**Kinectでビームを出すには**

安藤　史将

Fumimasa Ando

法政大学情報科学部ディジタルメディア学科

*Student ID:15K1004*

*E-mail:* *fumimasa.ando.9i@stu.hosei.ac.jp*

# **1. まえがき**

Kinectはプロジェクトで教授に教えてもらい初めて知ったが、プログラミング言語はC++ということで習っていたこともあり、もしかしたら自分でも作れるのかもしれないと興味を持った。今回作ったものはプログラムのサンプルにあったスペシウム光線とエメリウム光線を改良したものであるが、その発想に至ったのは昔アニメで見た手からビームを出すキャラクター達を思い出し、自分がそれを再現できるかどうか試してみようと思い、今回の作品を作った。

# **2. プログラムにおける基本情報**

**2.1 OpenGLのプログラミング**

OpenGL とは、2次元、3次元のコンピュータグラフィックスを描くための共通のライブラリ集である。OpenGLはVisual Studioというプログラミングツールでプログラムすることができる。これは、マイクロソフト社が開発した総合開発環境であり、OpenGLはC++でプログラミングをすることができる。

•display();

　描画関数。この関数を使うことで描画することができる。

• glColor3f(1.0, 0.0, 0.0); // R, G, B

• 3f で終わる関数は、float 3変数

glColor3fを使うことによって描画するものの色を設定できる。

• glBegin(図形名) と glEnd() の間で、図形を定義

　この間で書いた関数が表示される。

• glVertex3f(x, y, z):

3次元の float の頂点

ここで設定した座標から次の座標へと描画することができる

**2.2 Kinectのプログラミング**

Kinectはマイクロソフトから発売された、ジェスチャーや音声認識によって操作ができるデバイスであり、プログラミングにはいくつか特徴がある。

• CameraSpacePoint

• カメラの視点を原点にした X, Y, Z の右手系の3次元座標系

3次元の位置座標を表現するための「構造体」

• ひとつの構造体の中に、float が3個入っている。

typedef struct \_CameraSpacePoint {

float X;

float Y;

float Z;

} CameraSpacePoint;

• 使い方

CameraSpacePoint p;

p.X = 0.2;

p.Y = 0.1;

p.Z = 1.2;

• Body 情報

体の位置を読み取る。

•pJoint[JointType\_…].Position

…の部分に体の位置を入力し、CameraSpacePointに構造体として情報を入力する。

CameraSpacePoint\*right=pJoint[JointType\_HandRight].Position;

　右手の情報

CameraSpacePoint\*hright=pJoint[JointType\_HandTipRight].Position;

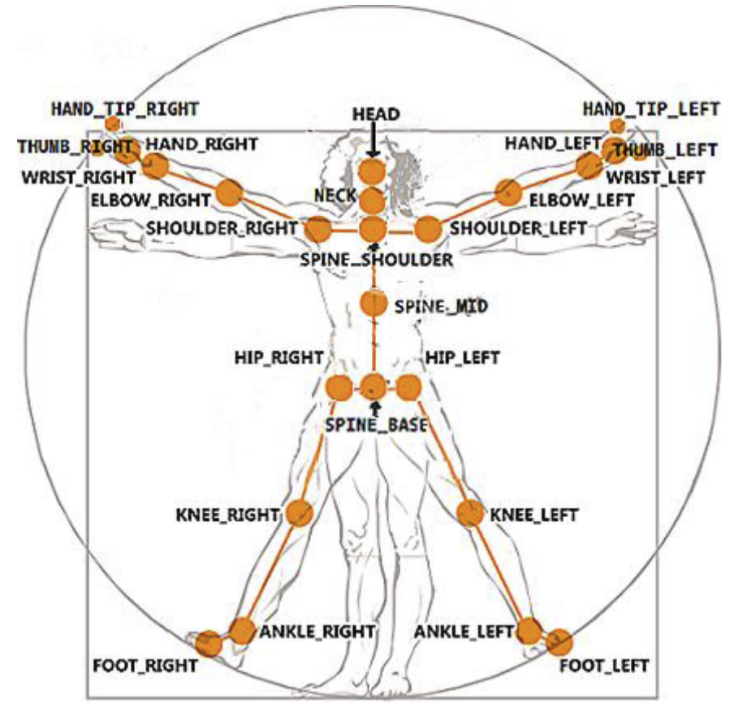
　右手の人差し指の情報

CameraSpacePoint\*elbow=pJoint[JointType\_ElbowRight].Position;

　右ひじの情報

CameraSpacePoint\*shoulder=pJoint[JointType\_ShoulderRight].Position;

　右肩の情報



float[3] の配列と併用

• CameraSpacePoint を使うと、3要素の計算

に、p.X, p.Y, p.Z と記述するため、面倒

• float[3] を利用すると、for文で3回回すこと

ができて便利

•正規化

長さを1にする

• void norm(CameraSpacePoint\* xp,

CameraSpacePoint\* rp);

• void norm(float\* x, float\* result);

ポインタでやるには

CameraSpacePoint p1, p2;// p1 の設定

norm(&p1, &p2);

正規化すると、処理する速度が速くなるため必ず正規化したほうがよい。

•方向ベクトル

• ポイントからポイントへの方向ベクトル

• 結果は、float [3] に格納

void direction(CameraSpacePoint\* xp,

CameraSpacePoint\* yp, float\* dirp) {

dirp[0] = yp->X - xp->X;

dirp[1] = yp->Y - xp->Y;

dirp[2] = yp->Z - xp->Z;

}

xpからypへの方向をdirpに保存

• 内積を用いて、2個のベクトルのはさむ角度θ

の cos を計算。

　今回は構造体でポインタを置いているので、この関数を使う

float cosine(CameraSpacePoint\* xp,

CameraSpacePoint\* yp) {

return dot(xp, yp) / length(xp) / length(yp);

}

**2.3 KinectとOpenGL との融合**• 初期化

• main() の中で、Kinect の準備作業を一通り行う。

• ソースコードでは、kinectInit() という関数の中で、準備作業をまとめている。

• 画像の読み込み

• OpenGL には、idle() 関数という「暇な時に呼ばれる」関数を定義できるので、この idle() 関数の中で、Kinect のイベント待ちを行えばよい。

• 描画

• idle()の中で画像の読み込みが終わったら、glutPostRedisplay() を呼び出して、display() 関数の中で、　画像データを OpenGL の画面に描き出す。

# **3. 実際のプログラム**

始めにVoid関数で今回のプログラムを作る。

Void beams(Joint\* pJoint , int iro)

今回、引数は体の情報を読み取る構造体のpJointと、手の形を判断して色を決定するためにintを使う。

始めにpJointで得た情報をもとに体の位置を判断する。

情報をCameraSpacePointによって構造体に入れる。

CameraSpacePoint\*right=pJoint[JointType\_HandRight].Position;

　右手の情報

CameraSpacePoint\*hright=pJoint[JointType\_HandTipRight].Position;

　右手の人差し指の情報

CameraSpacePoint\*elbow=pJoint[JointType\_ElbowRight].Position;

　右ひじの情報

CameraSpacePoint\*shoulder=pJoint[JointType\_ShoulderRight].Position;

　右肩の情報

次にfloat[3]を2つ作り、direction関数を使い、右肩から肘と肘から右手の方向ベクトルを決める。これは、肘を伸ばしたときにビームが出るようにしたかったので、cos関数を使って二つの方向ベクトルの角度を求めて判断するためである。

direction(&elbow, &right, dir0);

direction(&shoulder, &elbow, dir1);

if (cosine(dir0, dir1) > 0.3 ) {

これでif文が成り立つとき、作動するようになる。

float dir[3];

direction(&elbow, &right, dir);

今度も同様に光が出る向きを決めるため、dirに肘から右手の向きに出るように設定。

次は色の設定であるが、今回は手の形が「パー」(iro=0)だと様々な色、「グー」(iro=2)だと黒色、「チョキ」(iro=1)だと白色のビームが出るように設定する。

if (iro == 1){

glColor3f(0.0, 0.0, 0.0);

}

else if (iro == 2){

glColor3f(1.0, 1.0, 1.0);

}

else if (iro == 0){

double x = (float)rand() / RAND\_MAX;

double y = (float)rand() / RAND\_MAX;

double z = (float)rand() / RAND\_MAX;

glColor3f(x, y, z);

}

ここで、「パー」のときに様々な色にするために乱数を使う。glColor3fの中身は0~1の値になるので、(float)rand() / RAND\_MAXと置くことで、値を0~1の乱数に調節することができる。

ここで、一通り設定が終わったのでいよいよ描画していこうと思う。

まず、glBegin(GL\_LINES);で始める。

そして、ビームが出るような見た目にするために出る角度や長さを乱数で変えたいのでfor文で回す。

for(int i = 0; i < 100; i++) {

　　ここで光の長さを決める。

float len = rand() % 300 / 100.0;

float wide = 0.03;

乱数で、少しずつ光線の方向をずらす

float vec[3];

vec[0] = (rand() % 200 - 100) / 100.0;

vec[1] = (rand() % 200 - 100) / 100.0;

vec[2] = (rand() % 200 - 100) / 100.0;

方向を正規化する。

norm(vec, vec);

右手からビームを出すために始点の座標を決める。

glVertex3f(right.X, right.Y, right.Z);

ビームが行き着く終点の座標を決める。

glVertex3f(right.X + dir[0] \* len + vec[0] \* wide,

right.Y + dir[1] \* len + vec[1] \* wide,

right.Z + dir[2] \* len + vec[2] \* wide);

ここでは、(座標+方向ベクトル\*長さ+散らばりのベクトル\*広さ)をしている。

}

最後にglEnd();で終了。

これでビームは出るのだが、手の形を判断するプログラムを作らなければならない。これは体の情報を読み取ったときに画面を出力する関数の中で判断している。KinectにはHandStateという自動で手の形を判断してくれる構造体がある。HandState の構造体から返ってくる数字によって「4.グー」「3.チョキ」「2.パー」を判断することができる。

始めにhandstateという構造体を作る

HandState handstate;

ここで体の情報を構造体に入れる。

pBody[i]->get\_HandRightState(&handstate);

int iro;

handstateが2のときは手の形は「パー」

3のときは「チョキ」4のときは「グー」

であるのでif文を使って状況に応じて色を変える。

そして、pBody[i]->GetJoints(JointType\_Count, pJoint);この構造体を使うことでpJointに情報を入れる。

if (handstate==2) {

iro = 0;

pBody[i]->GetJoints(JointType\_Count, pJoint);

beams(pJoint, iro);

}

else if(handstate==3){

iro = 1;

pBody[i]->GetJoints(JointType\_Count, pJoint);

beams(pJoint, iro);

}

else if (handstate == 4){

iro = 2;

pBody[i]->GetJoints(JointType\_Count, pJoint);

beams(pJoint, iro);

}

これで、「グー」「チョキ」「パー」でint iroを「2」「1」「0」と設定し、beams関数に送ることができた。

これで、関数に手の情報が送られ、それに応じたプログラムが動く。

# **4. 動作**

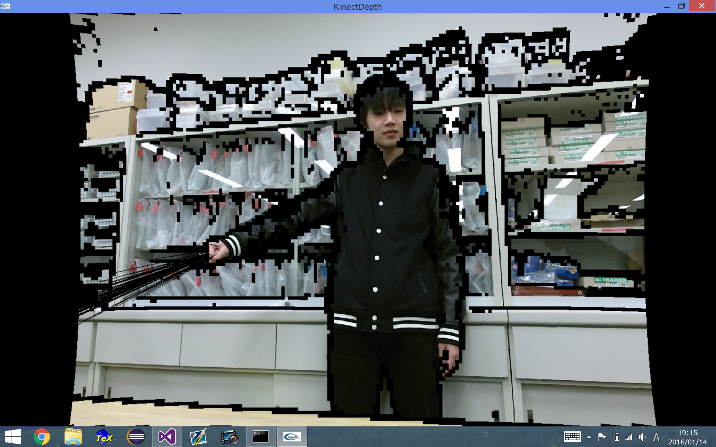
今回は、「パー」の時に様々な色でビームを出すというのが静止画では分かりづらいため、動画で撮影を行った。



※インターネットに接続していなければ見れません！

・静止画

「グー」

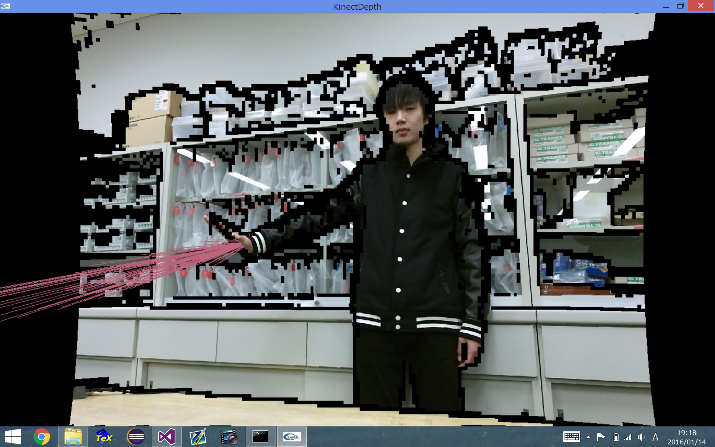


「チョキ」



「パー」





# **5. 考察**

今回、Kinectの使い方、OpenGLの使い方、C++を用いたVisual Studioの使い方を学んだ。Kinectにおいては全く知識がなく、教授に頼りきりだったが最後には自分で考えて動かすことができてよかったと思う。CameraSpacePointの使い方やHandStateなど様々な機能がKinectには備わっており、それらを駆使してプログラムするのはとても難しく、まだまだやりがいがあると思えた。OpenGLにも全く同じことが言える。予備知識がなかったので使いこなすとまではいけなかったが、最初よりはうまく扱えるようにはなっていたと思う。できることならば、まだまだKinectとOpenGLを調べればより豊かな発想を持って作品を作れるのだろうなと思い、もっと研究がしたくなった。C++は前々から学んでいたこともあり、Visual Studioを使う際にはそこまで苦労はしなかった。しかし、Kinectとの関係を繋げるにはやはり新しい発想が必要で、学べるところが多くこれからの知識に非常に役立つ発想を得ることができた。

そして今回は何より、Kinect,OpenGL,Visual Studio(C++)の3つをうまく扱って作品を作る必要があり、学んでいくにつれて3つの言語を使うならではの発想の転換や多量の知識を学ぶことができたと思う。そして次に研究するときには、今回の研究で鍛えた発想や経験を生かしてよりクオリティの高い作品を作りたいと思う。

# **6. むすび**

今回は初めての研究の割には、自分なりにちょっと工夫した作品ができた。次回には、今回の研究を踏まえたより質の高い作品を作りたいと思う。

**文　献**

1. 法政大学, “藤田悟研究所 2015年度秋学セメスタ”, http://cis.k.hosei.ac.jp/~fujita/local/projects/fall2015/index.html, 2015年1月15日参照.