するプログラムを作成する。このプログラムは、gnuplot で可視化をおこなった。また、内積と外積をどのように応用するかを調べた。

レポートの目的

内積と外積について 不明な点を調べる。とても有用な道具である事がわかっているので、どこでこれらを使うべきかを 3DCG の分野から調べた。

1 言葉の定義

1.1 線形とは

線形なものは、線形な関数で表す事ができる。線形な関数とは、直線で表す事が出来る関数の事である。また、線形な関係が成り立つような要素の集まりを、線形空間という。線形空間とは、実数と対応できる軸を持つ集合である。たとえば、3 次元空間や時間、複素数や 三角関数なども含まれる。もっとも典型的な線形空間は 1 次元の実数である。また、複素平面も線形であるが、通常の演算規則が実数とは異なる。ベクトル空間も線形である。しかし、対象によっては何次元でも考える事が出来る。三角関数と多項式については、無限次元の線形空間となる。

1.2 線形とベクトル

ベクトルとは、空間の中で方向を持った量の事である。たとえば速度などは、方向を考えなければならないのでベクトルを使って考える事が出来る。ベクトルは線形な空間の要素である。

1.3 内積とは

内積 (dot product) とは、二つのベクトルの対応する要素をかけて 総和をとったものであり、計算の結果は一つの値となる。式にすると

$$A \cdot B = (a_x b_x + a_y b_y + a_z b_z)$$

となる。内積は 以下の式が成り立つ。

$$\vec{a} \cdot \vec{b} = |a||b| \cos\theta = a_x b_x + a_y b_y + a_z b_z$$

また、二つのベクトルのなす角を求める事が出来る。

1.4 外積とは

内積は一つの値となるが、外積は ベクトルになる。外積 (cross product) とは $a \times b$ と書き二つのベクトルの積の形である。外積 $a \times b$ は二つのベクトル の作る 平行四辺形の面積の大きさであり、かつ 二つのベクトルのどちらともに垂直である。この外積を使って、二つのベクトルがどの

面に向かっているかなどを調べることが出来る。また、外積は 3 次元空間のなかで定義されるものである。式は

$$\vec{a} \times \vec{b} = (a_y b_z - a_z b_y)e1, (a_z b_x - a_x b_z)e2, (a_x b_y - a_y b_x)e3 = |a||b| \sin\theta \cdot c$$

ここで、$e1$ は x の単位ベクトル $(1, 0, 0)$、$e2$ は y の単位ベクトル $(0, 1, 0)$、$e3$ は z の単位ベクトル $(0, 0, 1)$、 c は z の単位ベクトルである。

2　課題の内容

ベクトルの内積、外積を計算するプログラムを作成する。要素は 3 つと限定する。

2.1　作成した関数の概要

以下がヘッダファイル 'product.h' の内容である。定義されている関数のプロトタイプ宣言はここで行っている。

```
#define PI     3.1415
typedef struct component VectorComponent;
typedef struct operator  MessageSelector;
typedef double           quantity;

struct component {
   quantity  x, y, z;
  /* the place where this is now. */
   double    thisX, thisY, thisZ;
};
struct operator {
   quantity         (*dotProduct)(VectorComponent *A, VectorComponent *B);
   VectorComponent *(*crossProduct)(VectorComponent *A, VectorComponent *B);
   VectorComponent *(*crossProductWithAngle)(VectorComponent *A,
                                        VectorComponent *B, double theta);

   VectorComponent *(*newVectorComponent)(void);
   VectorComponent *(*newUnitVectorComponent)(void);
};
quantity          lambda1(VectorComponent *A, VectorComponent *B);
VectorComponent  *lambda2(VectorComponent *A, VectorComponent *B);
VectorComponent  *lambda3(VectorComponent *A, VectorComponent *B,
                         double theta);
VectorComponent  *lambda4(void);
VectorComponent  *lambda5(void);
MessageSelector  *newMessageSelector(void);
```

構造体 *component* は ベクトルを表している。また、構造体 *operator* はベクトルを操作する関数をひとまとまりに扱うための物である。

2.1.1 内積を求める関数 $dotProduct$

内積を求める為に、ベクトルの成分を掛け合わせ、総和を取る方法を使った。これは Lisp で実装したことがある。$dotProduct$ と言う名前は、構造体 $component$ 内に記述されている。

```
quantity  lambda1(VectorComponent *A, VectorComponent *B)
{
   return ((A->x * B->x) + (A->y * B->y));
}
```

2.1.2 外積を求める関数 $crossProduct$

外積を求める為に、

$$\vec{a} \times \vec{b} = (a_y b_z - a_z b_y)e1, (a_z b_x - a_x b_z)e2, (a_x b_y - a_y b_x)e3$$

という式を使った。

```
VectorComponent *lambda2(VectorComponent *A, VectorComponent *B)
{
  VectorComponent *com;
  MessageSelector *message;

  message  = newMessageSelector();
  com      = message->newVectorComponent();

  com->x = ((A->y * B->z) - (A->z * B->y));
  com->y = ((A->z * B->x) - (A->x * B->z));
  com->z = ((A->x * B->y) - (A->y * B->x));

  free(message);
  return com;
}
```

$|a||b| \sin\theta \cdot c$ という式も試してみたが、うまくいかなかった。調査中である。

2.1.3 その他の関数

その他のツールとして作成した関数は割愛する。

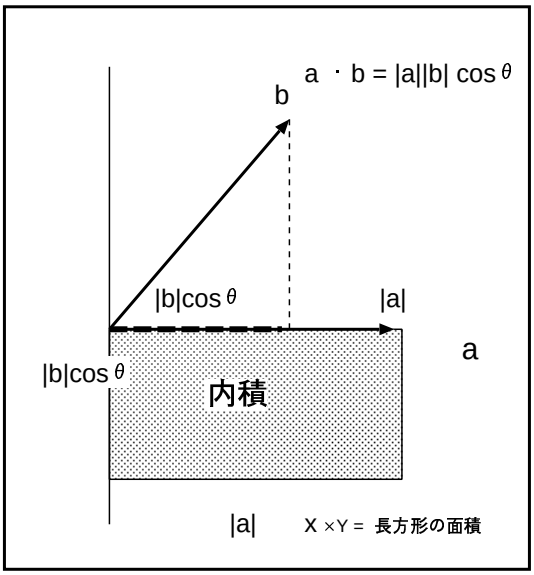
3 関数の実行

3.1 方針

どちらも、ただ値を表示してもなにもわからないので gnuplot で、その実行結果の値が何を表すのかを可視化してみた。 また、アニメーションとして表示する事で値が変化した時の状態を把握できるようにした。

3.2 $dotProduct$ 関数

表示しているものは、二つのベクトルがなす角が変化する時の長方形の面積である。 内積は、ある量を表しているので私にとっては 面積という見える形で表されると分かりやすい。内積の図形的な解釈は以下の通りである。



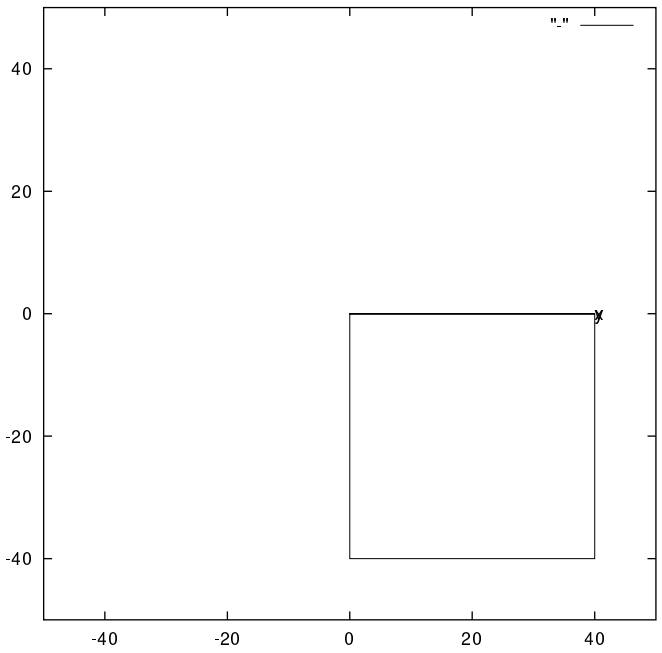
main 関数内で、以下のように $dotProduct$ を実行している。

```
dotproduct = message->dotProduct(A, B);
```

データとして、

$$\vec{A} = (40.0, 0.0, 0.0), \qquad \vec{B} = (0.0, 0.0, 0.0)$$

とした。ここで ベクトル B は移動してゆくベクトルである。また、ベクトル B の量として double 型の変数 に 40.0 が格納されている。表示は、ベクトル A との 角度を 微小にふやしていった時の ベクトル A, ベクトル B, 内積を表示している。



これは静止画像であるが、実行すると y は $0.0 < \theta < 1.0$ 分 移動する。

3.3 *crossProduct* 関数

外積を、ベクトル A とベクトル B が作る平行四辺形として表現した。main 関数では 以下のように *crossProduct* を実行している。

```
C = message->crossProduct(A, B);
```

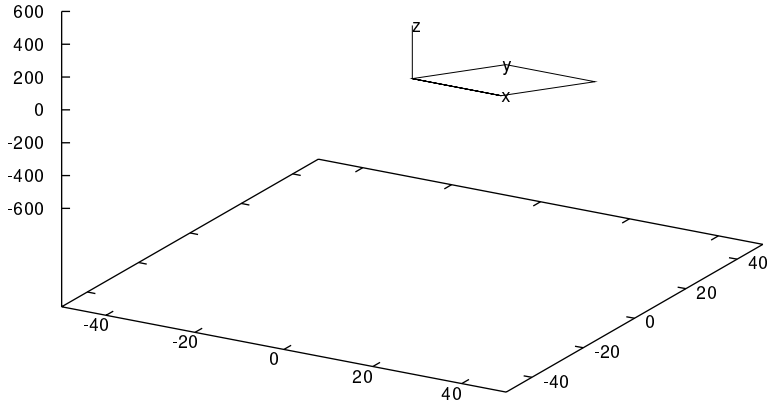
ここでは

$$\vec{A} = (20.0, 0.0, 0.0), \quad \vec{B} = (0.0, 0.0, 0.0), \quad \vec{C} = (0.0, 0.0, 1.0)$$

C は単位ベクトルである。

表示をしているのは、ベクトル A と ベクトル B の $\vec{A} \times \vec{B}$ の間で、 ベクトル B が z 軸を中心として回した時の z 軸の長さである。

"-" ——



y は 360 度 z 軸を中心に回転するが、180 度をすぎた時点で 外積の符号が反転する。

4 考察

4.1 外積と内積は何に使うか

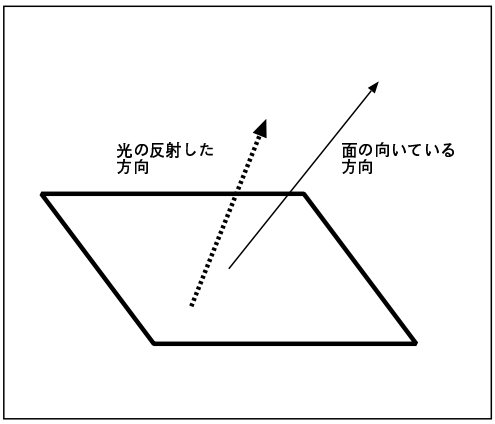
外積と内積の調査をしてゆくと、コンピュータグラフィクスの世界では欠かす事の出来ない概念である事が分かってきた。 ある方向から何かの量を取り出す事が出来る、とても便利なものだそうである。いろいろなキーワードも出てきた。 クォータニオンというものも、とても便利な道具として使われていると書かれていた。 極端なところでは、Java でクォータニオンの 4 つ程のメソッドがあれば、ほとんどの 3DCG が書けるそうである。こういった内容は とても興味はあるが、時間の都合上今回は内積が何につかわれているか、外積が何に使われているのかを、コンピュータグラフィクスの資料から調べ、考察した。

4.2 内積は何に使うか

内積とは、計算すると一つの値になる式が定義されている。このなかで

$$\vec{a} \cdot \vec{b} = |a||b| \cos\theta$$

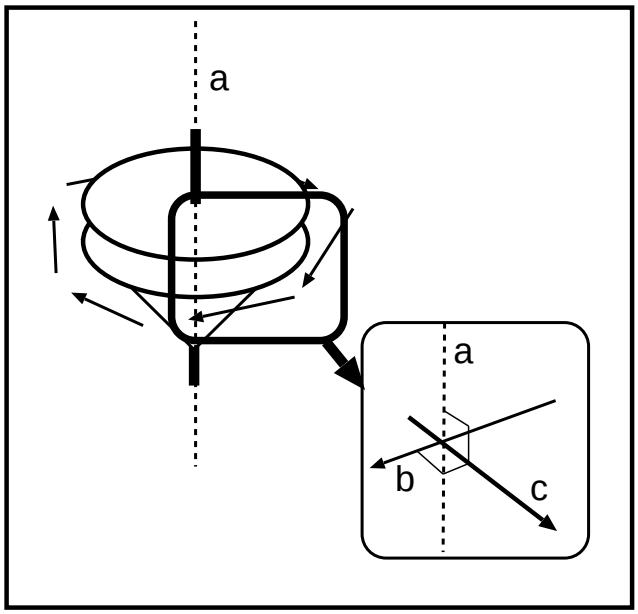
に関しては、 cos が入っているので、二つのベクトルのなす角度を求める事が出来る。 さらに具体的なところでは、内積を使えばある面の光のあたり具合などが分かるそうである。想像ではあるが、このような感じではないかと思った。



もし光が面の向いている方向に対して並行に反射すれば内積の定義から、内積の値は最大になるので光はおおく反射する。その他の用途についてはよくわからなかった。

4.3 外積は何に使うか

外積の値はベクトルである。 これは 二つのベクトルに直交するベクトルである。よって、ふたつのベクトルを含む面に対して垂直であるから、面の向きを計算する事が出来る。と、書いてあったのだが、これがどのように便利かはよくわからなかった。しかし、物理の方面で調べてみると回転に関してとても重要な概念である事がわかった。ある文献で コマに関して簡単にふれているものがあったのでこれについて考察してみたい。
まず、コマがあったとしてそれを回転させる。 そして、コマの軸を真上から押してみる。そうするとコマは押した方向には傾かず、横に傾くそうである。そして、コマは止まるまで倒す事は出来ない。ここで、回転軸を a , 回転する力の方向を b とすると、真上から軸を押したときに傾く横の方向が a と b に直交する外積である。これを図で書いてみた。



実際には b の力の方向はこれほどおおざっぱではないのでおそらくここで 微分をするのだと思う。

4.4　行列はどう使うか

行列は 座標の変換に使われていた。先人がもう、変換の為の行列を作ってくれているそうなので私もこれらを ライブラリとして作成したい。また、座標系という言葉が出てきたが ベクトルを移動する方向によって座標系の名前がついているそうである。 先日 シミュレーションの時間に教わった 外積の座標系は 右手座標系と左手座標系といわれている。この座標系をもとに 座標を変換する。カメラ座標系と言うものもあるそうである。

まとめ・結論

外積と内積は、やはり何かの量であり 可視化は可能であった。とりあえず おぼえることが出来ないので、このように何らかの形 (インスタンス) として出てくれば 応用の幅が広がるはずである。

感想

最終的に 外積は 藤本先生から以前聞いた 「コマ」の話にいきついてしまった。偶然ではあったが、以前 聞く事と 現在聞く事では少し 捉え方も違い、絵が見えやすかったのでよかったと思う。また、時間が足らなくて 行列の 座標変換のライブラリが作れなかった事が残念である。

参考文献

[1] 3D_programing
http://homepage1.nifty.com/mimi_kaki/D3DRM/title.html
2002 年 5 月 15 日 参照

[2] 代数
http://www.ee.seikei.ac.jp/user/seiichi/lecture/MathColum/ColumB/b1/b1.html
2002 年 5 月 15 日 参照

[3] 3D 補講
http://www.cc.rim.or.jp/ devilman/3dCoding/3dCoding.html
2002 年 5 月 15 日 参照

[4] ベクトルテクニックノート
http://www.yamagame.com/MyWeb/Heart/heart13.html
2002 年 5 月 15 日 参照

[5] 3D 幾何学
http://hp.vector.co.jp/authors/VA013845/algorithm/vector.html
2002 年 5 月 15 日 参照

[6] 外積
http://www.hat.hi-ho.ne.jp/morishima/sub208.htm
2002 年 5 月 15 日 参照