三次元幾何要素表現と交差交線計算について

渡辺大地

1　空間上の直線・線分の幾何表現

相異なる 2 点 A,B に対し、直線 AB 上の任意の点 P は、次のようにして表すことができる。(t は実数)

$$\begin{aligned}\mathbf{P} &= \mathbf{A} + t(\mathbf{B} - \mathbf{A}) \\ &= (1-t)\mathbf{A} + t\mathbf{B}\end{aligned}$$

また、t と線分 AB の位置関係は以下のようになる。

$$\begin{cases} t < 0 & \text{線分 AB の外 (A 側)} \\ t = 0 & \text{点 A} \\ 0 < t < 1 & \text{線分 AB 上} \\ t = 1 & \text{点 B} \\ t > 1 & \text{線分 AB の外 (B 側)} \end{cases}$$

2　空間上の平面の幾何表現

平面 α 上の同一直線上にない三点 A,B,C に対し、α 上の任意の点 P は、次のようにして表すことができる。(t, s は実数)

$$\mathbf{P} = \mathbf{A} + u\mathbf{L} + v\mathbf{M} \qquad (\text{ただし、}\ \mathbf{L} = \mathbf{B} - \mathbf{A},\quad \mathbf{M} = \mathbf{C} - \mathbf{A})$$

このとき、各幾何要素は以下のような条件を満たす P の集合として考えることができる。

点 A	$\Longrightarrow$	$u = 0, v = 0$
点 B	$\Longrightarrow$	$u = 1, v = 0$
点 C	$\Longrightarrow$	$u = 0, v = 1$
直線 AB 上	$\Longrightarrow$	$v = 0$
直線 AC 上	$\Longrightarrow$	$u = 0$
三角形 ABC の内部	$\Longrightarrow$	$u > 0, v > 0, u + v < 1$
$\overrightarrow{\mathrm{AB}}, \overrightarrow{\mathrm{AC}}$ による平行四辺形の内側	$\Longrightarrow$	$0 < u < 1, 0 < v < 1$

3　二直線の交点

三次元空間内で直線 AB と直線 CD が点 P で交わると仮定する。このとき、直線 AB, 直線 CD はそれぞれ

$$\begin{cases} (1-s)\mathbf{A} + s\mathbf{B} \\ (1-t)\mathbf{C} + t\mathbf{D} \end{cases}$$

で表される。これが点 P 上では両方の式を満たすので、

$$(1-s)\mathbf{A} + s\mathbf{B} = (1-t)\mathbf{C} + t\mathbf{D}$$
$$\therefore \quad s(\mathbf{B}-\mathbf{A}) + t(\mathbf{C}-\mathbf{D}) = \mathbf{C}-\mathbf{A}$$

これを各座標要素に分解すると、

$$\begin{cases} s(\mathrm{B}_x - \mathrm{A}_x) + t(\mathrm{C}_x - \mathrm{D}_x) = \mathrm{C}_x - \mathrm{A}_x \\ s(\mathrm{B}_y - \mathrm{A}_y) + t(\mathrm{C}_y - \mathrm{D}_y) = \mathrm{C}_y - \mathrm{A}_y \\ s(\mathrm{B}_z - \mathrm{A}_z) + t(\mathrm{C}_z - \mathrm{D}_z) = \mathrm{C}_z - \mathrm{A}_z \end{cases}$$

となる。これをさらに行列式で表すと、

$$\begin{pmatrix} \mathrm{B}_x - \mathrm{A}_x & \mathrm{C}_x - \mathrm{D}_x & 0 & 0 \\ \mathrm{B}_y - \mathrm{A}_y & \mathrm{C}_y - \mathrm{D}_y & 0 & 0 \\ \mathrm{B}_z - \mathrm{A}_z & \mathrm{C}_z - \mathrm{D}_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s \\ t \\ p \\ 1 \end{pmatrix} = \begin{pmatrix} \mathrm{C}_x - \mathrm{A}_x \\ \mathrm{C}_y - \mathrm{A}_y \\ \mathrm{C}_z - \mathrm{A}_z \\ 1 \end{pmatrix}$$

である。このとき、行列が正則でない場合は二直線が平行である。正則行列であれば、逆行列を求めて s, t を算出する。二直線が交点を持つかどうかが確定しない場合は、計算結果で算出される p が 0 であるかどうかを確かめる必要がある。

4 平面と点の間の距離

対象とする平面 α の方程式を平面上の三点 A,B,C 及び実数 u, v を用いて以下のように表す。

$$\mathbf{A} + u\mathbf{L} + v\mathbf{M} \qquad (\text{ただし、}\ \mathbf{L} = \mathbf{B} - \mathbf{A},\quad \mathbf{M} = \mathbf{C} - \mathbf{A})$$

また、対象となる点を V とする。

まず、平面の正規化法線ベクトル $\mathbf{N}$ を求める。$\mathbf{N}$ は以下の式で求められる。

$$\mathbf{N} = \frac{\mathbf{L} \times \mathbf{M}}{|\mathbf{L} \times \mathbf{M}|}$$

ここで、$\times$ はベクトルの外積演算を表す。あとは、以下の公式を用いることで平面 α と点 V の距離 d を求めることができる。

$$d = |(\mathbf{V} - \mathbf{A}) \cdot \mathbf{N}|$$

5 平面と直線の交点

直線式と平面式を以下のようにおく。

直線式:

直線上の二点 P,Q に対し、実数 t を用いて

$$(1-t)\mathbf{P} + t\mathbf{Q}$$

平面式:

平面上の三点 A,B,C に対し、実数 u, v を用いて

$$\mathbf{A} + u\mathbf{L} + v\mathbf{M} \qquad (\text{ただし、}\ \mathbf{L} = \mathbf{B} - \mathbf{A},\quad \mathbf{M} = \mathbf{C} - \mathbf{A})$$

5.1 連立方程式による解法

直線と平面のそれぞれの式からベクトル方程式を導くと、

$$\begin{aligned} & (1-t)\mathbf{P} + t\mathbf{Q} = \mathbf{A} + u\mathbf{L} + v\mathbf{M} \\ \therefore \quad & u\mathbf{L} + v\mathbf{M} + t(\mathbf{P} - \mathbf{Q}) = \mathbf{P} - \mathbf{A} \end{aligned}$$

これを行列式で表すと、

$$\begin{pmatrix} l_x & m_x & \mathrm{P}_x - \mathrm{Q}_x & 0 \\ l_y & m_y & \mathrm{P}_y - \mathrm{Q}_y & 0 \\ l_z & m_z & \mathrm{P}_z - \mathrm{Q}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ t \\ 1 \end{pmatrix} = \begin{pmatrix} \mathrm{P}_x - \mathrm{A}_x \\ \mathrm{P}_y - \mathrm{A}_y \\ \mathrm{P}_z - \mathrm{A}_z \\ 1 \end{pmatrix}$$

となる。この行列が正則でない場合、平面と直線は平行であることになる。正則な場合は、逆行列を求めてから u, v, t を求める。交点の幾何的な位置関係 (たとえば三角形 ABC 内になるかなど) を調べたい場合は、それぞれの値と幾何条件から判断すればよい。

5.2 原点からの距離を利用する解法

?? 節で述べた正規化法線ベクトル $\mathbf{N}$ 及び原点からの距離 d を用いると、次のような公式で直線式のパラメータ t を求めることができる。

$$t = \frac{d - \mathbf{N} \cdot \mathbf{P}}{\mathbf{N} \cdot (\mathbf{Q} - \mathbf{P})}$$

d を求めるのにある程度の計算コストがかかるが、それ以外は前節の連立方程式の解法よりも高速である。従って、次の条件を満たす場合にはこちらの方が高速に処理することができる。

1. 平面が空間上を移動することがほとんどなく、あらかじめ d を計算しておくことができる場合。
2. 平面上のパラメータ u, v を求める必要が無い場合。

6 サンプルプログラム

```
 1: #include <FK/FK.h>
 2: #include <FL/Fl_Value_Slider.h>
 3:
 4: // スライダー作成用関数
 5: Fl_Value_Slider *newSlider(int x, int y, int w, int h, char *label)
 6: {
 7:     Fl_Value_Slider     *slider;
 8:
 9:     slider = new Fl_Value_Slider(x, y, w, h, label);
10:     slider->type(FL_HOR_NICE_SLIDER);
11:     slider->minimum(-100.0);
12:     slider->maximum(100.0);
```

```
13:     slider->value(0.0);
14:     slider->labelsize(12);
15:     slider->textsize(12);
16:
17:     return slider;
18: }
19:
20: int main(int argc, char *argv[])
21: {
22:     Fl_Window       mainWin(540, 320, "TEST");
23:     fk_Window       viewWin(10, 10, 300, 300);
24:     Fl_Value_Slider *sliderX1, *sliderY1, *sliderZ1;
25:     Fl_Value_Slider *sliderX2, *sliderY2, *sliderZ2;
26:     fk_Vector       start, end, pos[3];
27:     fk_Line         lines;
28:     fk_Polygon      poly;
29:     fk_Light        light;
30:     fk_Model        lineModel, polyModel, lightModel, cameraModel;
31:     fk_Scene        scene;
32:
33:     // 各スライダー生成
34:     sliderX1 = newSlider(320, 20, 200, 20, "X1");
35:     sliderY1 = newSlider(320, 70, 200, 20, "Y1");
36:     sliderZ1 = newSlider(320, 120, 200, 20, "Z1");
37:     sliderX2 = newSlider(320, 170, 200, 20, "X2");
38:     sliderY2 = newSlider(320, 220, 200, 20, "Y2");
39:     sliderZ2 = newSlider(320, 270, 200, 20, "Z2");
40:
41:     mainWin.end();
42:     fk_InitMaterial();
43:
44:     // 線分生成
45:     start.set(0.0, 0.0, 0.0);
46:     end.set(0.0, 0.0, 0.0);
47:     lines.pushLine(start, end);
48:     start.set(-60.0, 70.0, 40.0);
49:     end.set(70.0, -40.0, -50.0);
50:     lines.pushLine(start, end);
51:     lineModel.setShape(&lines);
52:     lineModel.setLineColor(0.0, 1.0, 1.0);
53:
54:     // 三角形生成
55:     pos[0].set(-50.0, 70.0, 80.0);
56:     pos[1].set(-20.0, -40.0, -60.0);
57:     pos[2].set(30.0, -20.0, -50.0);
58:     poly.pushVertex(pos[0]);
59:     poly.pushVertex(pos[1]);
60:     poly.pushVertex(pos[2]);
61:     polyModel.setShape(&poly);
62:     polyModel.setMaterial(Yellow);
63:
64:     // 光源設定
65:     lightModel.setShape(&light);
66:     lightModel.setMaterial(White);
67:     lightModel.glFocus(0.0, 0.0, -1.0);
68:
69:     // カメラ設定
70:     cameraModel.glMoveTo(0.0, 0.0, 500.0);
```

```
 71:     cameraModel.glFocus(0.0, 0.0, 0.0);
 72:     cameraModel.glUpvec(0.0, 1.0, 0.0);
 73:
 74:     scene.entryModel(&lineModel);
 75:     //  scene.entryModel(&polyModel);
 76:     scene.entryModel(&lightModel);
 77:     scene.entryCamera(&cameraModel);
 78:     viewWin.setScene(&scene);
 79:
 80:     mainWin.show();
 81:     viewWin.show();
 82:     while(true) {
 83:         if(mainWin.visible() == 0) {
 84:             if(Fl::wait() == 0) {
 85:                 break;
 86:             } else {
 87:                 continue;
 88:             }
 89:         }
 90:         if(viewWin.drawWindow() == 0) break;
 91:         if(Fl::check() == 0) break;
 92:         if(viewWin.winOpenStatus() == false) continue;
 93:
 94:         // 線分の視点と終点をスライダーから取得
 95:         start.set(sliderX1->value(), sliderY1->value(), sliderZ1->value());
 96:         end.set(sliderX2->value(), sliderY2->value(), sliderZ2->value());
 97:         // 線分再設定
 98:         lines.changeLine(0, start, end);
 99:     }
100: return 0;
101: }
```

7 演習問題

ここでの課題は行列演算を多用するので、fk_Vector クラスと fk_Matrix クラスを適宜利用するとよい。

(1) サンプルプログラム中の二直線に関し、3 で述べられている交点の算出法を用いて s, t を算出し、対応するそれぞれの場所に半径 10 の球を配置するように修正せよ。

(2) (1) に加え、$-1 < p < 1$ を満たす場合は二直線が交わっていると仮定し、その場合は球が赤く表示され、交わっていない場合には黄色く表示されるように修正せよ。

(3) サンプルプログラムの 50 行目の部分をコメントアウトし、75 行目のコメントアウトを外して三角形が表示されるように修正せよ。

(4) (3) の状態に対し、直線と平面の交点の部分が存在する場合に、交点の箇所に赤い球が配置されるように修正せよ。

(5) (4) の条件に加え、線分と三角形の交点の部分が存在する場合に、交点の箇所に赤い球が配置されるように修正せよ。