「InteractionBox」を使う

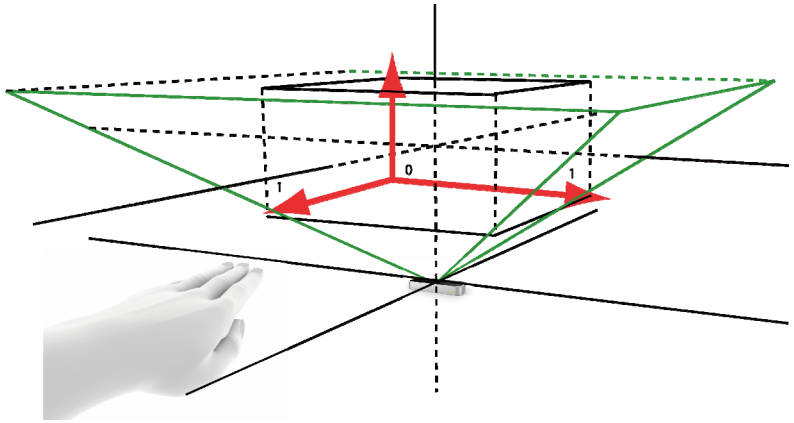
ここでは、「Leap Motion」の「座標を変換する方法」について解説します。

＊

「Leap Motion V1」では、いくつかの座標変換の方法がありましたが、「Leap Motion V2」では「InteractionBox」の変換のみとなっています

※より正確には、「インターセクション・ポイント」(Intersection Point) および「プロジェクション・ポイント」(Projection Point) V1.2から非推奨となりました。

「InteractionBox」のイメージは、次の図です。



「InteractionBox」のイメージ

「Leap Motion」の逆ピラミッドの中に直方体の検出環境を作ります。この直方体の左下奥を(0,0,0)として、「0」から「1」までの範囲で値が変化します。

たとえば、この値にウィンドウの「幅」や「高さ」を掛けることで、「指がウィンドウのどの位置にいるか」ということを知ることができます。

これを使ったプログラムは「**4-2　お絵かきツールを作成する**」で解説しています。

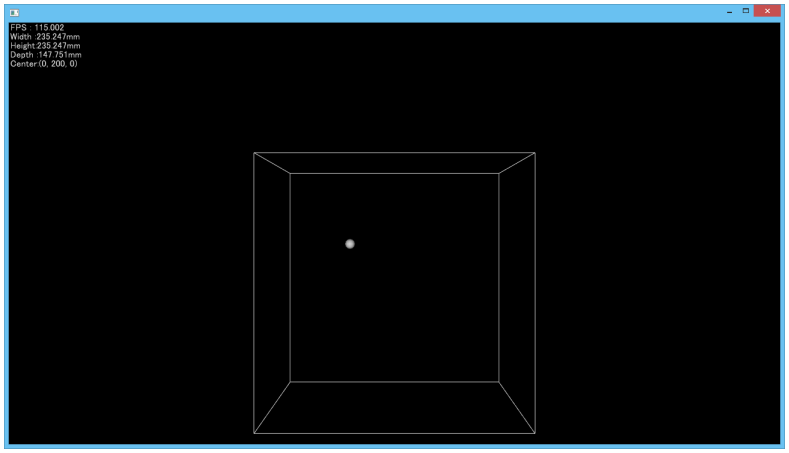
変換された位置の動きを見る

「InteractionBox」がどのように動作するのか見てみましょう。

■ プログラム

全体のコードは「InteractionBox01」にあります。

「InteractionBox」を模した枠の中を、指の位置を表わした球が動きます。



このプログラムの実行結果

● update()

フレームの更新と各座標の取得を行ないます。

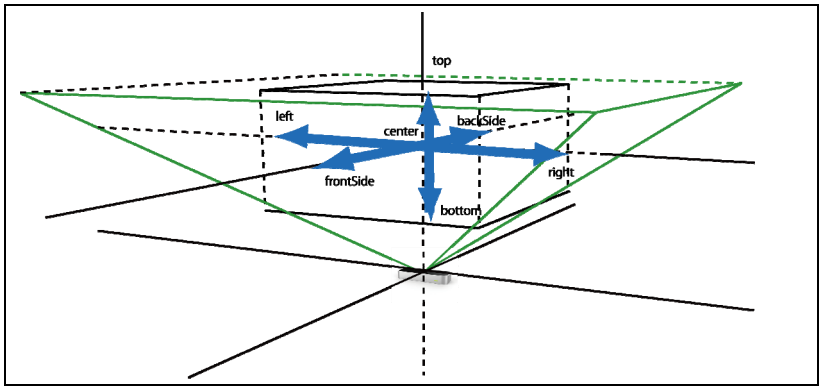
「InteractionBox」からは、「中心の座標」と「幅」「高さ」「奥行き」がそれぞれ取得(単位はすべて mm(ミリメートル))できるので、計算して、「左右」「上下」「前後」の位置を求めます。

```
void update()
{
  // フレームの更新
  mLastFrame = mCurrentFrame;
  mCurrentFrame = mLeap.frame();

  iBox = mCurrentFrame.interactionBox();

  mLeft = iBox.center().x - (iBox.width() / 2);
  mRight = iBox.center().x + (iBox.width() / 2);
  mTop = iBox.center().y + (iBox.height() / 2);
  mBaottom = iBox.center().y - (iBox.height() / 2);
  mBackSide = iBox.center().z - (iBox.depth() / 2);
  mFrontSide = iBox.center().z + (iBox.depth() / 2);

  renderFrameParameter();
}
```

● renderFrameParameter()

「幅」「高さ」「奥行き」の値を表示します。

```
void renderFrameParameter()
{
  …

  ss << "Width :" << iBox.width() << "mm" << "¥n";
  ss << "Height:" << iBox.height() << "mm" << "¥n";
  ss << "Depth :" << iBox.depth() << "mm" << "¥n";
  ss << "Center:" << iBox.center() << "¥n";

  …
}
```

● drawLeapObject()

「drawInteractionBoxFrame()」で「InteractionBox」の枠を描画し、「drawFingerPoint()」で「InteractionBox」内の指の位置を描画します。

```
void drawLeapObject()
{
  …

  // 表示処理
  drawInteractionBoxFrame();
  drawFingerPoint();

  …
}
```

● drawInteractionBoxFrame()

「update()」で求めた頂点の座標から、「InteractionBox」の枠を描画します。

```
void drawInteractionBoxFrame()
{
  // 中心点
  //gl::drawSphere( toVec3f( iBox.center() ), 5 );

  // 上面
  gl::drawLine( Vec3f( mLeft, mTop, mBackSide ),
                Vec3f( mRight, mTop, mBackSide ) );

  gl::drawLine( Vec3f( mRight, mTop, mBackSide ),
                Vec3f( mRight, mTop, mFrontSide ) );

  gl::drawLine( Vec3f( mRight, mTop, mFrontSide ),
                Vec3f( mLeft, mTop, mFrontSide ) );
```



```
  gl::drawLine( Vec3f( mLeft, mTop, mFrontSide ),
                Vec3f( mLeft, mTop, mBackSide ) );

  // 下面
  gl::drawLine( Vec3f( mLeft, mBaottom, mBackSide ),
                Vec3f( mRight, mBaottom, mBackSide ) );

  gl::drawLine( Vec3f( mRight, mBaottom, mBackSide ),
                Vec3f( mRight, mBaottom, mFrontSide ) );

  gl::drawLine( Vec3f( mRight, mBaottom, mFrontSide ),
                Vec3f( mLeft, mBaottom, mFrontSide ) );

  gl::drawLine( Vec3f( mLeft, mBaottom, mFrontSide ),
                Vec3f( mLeft, mBaottom, mBackSide ) );

  // 側面
  gl::drawLine( Vec3f( mLeft, mTop, mFrontSide ),
                Vec3f( mLeft, mBaottom, mFrontSide ) );

  gl::drawLine( Vec3f( mLeft, mTop, mBackSide ),
                Vec3f( mLeft, mBaottom, mBackSide ) );

  gl::drawLine( Vec3f( mRight, mTop, mFrontSide ),
                Vec3f( mRight, mBaottom, mFrontSide ) );

  gl::drawLine( Vec3f( mRight, mTop, mBackSide ),
                Vec3f( mRight, mBaottom, mBackSide ) );
}
```

● drawFingerPoint()

「人差し指」の座標を描画します。

```
void drawFingerPoint()
{
  // 人差し指を取得する
  Leap::Finger finger = mLeap.frame()
                            .fingers()
                            .fingerType( Leap::Finger::Type::TYPE_
INDEX )[0];
  if ( !finger.isValid() ) {
    return;
  }

  Leap::Vector normalizedPosition =
             iBox.normalizePoint( finger.tipPosition() );

  // 位置の割合から実際の座標を計算
  // 原点を左下奥にする
  auto x = (normalizedPosition.x * iBox.width()) + mLeft;
  auto y = (normalizedPosition.y * iBox.height()) + mBaottom;
  auto z = (normalizedPosition.z * iBox.depth()) + mBackSide;
  gl::drawSphere( Vec3f( x, y, z ), 5 );
}
```

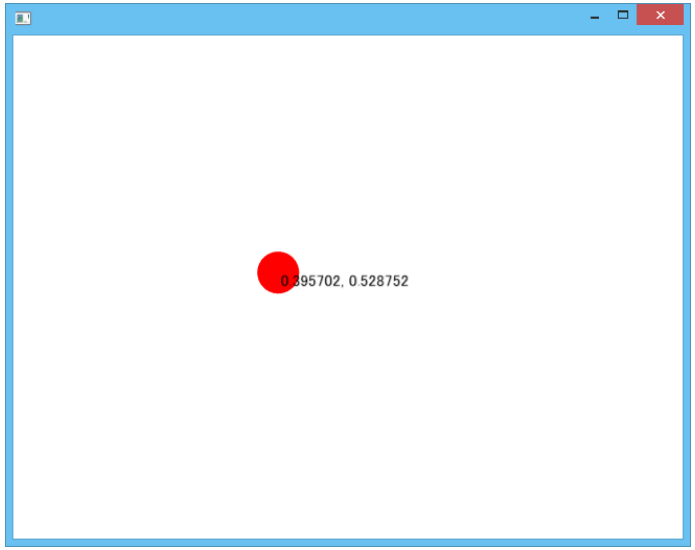
現在のフレームで検出された指から、「人差し指」を取得します。

その指が有効であれば、指先の座標を「InteractionBox」の位置に変換します。

変換された「X,Y,Z」の位置に「幅」「高さ」「奥行き」の値を掛け、原点(左下奥)中心にすることとで、「InteractionBox」の座標になります。

2 次元の位置座標として利用する

「InteractionBox」で変換された位置割合のうち、「X,Y」を使うことで、2 次元の位置座標として利用することができます。

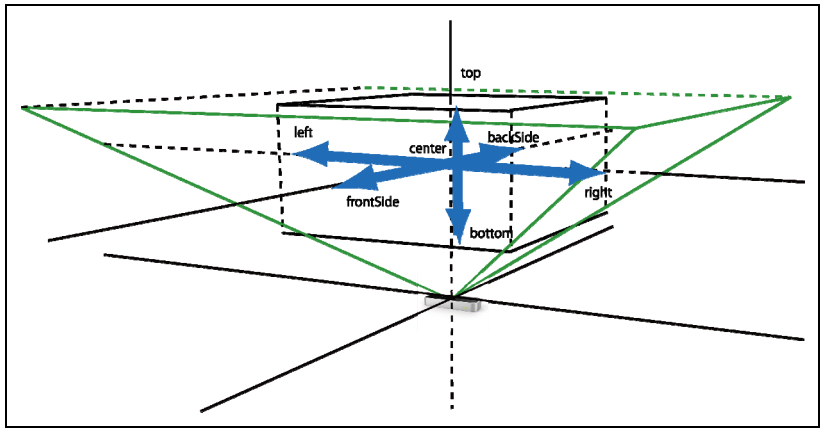


■ プログラム

それではコードを見てみましょう。

全体のコードは「InteractionBox02」にあります。

● draw()



「InteractionBox」の座標に変換します。

```
// 人差し指を取得する
Leap::Finger finger = mLeap.frame()
                          .fingers()
                          .fingerType( Leap::Finger::Type::TYPE_IN
DEX )[0];
if ( !finger.isValid() ) {
  return;
}

// InteractionBoxの座標に変換する
Leap::InteractionBox iBox = mLeap.frame().interactionBox();
Leap::Vector normalizedPosition =
               iBox.normalizePoint( finger.stabilizedTipPosition
() );
```

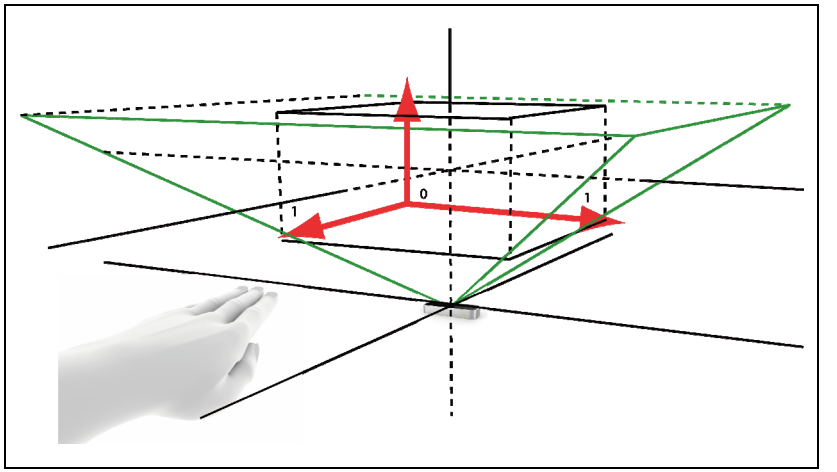
「InteractionBox」は、「Leap::Frame::interactionBox()」で取得

できます。

変換したい座標を「Leap::InteractionBox::normalizePoint()」に指定します。

「normalizePoint()」で返される座標系は「InteractionBox」の左下奥が原点となり、それぞれ右、上、手前方向に「0」から「1」の範囲の InteractionBox の位置の割合で変化します。

ここでは、あらかじめ取得した人差し指の座標を変換しています。



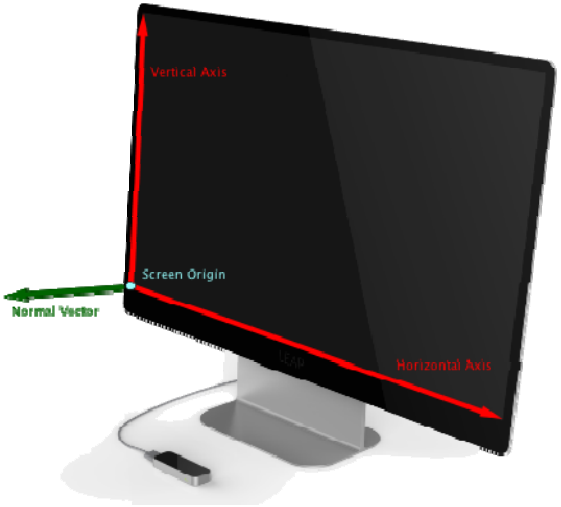
「normalizePoint()」で返される座標系

これを、スクリーン(ウィンドウ)の座標に変換します。

```
// ウィンドウの座標に変換する
float x = normalizedPosition.x * WindowWidth;
float y = WindowHeight - (normalizedPosition.y * WindowHeight);
```

「X 座標」はそのままウィンドウサイズを掛ければよいのですが、「Y 座標」は「InteractionBox の原点が下」「ウィンドウの原点が上」

なので、上下を反転しています。



これで指の位置をスクリーン(ウィンドウ)の位置に合わせることができました。

タッチ状態を判定する

ここまでで「Leap Motion」の座標とスクリーン(ウィンドウ)の座標を結び付けることができました。

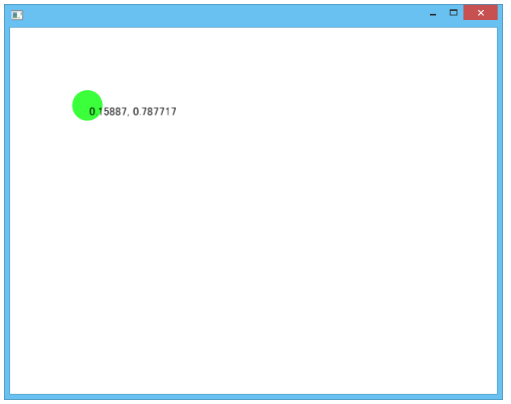
ここにタッチスクリーンでのタッチ状態を模擬した判定を加えることで、非接触のタッチスクリーンを実現できます。

「Leap Motion SDK」では、タッチの判定を API としてもっており、それを利用することで簡単に実装できます。

先のプログラムにタッチの判定を追加することで、次のように動作します。

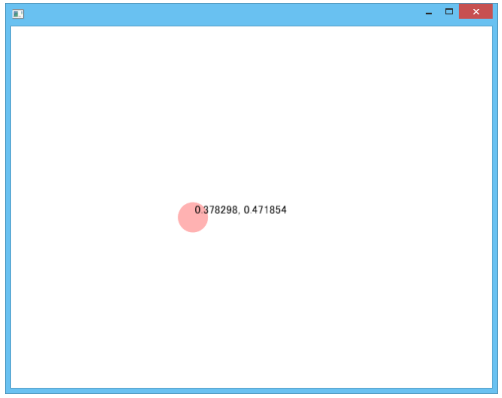
ホバー状態(指を検出しているがタッチはしていない状態)では、指の

位置が緑色で表わされます。



ホバー状態

タッチと判定されると、指の位置が赤くなります。



タッチ状態

非接触のインターフェイスでは、操作しようとしている位置(ここでは指)がわかりづらいことが難点です。

ホバー状態を明確に判定することで、これからタッチしようとしている位置をユーザーに知らせることができ、より使いやすい操作体系を提

供できます。

■ プログラム

それではコードを見てみましょう。

全体のコードは「InteractionBox03」にあります。

ほとんどのコードは「InteractionBox02」であり、タッチの判定コードのみ追加しています。

```
void draw()
{
  …

  // ウィンドウの座標に変換する
  float x = normalizedPosition.x * WindowWidth;
  float y = WindowHeight - (normalizedPosition.y * WindowHeight);

  // ホバー状態
  if ( finger.touchZone() == Leap::Pointable::Zone::ZONE_HOVERING
 ) {
    gl::color(0, 1, 0, 1 - finger.touchDistance());
  }
  // タッチ状態
  else if( finger.touchZone() == Leap::Pointable::Zone::ZONE_TOUC
HING ) {
    gl::color(1, 0, 0, -finger.touchDistance());
  }
  // タッチ対象外
  else {
    gl::color(0, 0, 1, .05);
  }
```



```
  gl::drawSolidCircle( Vec2f( x, y ), 20 );

  …
}
```

指の位置を描画する前に、タッチ状態によって色を変えています。

タッチの状態は、「Leap::Pointable::Zone 列挙体」で定義され、「Leap::Pointable::touchZone()」で取得できます。

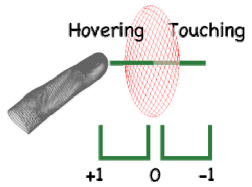
「Leap::Pointable::Zone 列挙体」には、次の値があります。

値	意　味
ZONE_NONE	タッチでもホバーでもない
ZONE_HOVERING	ホバーしている
ZONE_TOUCHING	タッチしている

また、タッチ状態の度合い(タッチの深さ)を取得することもできます。

この値は「Leap::Pointable::touchDistance()」で取得できます。

「タッチの状態」(touchZone()の値)と「タッチの度合い」(touchDistance()の値)の関係は次のようになります。



「Leap Motion」でタッチを判定する