KINECT for Windowsを使用したアプリケーションの制作

井上雅裕　上田洋平
小野弘貴　片山陽亮

１．研究概要

体感型ゲームデバイス「KINECT」（写真１）を使用し,身振り手振りといったジェスチャによるコンピュータの操作アプリケーションプログラムを制作した。

写真１　KINECT for Windows

２．研究の具体的内容

（１）KINECT for Windowsの概要

KINECTとは,2010年にMicrosoft社から発売されたXbox専用のゲームデバイスで,コントローラーを使用することなく身体を使って直観的な操作ができる体感型のゲームシステムである。また,KINECTに搭載されているマルチアレイマイクロフォンにより音声よる操作も可能である。

KINECTにはマルチアレイマイクロフォンの他にもRGBカメラや深度センサーが搭載されており,これらを利用すると図１のように,プレイヤがKINECTの認識範囲内に立つことで自動的にプレイヤが認識されるようなプログラムを組むことができる。現在KINECTは２種類ありゲームデバイスとして発売されたKINECT（Xbox版）と,コンピュータで使用することを目的としたKINECT for Windowsの２種類がある。

２つの相違点を次に示す。

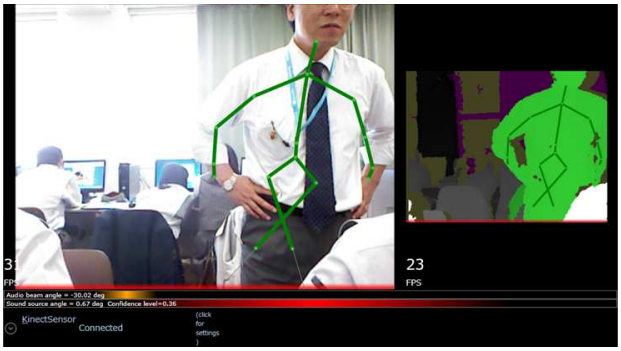

図１　プレイヤを認識

【KINECT for Windows】
（2012年２月２日発売）

- Kinect for Windows SDKのすべての機能を利用できる
- Xbox360で使用不可
- 40cmの距離から認識できる近距離モード搭載(Nearモード)
- 過電圧防止ドングルが付属
- ケーブルの長さ(3m → 1.5m)
- 本体前面のロゴ(Xbox 360 → KINECT)
- 商用利用可

【KINECT（Xbox版)】
（2010年11月20日発売）

- 個人用途または開発時に利用できる
- Kinect for Windows SDKの一部の機能が使用可能
- Nearモードはない
- Xbox360でも利用できる

このように,２種類のKINECTには多くの相違点がある。ここでKINECT for Windowsの「Nearモード」に注目した。KINECTで身体を認識させる際には,KINECTの認識範囲に入るためにある程度KINECTから離れなければならない。KINECT(Xbox版)の場合80センチ以上である。しかし,KINECT for Windows「Nearモード」

を使用するとこの距離を 40 センチまで縮めることが可能である。研究目的は「KINECT を使用したマウス操作」であったため,より近い距離での KINECT の使用を想定していた。そのため,「Near モード」を利用することができ,KINECT(Xbox 版)よりも認識範囲の広い KINECT for Windows を利用して研究を進めていくことに決定した。

（２）開発環境

開発環境は Microsoft Visual Studio 2008 を使用し,ライブラリは Kinect for Windows SDK を使用した言語は Visual C#を使用した。

（３）KINECT を使ったマウス操作の概要

```
  StartKinect(KinectSensor.KinectSensors[0]);
}

/// <summary>
/// RGBカメラ、スケルトンのフレーム更新イベント
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void kinect_AllFramesReady( object sender, AllFramesReadyEventArgs e )
{
  try {
    KinectSensor kinect = sender as KinectSensor;

    using ( ColorImageFrame colorFrame = e.OpenColorImageFrame() ) {
      if ( colorFrame != null ) {
        byte[] colorPixel = new byte[colorFrame.PixelDataLength];
        colorFrame.CopyPixelDataTo( colorPixel );
        imageRgb.Source = BitmapSource.Create( colorFrame.Width, colorFrame.Height, 96, 96,
            PixelFormats.Bgr32, null, colorPixel, colorFrame.Width * colorFrame.BytesPerPixel );
      }
    }

    using ( SkeletonFrame skeletonFrame = e.OpenSkeletonFrame() ) {
      if ( skeletonFrame != null ) {
        // トラッキングされているスケルトンのジョイントを描画する
        Skeleton skeleton = skeletonFrame.GetFirstTrackedSkeleton();

        if ( (skeleton != null) && (skeleton.TrackingState == SkeletonTrackingState.Tracked) ) {
          Joint hand = skeleton.Joints[JointType.HandRight];
          if ( hand.TrackingState == JointTrackingState.Tracked ) {
            ImageSource source = imageRgb.Source;
            DrawingVisual drawingVisual = new DrawingVisual();
```

図３ マウス操作

このプログラムは,研究の目的である「KINECT を使用したマウス操作」を実現することのできるプログラムである

このプログラムの大まかな流れは

① KINECT 接続の確認
② 人の有無
③ 骨格認識
④ 右手の位置の検索
⑤ 右手の位置の座標をコンピュータ内の座標に置き換える
⑥ 座標の位置にマウスの座標を移動

以降④⑤⑥を繰り返すような内容である。

また,左クリックの動作は同じ座標位置に３秒以上停止していると自動で左クリックをする使用である。このプログラムを実際に使用してみると画面の端までマウスが移動しないことや,左クリックの動作が連続して行われてしまうという問題点が見つかった。

この問題点を解決するため,一度左クリックをすると２秒以上のインターバルを空けないともう一度左クリックできない様にソースを変更した。また,使用者の動きが小さくても,画面のマウスが大きく動作するように倍率を変更することでマウスが画面の端まで移動することができるようになった。図４―２は,変更したソースの一部である。

```
imageRgb.Source = bitmap;

// Frame中の手の位置をディスプレイの位置に対応付ける
ColorImagePoint point = kinect.MapSkeletonPointToColor( hand.Position,
  kinect.ColorStream.Format );
System.Windows.Forms.Screen screen = System.Windows.Forms.Screen.AllScreens[0];
point.X = ((point.X * screen.Bounds.Width) / (kinect.ColorStream.FrameWidth )) ;
point.Y = ((point.Y * screen.Bounds.Height) / (kinect.ColorStream.FrameHeight )) ;
```

図４－１プログラム改善前

```
imageRgb.Source = bitmap;

// Frame中の手の位置をディスプレイの位置に対応付ける
ColorImagePoint point = kinect.MapSkeletonPointToColor( hand.Position,
  kinect.ColorStream.Format );
System.Windows.Forms.Screen screen = System.Windows.Forms.Screen.AllScreens[0];
point.X = ((point.X * 3 * screen.Bounds.Width) / (kinect.ColorStream.FrameWidth * 2)) - 200;
point.Y = ((point.Y * 3 * screen.Bounds.Height) / (kinect.ColorStream.FrameHeight * 2)) - 200;
```

図４－２プログラム改善後

図４－２は,実際の使用者の右手の位置の座標をマウスの座標に置き換える際の倍率を変更している所である。何度も実験を繰り返し,最適な動作を行うことができるパラメータを設定した。右クリックや左クリックを押したままの動作など,細かく調整すると終わりはないので,今回はとにかく当初の目的であった「KINECT を使用したマウス操作」がこれで達成することができた。

（４）アルバムめくりアプリケーション

このアプリケーション「文化祭での発表で何か面白いことはできないか？」と考え,制作したものである。KINECT を使用したマウス操作では動きに華やかさがないためあまり文化祭での発表に向いていないと判断した。そこで,写真を手でめくっていくという大きな動きができるこのアプリケーションが最適だと考えた。

このプログラムの流れは

① KINECT の有無
② 人の骨格同期
③ 右手 or 左手の判断
④ 腕の動作の認識

⑤　写真移動

となっている。ソースの大部分は「マウス操作」で使用したものを流用し,腕の動作に対応してスライドのページめくりがおこなうようにしている。

図５　アルバムめくり

このような変更を行ったことで,ユーザがより小さい動きでマウスが動作するように改善することができた。

（４）MMD の概要と結果

MMD とは MikuMikuDance の略で,キャラクタの 3D モデルを操作しコンピュータアニメーションを作成することのできる 3DCG ソフトウェアである。MMD では,キャラクタの関節を動かすことで 3D モデルを動かすことが可能である。KINECT は人体の骨格,関節を認識する機能が優れているため,関節を指定してキャラクタを動かすことができる MMD とはとても相性が良い。このソフトを KINECT で動かすことで KINECT の使い方に慣れるというのが一学期の目標であった。

MMD の 3D モデルには図６のように関節ごとにボーンと呼ばれるものが設定されており,上記でも示したように動かしたいボーンと指定することでモデルを動かすことができる。

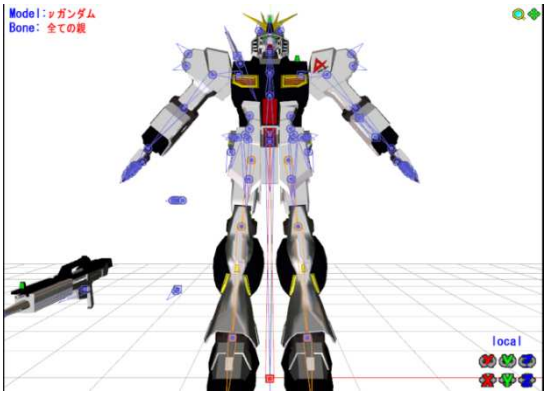

図６　MMD モデル

本来ならば一つひとつボーンを指定し動かさなければならないのだが, KINECT で人を認識させることでモデルの関節と人の関節を対応させることができ,MMD モデルを動かすことができようになっている。図６のモデルはガンダムになっているが,このモデルは自由に変更することができる。研究を行ってく中で空いた時間を利用し,このモデルを作ることにも挑戦した。

MMD モデルを作成するために必要なソフトは「Blender」と呼ばれるソフトである。このソフトは３次元コンピュータグラフィックスソフトウェアの一つで 3D の作成,レンダリングやアニメーション,コンポジット機能も備えている。このソフトを使用し図７のように 3D モデルを作成する。そして関節にボーンを設定することで,KINECT を使用した際にも動かすことができる MMD モデルの完成である。

図７ Blender でのモデル作成

また,MMDにはいくつかのバージョンがありKINECTに対応しないものや,KINECT(Xbox版)に対応のバージョン,KINECT for Windows版に対応しているバージョンがあり,目的にあったバージョンを選ばなければならない。MMDをKINECTで動作させる実験を始めた時に,誤ったバージョンを使用して実験を進めていたためKINECTで動作させるまでに通常の倍以上の時間を要してしまった。
(今回使用したバージョン:Ver.7.39)

３．研究のまとめ

当初の予定ではKINECTを使ったゲームの制作を考えていたが,開発環境などが整っておらず,ゲームを作ることができなかった。それでもなんとか手段はないかと思い,「Unity」というゲーム開発ツールを使ってみたが作成することができなかった。開発環境などを整えるために時間が多くかかってしまったので,サンプルプログラムを参考に「マウス操作」と「アルバムめくり」を作成した。

マウス操作ではマウスポインタが端から端まで動かせるように少しずつプログラムの値を変えるなどして正確に動かせるようにした。

またアルバムめくりではただ画像が変わるだけでは面白くないと思いクラス写真を取り入れることで情報技術科３年の写真が見られるようにした。

４．研究の感想

[小野]

課題研究を通してKINECTについて調べていくと東京大学先端科学技術研究センターと日本マイクロソフトが「Kinect for Windows」を使い重度の障害のある人たちを支援するシステムを開発したということを知りました。KINECTを使いそのような医療にも役立つということを知りさらにKINECTの性能やできることに興味がわきました。私たちはこのようなアプリケーションを制作することはできなかったがもっと時間をかけることでさらに新しいこともできると思う。

[上田]

この課題研究ではKINECTのアプリケーション制作のための C#について学ぶことができ,個人としてはMMDモデルを制作する技術を学ぶことができた。だがKINECTについてまだ深く理解することができなかった。これから時間があるときにでも学んでいきたいと思う。

[井上]

サンプルプルグラムを変更する際には,まずそのプログラムがそのような流れなのかを理解する必要がありました。はじめにプログラムを実行し,改善点を見つける。次にプログラムを読み,よりプログラムを深く理解していった。いくつかのプログラムに触れ,理解し,改善していくことで授業で学ぶプログラミングよりも,より実践的で高度なプログラミング技術を学ぶ事ができました。

[片山]

ゲームを作るためにはOpenNIという開発環境が必要でした。OpenNIを使えるようにするためにいろいろな方法を試しましたが,
よく,調べるとOpenNIはKINECT[Xbox]にしか対応していませんでした。そこでKINECT for Windowsに対応している環境を調べました。使えるのはKinect for Windows SDKだということがわかりました。それを使い無事サンプルプログラムを改良して使いやすいアプリケーションを作ることが出来ました。

５．参考文献

(1)KINECT for Windows SDKプログラミング
C#編　「著者　中村薫」
(2)KINECT センサープログラミング
「著者　中村薫」
(3)KINECTセンサー画像処理プログラミング
「著者　谷尻豊寿」