平成２６年度　卒業論文

KINECT を用いた人物抽出映像の立体視感マッピングによる臨場感の評価

平成 27 年 2 月 16 日

11111044

宮﨑　葵

指導教員　三浦　元喜　准教授

九州工業大学　工学部　総合システム工学科

概要

現在，ライブビューイングというアーティストやパフォーマーによるライブの同時中継が多数行われている。このライブビューイングとは小ライブハウスや劇場のスクリーンにて現地会場の様子を上映するというものである。本研究は，このライブビューイングにおいても現地会場との差をなくそうという考えから行ったものである。そこで今回，Kinect を用いて人物の映像を抽出し立体感を持たせてプロジェクションマッピングすることで，単に映像を投影する場合よりも比較的簡単に現地会場に近い臨場感が認められるのではないかという仮定をたてた。本研究では，その実装への取り組みと実際にプロジェクションマッピングを行った実験の際における周囲の評価を得て考察を行った。

目次

第 1 章　序論

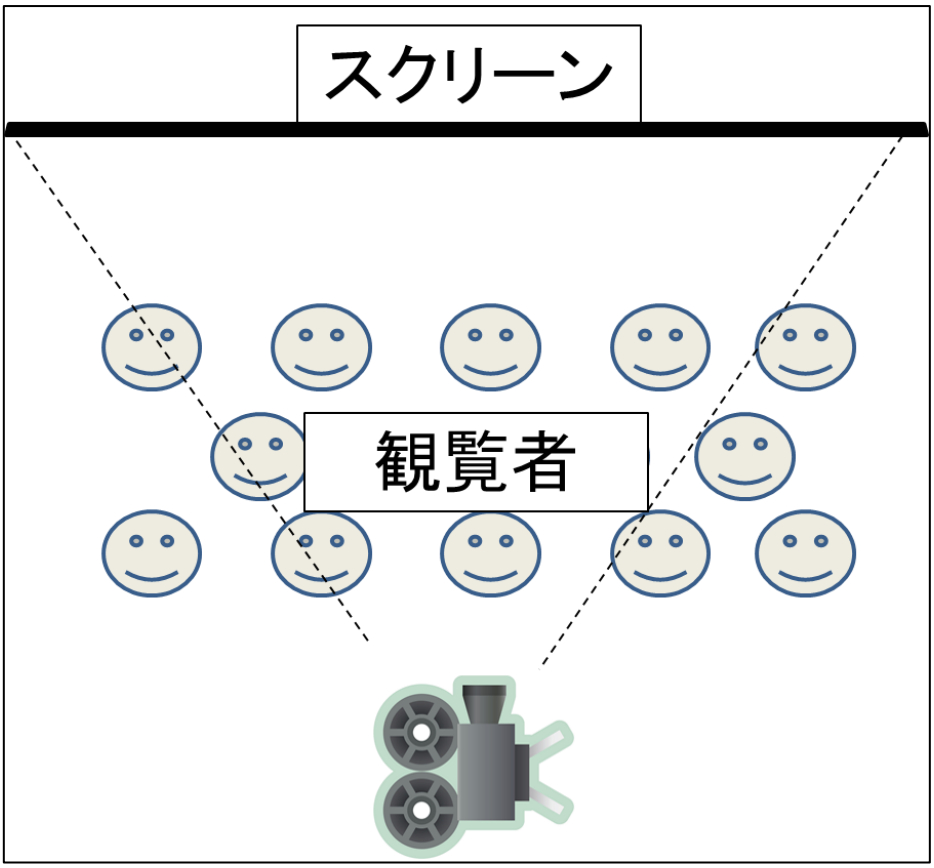
本章では，本研究にいたった背景と関連事項について述べる。

1.1　研究背景

現在，日本には数多くの音楽アーティストやダンスパフォーマーが存在し，その多くは日本各地の主要都市にあるアリーナ，ドーム，スタジアム，コンサートホールにおいてライブコンサート，ライブパフォーマンス(以下ライブ)を行っている。このライブで最も重要視されることは，テレビ放映やラジオ放送の通常放送のように録音，録画によるものではなくリアルタイムによる公演であり実際に足を運び自分の目で見て感じるということである。ライブならではの音響システム，スポット照明，会場との一体感など数値化は難しいが生でしか感じられない特別な感覚を味わうため多くの観客が足を運ぶ。しかし，これらのライブが行われる会場に収容できる人数には限界があり，特設ステージを除くと日産スタジアムの約 7 万人が最大であると言われており，全ての人が観覧に行けることはなく基本的に抽選が行われる。

また，開催地は首都圏で行われることが多く地方に住む人には少なからずハンディがある。このような問題の解決にしばしばライブビューイングが行われる。(図 1.1)ライブビューイングとは本会場のライブの様子を映画館等の上映施設にて生放送する方法で，抽選に落ちてチケット等を確保できなかった人だけでなく，地方に住んでいて会場が遠く行けない人も近場の上映場で観覧できる。このライブビューイングでも会場における公演の臨場感を表現できるよう大がかりな音響設

備を用いているようだが，映像に関しては単に上映だけを行うことが多い。そこで，視覚的にもより本物に近い臨場感を表現できれば，本会場との差異を減らせると考えた。映像による臨場感の表現として 3D つまり立体感が重要であると仮定した。ここでいう立体感とはライブを行う人物と背景を切り離して映像化することである。今回その方法として人物と背景を切り離す操作に Kinect の機能を使用し，映像化する操作にバーチャルアイドル「初音ミク」の 3D ホログラフィによるライブのように立体感をもたせた映像の上映方法を用いる手法を提案する。本研究では，この手法により簡単に低コストで立体感のある映像を実現し臨場感を表現できるかについて調査・評価を行う。



（図 1.1　ライブビューイングのイメージ図）

1.2　プロジェクションマッピングの歴史

プロジェクションマッピングとはコンピュータで生成したコンピュータグラフィック(CG)やカメラで撮影した映像やイメージを建物や車といったオブジェクトの物理表面などにプロジェクタを用いて映写し，オブジェクトの形状や状況に合わせイメージを張り付ける技術のことである。日本では「プロジェクションマッピング」という呼び方が一般的であるが,その他に「3D マッピング」や「ビジュアルマッピング」，また単に「マッピング」と称されることもある。

プロジェクションマッピングの起源はディズニーランドのホーンテッドマンションにおける胸像へのプロジェクションマッピングと言われている。コンピュータ制御の技術がまだ無い 1969 年にアナログ映写機を用いて顔の映像を胸像に投影したという。この技術はディズニーランドに存在する数多くのアトラクションを設計・製作しているウォルトディズニーイマジニアリング(WDI)によってもたらされた。開園当初から愛される「白雪姫と七人の小人」やスリリングなアトラクション「タワー・オブ・テラー」，大人気となった夜のエンターテイメントショー「ワンス・アポン・タイム」などにおいて，このプロジェクションマッピングの技術が惜しみなく用いられている[1]。

日本では 2010 年に行われた映画のフィルムプレミアが初と言われ，長崎ハウステンボスや東京ディズニーランドといった施設での商業利用にも用いられるようになった。現在は普及が進み様々な演出に用いられるほか，プロジェクタの普及が進んだこともあり個人で行う事例

もある。

1.3　バーチャルアイドルと透過スクリーン

バーチャルアイドルとは実在はしない仮想アイドルのことである。バーチャルアイドルと言ってもギャルコンで活躍した商品化を目指す3DCGによるバーチャルアイドル，グラビアアイドルをCG化したバーチャルネットアイドルなどがあるが，本研究で挙げているバーチャルアイドルとは2007年以降に普及を始めた「初音ミク」「鏡音リン・レン」「巡音ルカ」等の音声合成DTM(デスクトップミュージック)ソフトウェアである。(図1.2)これらは音声合成技術VOCALOIDを使用したバーチャルアイドル歌手で独自の音声を持つ人工物として画期的な技術であった。また初音ミクをはじめとするバーチャルアイドルは，非営利を目的としていればほぼ自由な利用を認めているため多くのユーザがこれらのバーチャルアイドルを利用した作品を公開した。普及を始めたDTMは3Dホログラム技術を活用したライブの開催が試みられ，その際に用いられたものがDILADスクリーンと呼ばれる透過スクリーンである。ガラスやアクリルに貼付し観測者に対して正面(スクリーンからして背後)からの投影(リアプロジェクション)を行うことでなにもない空間にバーチャルアイドルが実在するような錯覚を起こさせる。しかし透過スクリーンには拡散性に欠点があり，観測者と透過スクリーンとの間に大きな角度が生まれると観測は難しい。そこで高視野角を実現させるために複数台での同期投影が行われている。これによりDILADボードを用いた映像投影には大きなコストが掛かってしまう。この代替案としてアミッドPこと青木敬士氏が開発，公開し

たのが網戸を用いるアミッドスクリーン，また農業用ポリ袋を用いるポリッドスクリーンである。これらは精度こそ劣るものの低コストで準備できることから個人またその集まりによる研究・制作が盛んに行われるようになりニコニコ技術部でも多くの公開動画が見受けられるようになった。またこれらは動画投稿だけでなく，日本各地でイベントが開催されるようになり技術の伝達・発展が行われている。

(図 1.2　初音ミク[画像引用:初音ミク販売元 CRYPTON FUTURE MEDIA, INC 公式ブログより])[2]

1.4　本論文の構成

本論文において，

・第 2 章では，本研究に用いる手法について考察する。

・第 3 章では，本研究に用いた手法について述べる。

・第 4 章では，本研究に用いたプログラムの構成について述べる。

・第 5 章では，投影実験に基づく結果と考察を行う。

・第 6 章では，本研究と論文についての総括を述べる。

第 2 章　提案手法の考察

本研究に使用する系への採用を検討する際，特に次の 3 つの要素

①Kinect センサー

②透過スクリーン

③プロジェクタ(プロジェクションマッピング)

が重要となった。本章にてこれらの性質について考察を行う。

2.1　Kinect の機能について

本研究における人物検出の要となる Kinect とは，2010 年に Microsoft より発売されたセンサーデバイスである。当初はゲーム機である Xbox 360 端末向けの付属機器「Kinect For Xbox 360」として，ゲーム本体と同時に販売されたが，画期的なセンサーデバイスとしての注目が集まり 2012 年に Windows PC 向けの「Kinect for Windows」として開発者向けに新たにリリースされた。Kinect には RGB 情報を取得できる RGB センサー，センサーからの距離を取得できる深度センサーが扱え，その他にも赤外線カメラやマルチアレイマイクロフォンが搭載されている。これらを組み合わせることでカメラ画像の取得，距離画像の取得，人間の関節の特定，音源位置の特定などが行える。

当研究室で用意されていた Kinect は「Kinect for Xbox 360」であり(図 2.1)，本研究にはこれを用いているが，Windows 用と Xbox 用の違いは主に，

・商用利用の可否

・Xbox への対応

・Near モードの搭載

が挙げられる。商用利用に関しては，Windows 用にのみ認められている。Xbox への対応は Xbox 用のみ対応していて Windows 用の動作には PC が必要である。次に Near モードとは Windows 用から搭載された機能である。深度情報の計測可能範囲はデフォルトで 80cm～4m と設定してあり Windows 用ではさらに 40cm～の計測が可能な Near モードが搭載されている。ただしこのモードでは 6 人同時の人物追跡に HIP_CENTER(腰の位置)しか指定できず身体全体の指定はできない。

センサー情報を用いた PC での開発向けに Kinect for Windows SDK がリリースされており，Windows でのソフトウェア開発の際に用いると非常に便利である。Microsoft 提供のデバイスではあるが，Mac OS でも使用は可能であり，こちらでは OpenNI と NITE，またこれらをより使いやすくまとめた Simple OpenNI を導入する必要がある。ただし，OpenNI と NITE を提供する OpenNI 公式サイト (http://www.openni.org/)が 2014 年 4 月 Apple 社の買収により閉鎖しているため導入の際は注意が必要である。

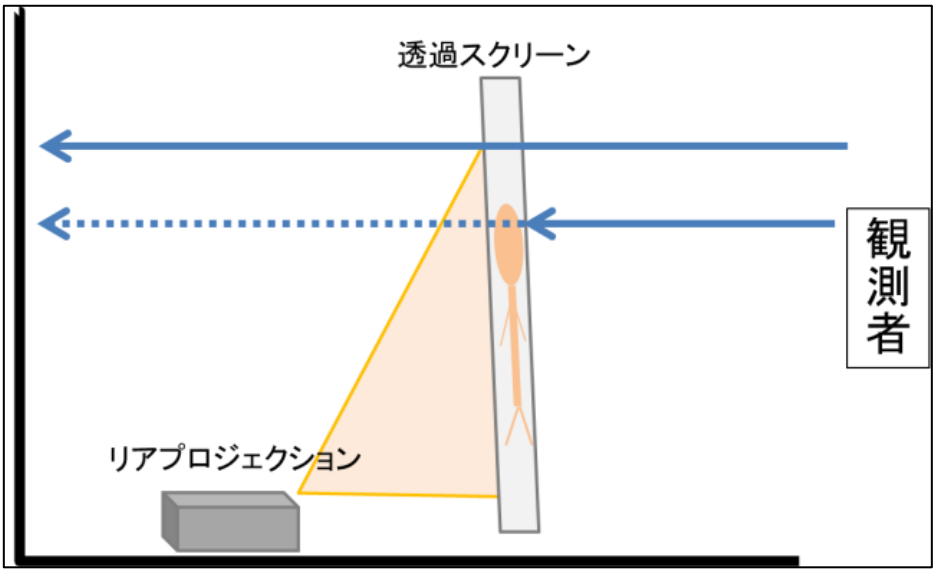
本研究には Mac OS を使用しているため Simple OpenNI を導入し，人物検出に関するプログラムを作成する。またリアルタイムな取得と投影を行うことを目的とするため，従来の人物抽出に用いられたクロマキー法[3]のように特殊な条件を付加したり，事前に背景等を用意し

たりせずに各映像の取得を行えるように処理を行うことを前提とする。

(図 2.1　研究に用いた Kinect for Xbox 360)

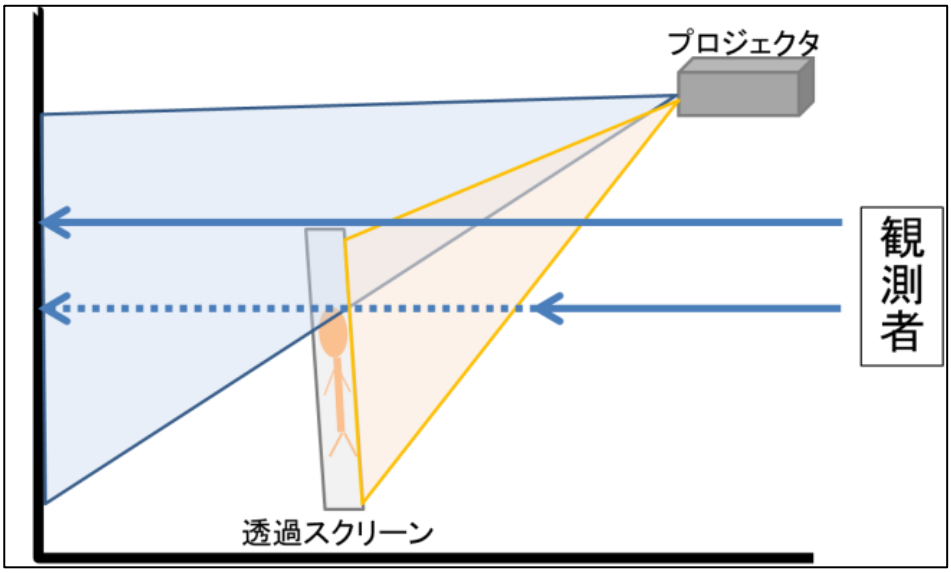
2.2　透過スクリーンについて

透過スクリーンは基本的に，光を通せる透明なガラスやアクリル板に特殊なフィルムを張ることで半透明スクリーンにしたものである。スクリーン背面から左右反転した映像を投影(リアプロジェクション)することで正面から見ると映像と背景が見えるという仕組みである。(図 2.2)



(図 2.2　透過スクリーンへのリアプロジェクション)

この透過スクリーンの特徴として光の透過率と視野角が相反することが挙げられ，透過率を上昇させると視野角が狭くなるため背面は見やすくなるが横からの視認が困難になり，透過率を下降させると視野角が広くなるため背面は見えにくいが広範囲からのスクリーン映像視認が可能になる。この特徴を最大限に生かす為に，使用時の条件に合わせた適切な素材選択や使用するスクリーンの構造を選定する必要がある。本研究ではスクリーンを通して見える背景にも別映像として Kinect で撮影した映像から被写体の背景情報切り離し投影(図 2.3)し，より被写体の実環境に近い状況を作り出すことを想定するため透過性を重視した素材選定を行う必要がある。しかし，ライブビューイングのように，多くの人が広い角度でスクリーンを観測する状況も想定した場合も考えてある程度の視野角の確保も必要である。



(図 2.3　背景と透過スクリーへの 2 面フロントプロジェクション)

透過スクリーンには比較的に安価で自作できるもの(アミッドスクリーン，ポリッドスクリーン)と営利目的時に使用する高価な DILAD スクリーンがある。本研究ではライブビューイング等を想定し，臨場感を得るために人物抽出映像の立体投影をより簡単に実現することが目的であるため，DILAD ボード使用申請を出すほどではなく自作可能な透過スクリーンで十分であると判断する。よってアクリルにポリ素材系のビニールを貼付したポリッドスクリーンか網戸を利用したアミッドスクリーンが良いのではないかと考えた。ポリッドスクリーンの特徴として透過性は非常に高いが視野角が厳しいため，本研究では透過性と視野角の両方を視点に入れたアミッドスクリーンのなかでも網目が大きめのものを用いる。

2.3　プロジェクションマッピングによる映像投影

プロジェクションマッピングはテーマパーク等で華やかな演出を表現するために用いられることが多いが，プロジェクションマッピング自体はすでに広く普及を始めているため，事前に画像や動画を用意していれば個人で簡単に行えることも多い。プロジェクションマッピングも可能なVJソフトやiPadとモバイルプロジェクタを接続してプロジェクションマッピングが行えるアプリ PRSPCTV(URL: http://prspctv.minoaimino.com/?lang=ja)等が公開されているからである。よって単に個人で楽しむ程度のプロジェクションマッピングの実現は簡単であるが,本実験のようにKinectからの取得画像をリアルタイムで処理し，さらにプロジェクションマッピングを行うという特殊な環境では既製品を用いることは難しく余計な手間が増えてしまうと考えられるため，Kinectとの親和性を視野にいれたプロジェクションマッピングプログラムの作成を行う。

プログラム作成に当たってはProcessingというビジュアルプログラミング環境を使用した。これはビジュアルアートに特化したプログラミング言語であり2次元，3次元空間を想定した図形の描画，マウスやタッチ操作で図形を操作する直感性に利があり，コード作成はJavaを基本とした構成で書くことが可能である。またProcessing内でKinectを使用することが可能という特徴もあり，本研究のようにKinectからのイメージ情報の操作，プロジェクションマッピングといった映像の加工と直観的な操作にあたっては最適であると考えた。

本研究では，小さなライブハウスや劇場といった場所でも簡単な方

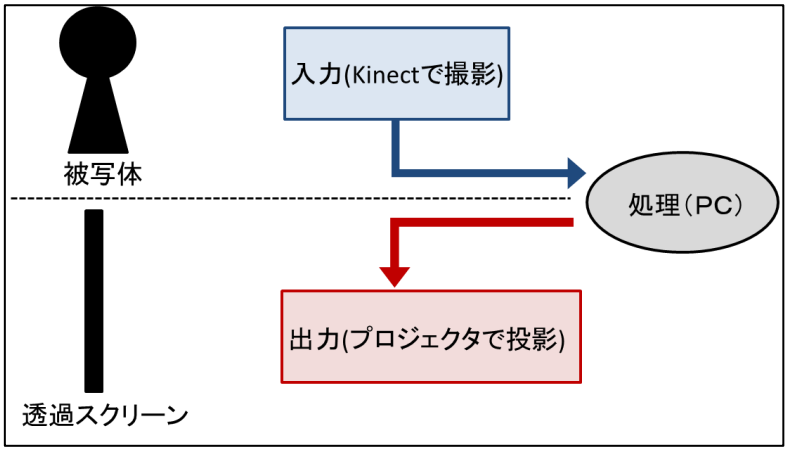
法と操作で映像内の人物抽出とその映像の投影により立体感の再現を行える環境の開発を考えているため，できる限り標準環境内の機能または最低限追加するべきフレームワークで実現できる機能を使用したプログラム構築を目標とする。具体的なプログラムの内容と構成については後述(第 4 章)するが Kinect 導入の際に使用するフレームワーク Simple OpenNI の標準機能と Processing 搭載の関数によって比較的簡単なプログラムコードの作成に至った。

第 3 章　研究に用いた手法の詳細

本章では，研究に使用した装置について述べる。入力側として人物とその背景を含む被写体の Kinect を使用した撮影，出力側としてプロジェクタとアミッドスクリーン(透過スクリーン)を使用したプロジェクションマッピングがある。

3.1　全体の構成

全体の大まかな構成として 3 つに大別できる。(図 3.1)実際に被写体を Kinect のカメラで撮影する入力側，処理した映像をプロジェクタで透過スクリーンと背景に投影しマッピングする出力側，これらの処理を行う PC となっている。また Kinect の性能上約 1m 以上は被写体と距離を置き，全身が収まる必要がある。装置間の接続は有線でありプロジェクションマッピングは目視をしながら行うため，あまり PC との距離をおくことができない点に注意した。



(図 3.1　研究に用いた装置の全体構成)

3.2　入力(撮影)側に用いる手法

第 2 章 1 節でも紹介したが本研究では Kinect for Xbox 360 を使用した。Kinect の機能として複数のカメラセンサー情報(RGB 情報，深度情報，赤外線情報)を使用することで人の骨格を判断できる。(図 3.2) またその領域に人が居ることを判断して処理ができる[4][5]。Kinect 自体は Microsoft 社が開発しているものの，MacOSX 等でも Kinect の操作は可能であり今回はこちらで操作を行った。

(図 3.2 Kinect に搭載されているカメラとセンサー)

また，この Kinect センサーには Windows 版と Xbox 版との違いのほかに Kinect v2 という次世代型のバージョンアップした形態のものも開発されている。今回用いた Kinect for Xbox360 を Kinect v1 とすると，これらの大きな違いは深度情報の読み取り方にある。Kinect v1

では，深度センサーから発光した赤外線を深度カメラで受光することで対象物との距離を測定しているのに対し，Kinect v2 では深度センサーと深度カメラが一体化し発光した赤外線が反射して戻ってくるまでの時間により対象との距離を測定している。その他の異なる点，改善されている点を下表(表 3.1)にまとめた。後述する本研究における改善点のいくつかは，この Kinect v2 を用いることでも改善できるであろう。

(表 3.1 Kinect v1 と Kinect v2 の違い)

	Kinect v1	Kinect v2
カメラ解像度	640×480	1920×1080
人物姿勢検出可能人数	2 人	6 人
人物関節可能検出数	20 関節	25 関節
深度情報の取得範囲	0.8～4.0m	0.5～8.0m

* Kinect v2 では環境光が少ない場合自動的に fps を落とす機能もある。

今回は Kinect v1 ということで,被写体とカメラとの距離は 4.0m 以内にする必要があり,本研究では 2.5m ほどの距離をおいて撮影した。また，投影するアミッドスクリーンの色は灰色であったため被写体の服装が暗い色であったとき投影時に映りづらい，もしくは見えづらいこともあった。そのため，撮影時は比較的明るい服(実験では上着が赤

色のパーカー，下が灰色のジーンズ)を着て撮影を行った。Kinect の設置場所は床面から約 1.0m の距離(実験時は脚立の天板部)のところに設定した。撮影時の環境光が少ない場合，Kinect による RGB カメラ情報がうまく取得できないため撮影側の部屋では電灯を点けて撮影した。(図 3.3)

(図 3.3 Kinect による撮影側(入力側)の撮影時の様子)

3.3　出力(投影)側に用いる手法

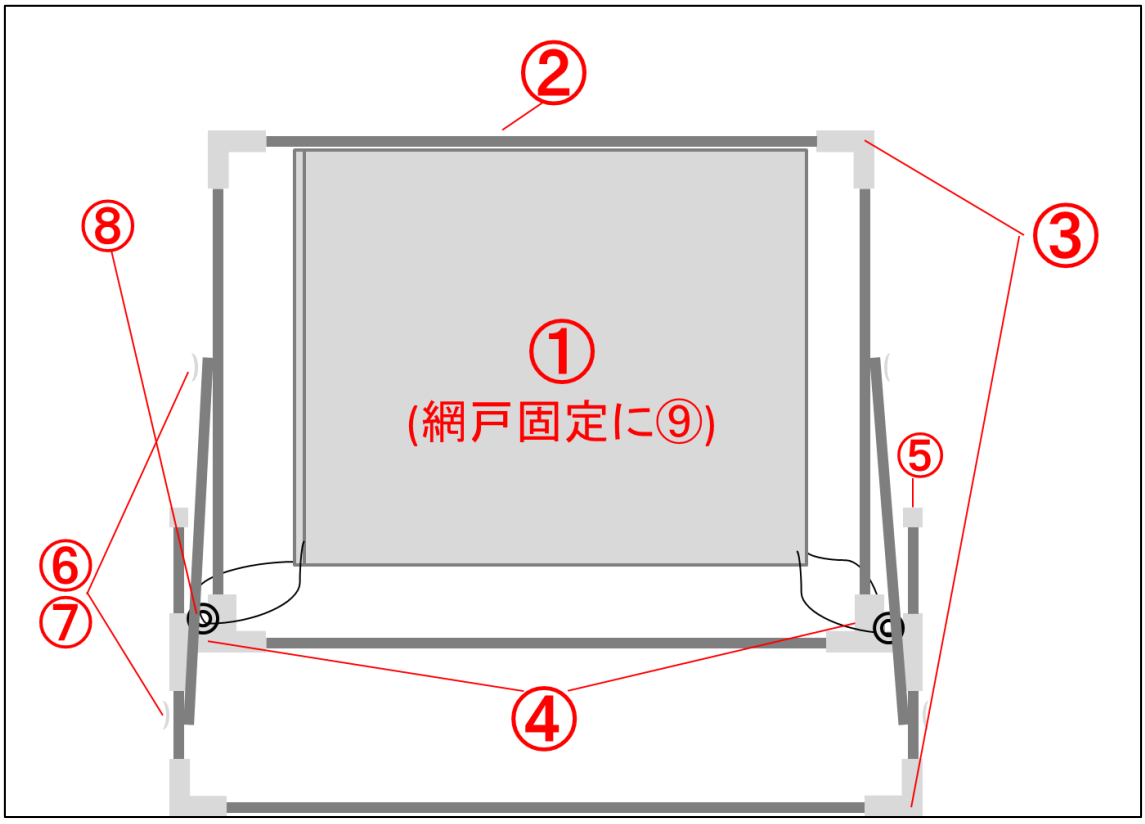
本研究においての出力側とはプロジェクタとアミッドスクリーン(透過スクリーン)を用いたプロジェクションマッピングのことを指す。この透過スクリーンへ人物(多くはバーチャルアイドルといったキャラクタ)を投影するという技術は開発者の青木敬士をはじめ，一般のユーザ(ニコニコ動画ではニコニコ技術部のタグがついたもの)により発展を遂げている。今回用いた手法の参考にしたものは，その中のデスクトップライブステージというものである[6]。この手法ではバーチャルアイドルの動作等を作成する MikuMikuDance(MMD)にてモーションを作成する際，ライブステージ(背景)，バーチャルアイドル(人物)，ステージ演出(床面)をそれぞれ分離し 1 つの動画内でそれぞれが再生される状態にした後，接続した iPad とモバイルプロジェクタを用いて専用アプリでプロジェクションマッピングを行い机上でホログラフィライブを再現するというものである。従来の透過スクリーンを用いたバーチャルアイドルの投影はリアプロジェクションを行いキャラクタだけを空間に投影しキャラクタの存在を表現するといったものに対し，この手法ではフロントプロジェクション(この場合セットに対して上方からの投影)を行うことで透過スクリーンだけでなく背部のセットに対し背景の投影，床面に対しスポットライト等のステージ演出の投影が可能となった。私はこの手法を拝見した際，従来の投影方法よりさらにキャラクタの存在に臨場感を覚えたため，この手法を参考にすることを決めた。この手法を公開している HP では約 30 ㎝四方のセットでの製作であったが，本研究では実際に撮影した人物を投影すると

いう状況のため，プロジェクタの設置位置も加味し透過スクリーンを約 1m の高さになるように製作した。製作に当たってはアミッドスクリーン開発者の青木敬士による製作方法を参考にした。製作に用いた材料はホームセンター等で比較的安価に手に入るものを使用している。材料は下表に示す。(表 3.2)

(表 3.2 透過スクリーン製作材料)

スクリーン部	
①網戸用張替防虫ネット 91cm×2m 18 メッシュ グレイ	1 巻
骨組み部	
②水道用塩化ビニルパイプ(径 13m)　2m	5 本
③TS 継手エルボ(径 13mm)	4 個
④TS 継手ティーズ(径 13mm)	4 個
⑤TS 継手キャップ(径 13mm)	2 個
結合部	
⑥ステンレスなべ小ネジ M4×40	4 本
⑦ステンレス蝶ナット M4	2 個
⑧ステンレスリングキャッチ 3mm	2 個
⑨ロックタイ黒(結束バンド)	1 袋

組み立てのイメージは(図 3.4)にあるが本研究で製作したアミッドスクリーンは，網戸が 2 重になっている点に注意する。



(図 3.4 アミッドスクリーンを背面から見たイメージ)

上図の番号は(表 3.2)内の材料に割り振った番号と対応している。転倒防止のため背面接地部の骨組みを囲う様に組み立て網戸側面のパイプと背面接地部のパイプを斜めのパイプで固定している。

プロジェクタに関してはアミッドスクリーンと背景投影部の両方を含んで光が投影されるようにより高さのある上部に設置し，危機の設定で台形補正が入らないようにすること，光量を強くすることに注意する。

第 4 章　研究に用いたシステムの詳細

本章では,研究に使用した装置の PC における処理について述べる。PC での処理は Kinect からの情報の処理，処理した映像のプロジェクションマッピングを一個のプロジェクト内で一括して行う。プログラム詳細は本論文付録に記載する。

4.1　開発環境

本研究において処理に用いた PC の詳細は下表である。

(表 4.1 研究に用いた PC の詳細)

機　種	MacBook Air
プロセッサ	Intel Core i5 1.7GHz
メモリ	4GB 1600MHz
ソフトウェア	OSX 10.9.5
内蔵ディスプレイ	11 インチ(1366×768)
USB	USB3.0 Hi-Speed/SuperSpeed パス

Kinect のソフトウェア要件ではメモリが 2GB(推奨は 4GB)，USB2.0 端子での接続となっているのでこの点は問題ない。

次に，実際に処理に関するプログラムの作成を行った開発環境であるが Processing を用いた。Processing とは MIT メディアラボによって開発されたオープンソースである。Java を単純化しグラフィック機能に特化した言語を用いていて，電子アートやビジュアルデザインの

ための統合開発環境である。Processing で作成されたコードはブラウザ上での動作，Android 端末や iOS 端末での動作も可能である。本研究では Processing2.2.1 を導入した。

Processing で今回のコードを作成するにあたり導入したライブラリは，Processing 上 Kinect を操作するための 3 種類である。

①OpenNI

②NITE

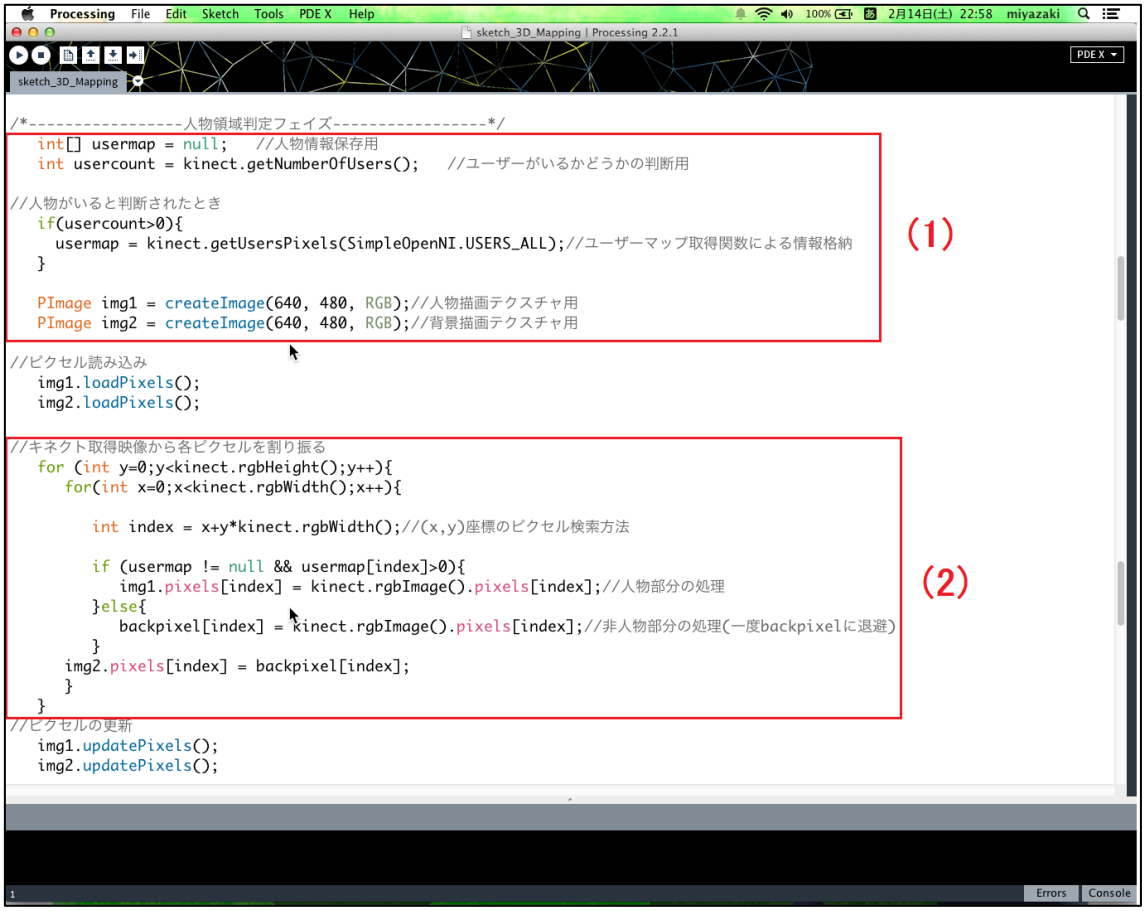
③Simple-OpenNI

①, ②は Kinect からのデータを取得したり人物の姿勢を認識したりする際に使用し, ③は Processing で OpenNI を使えるようにするために使用する。またこれらは，バージョンアップに伴い内部関数の改変等が行われているためそれぞれバージョンをそろえて用いることが望ましい。本研究では，①～③それぞれ OSX 用のバージョン 0.20 にあたるバージョンのものを使用した。(現在①，②に関しては公開 HP の閉鎖に伴い導入する際には自己責任となる)

4.2　人物抽出のプログラム

本節では，本研究で作成したプログラムコード内の人物検出と検出領域の処理について述べる。この処理はプログラム内における「人物判定フェイズ」にあたる。(図 4.1)

このフェイズの中でも人物検出に関する部分(図中(1))と検出領域の処理(図中(2))に分けられ，以下これらについて説明する。



(図 4.1 プログラム内人物検出フェイズ抜粋)

(1)人物抽出について

人物抽出についての処理方法は，OpenNI の機能で行うことができる。ただし，この方法はバージョン 0.20 における機能に準拠しており，バージョンが上がると

.getUsersPixels( )

関数内において，

SimpleOpenNI.USERS_ALL ------ 人の全身骨格

を指定することができない。これは，上記定数がバージョンアップに伴い削除されているためであり，これ以外の頭部，肩部，肘部等といった部分指定なら可能である。この関数を用いる際はプログラム冒頭の初期設定時に，

.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);

を記載し，骨格検出(全身)を有効化しなければならない。

(2)検出領域の処理について

(1)で検出した人物領域とそれ以外の背景の領域の処理を行う。Processing ではイメージを扱う際 PImage という特殊な型を用いるが，人物情報を保存するものと背景情報を保存するものを 2 つ用意する。PImage で用意されたイメージはピクセル単位で操作可能で，

.loadPixels() ---------- ピクセルの読み込み

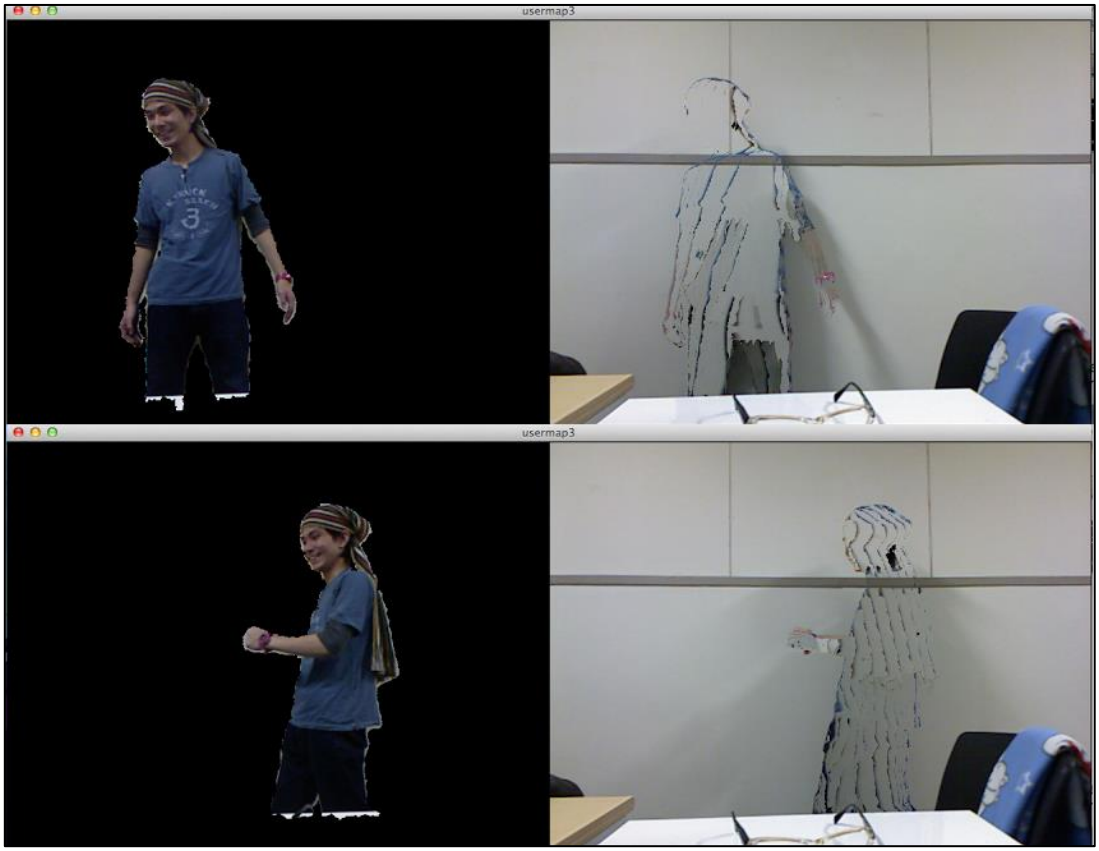
.updatePixels()---------- ピクセルの更新

を使用することでピクセル情報の読み込みと更新を行う。Kinect からの RGB 情報やピクセルのカラー情報は 1 行の配列として渡されるため，任意座標のピクセル検索は

x+y*kinect.rgbWidth() --------- (x,y)座標のピクセル

を用いて行う。人物領域と背景領域の条件分岐は(1)で渡された usermap を用い，ピクセル単位で判断と処理をそれぞれ行う。人物領域の処理は，その領域に該当するピクセルをそれぞれ img1 に格納する。背景領域の処理は，人物がいる場所の背景を補填するために backpixel[]という色情報を一時的に保存するカラー配列を用意し，1

度その配列に RGB 情報の該当ピクセルカラーを保存して，改めて img2 に格納を行う。これにより RGB 情報の各ピクセルは背景領域と判定されるたびに backpixel[]に保存されることになり，人物領域にあたるピクセルは直前のカラー情報を読み出すことにより背景として表示することができる。下図は実際に Kinect で撮影した映像を人物領域と背景領域に分離して表示した様子である。(図 4.2)



(図 4.2 人物領域と背景領域の分離表示 上：正面からの撮影 下：姿勢変更時)

上図の通り，今回の人物検出は Kinect の標準機能のみで行っており検出精度は Kinect に依存するためノイズがあることがわかる。人物検出に関しては上図下部のように姿勢を変えた場合も検出可能なことがわかる。

4.3　プロジェクションマッピングのプログラム

本節ではプログラム内のプロジェクションマッピングに関する処理について述べる。この処理はプログラム内における「2 面マッピング設定フェイズ」及び「テクスチャ頂点操作フェイズ」にあたる。(図 4.3)

sketch_3D_Mapping | Processing 2.2.1

```
/*--------------2面マッピングフェイズ-------------------*/
   noStroke();//縁なしを設定

// img1(人物側)をテクスチャとして設定し頂点指定
   beginShape();
   texture(img1);
      vertex(pos[0][0],pos[0][1], 0, 0); // 左上頂点指定
      vertex(pos[1][0],pos[1][1], img1.width, 0); // 右上頂点指定
      vertex(pos[2][0],pos[2][1], img1.width, img1.height); // 右下頂点指定
      vertex(pos[3][0],pos[3][1], 0,img1.height); // 左下頂点指定
   endShape(CLOSE);
// img2(非人物側)をテクスチャとして設定し頂点指定
   beginShape();
   texture(img2);
      vertex(pos[4][0],pos[4][1], 0, 0); // 左上頂点指定
      vertex(pos[5][0],pos[5][1], img2.width, 0); // 右上頂点指定
      vertex(pos[6][0],pos[6][1], img2.width, img2.height); // 右下頂点指定
      vertex(pos[7][0],pos[7][1], 0,img2.height); // 左下頂点指定
   endShape(CLOSE);
```

sketch_3D_Mapping | Processing 2.2.1

```
/*-----------------頂点操作フェイズ----------------------*/
//ある点がクリックされていたら
   if( mousePressed && selected >= 0 )
   {
      pos[selected][0] = mouseX;
      pos[selected][1] = mouseY;
      ellipse( mouseX, mouseY, 20, 20 );//選択円の表示
   }
//頂点がクリックされていないとき
   else
   {
      minD = 25; //基準吸着距離（２５）
      selected = -1; //選択頂点の初期化
      for (int i=0; i<8; i++) {
         D = dist( mouseX, mouseY, pos[i][0], pos[i][1] );  //ポインタ位置と頂点の距離を格納
//ある頂点とポインタが基準距離より近いとき
         if ( D < minD )
         {
            minD = D;
            selected = i;
            ellipse( mouseX, mouseY, 20, 20 );//選択円の表示
         }
      }
   }


}
```

(図 4.3 プログラム内 2 面マッピング設定/テクスチャ頂点操作フェイズ抜粋)

2 面マッピング設定について

本研究において Processing を用いてプロジェクションマッピングを行う際，OpenGL の機能であるテクスチャ(texture)を用いた。現行バージョンの Processing では OpenGL ライブラリを特に指定しなくても使うことができる仕様となっている。テクスチャとして画像イメージや動画映像を設定すると，テクスチャ内イメージを動的に操作可能となる。テクスチャの設定は

beginShape()　～　endShape()

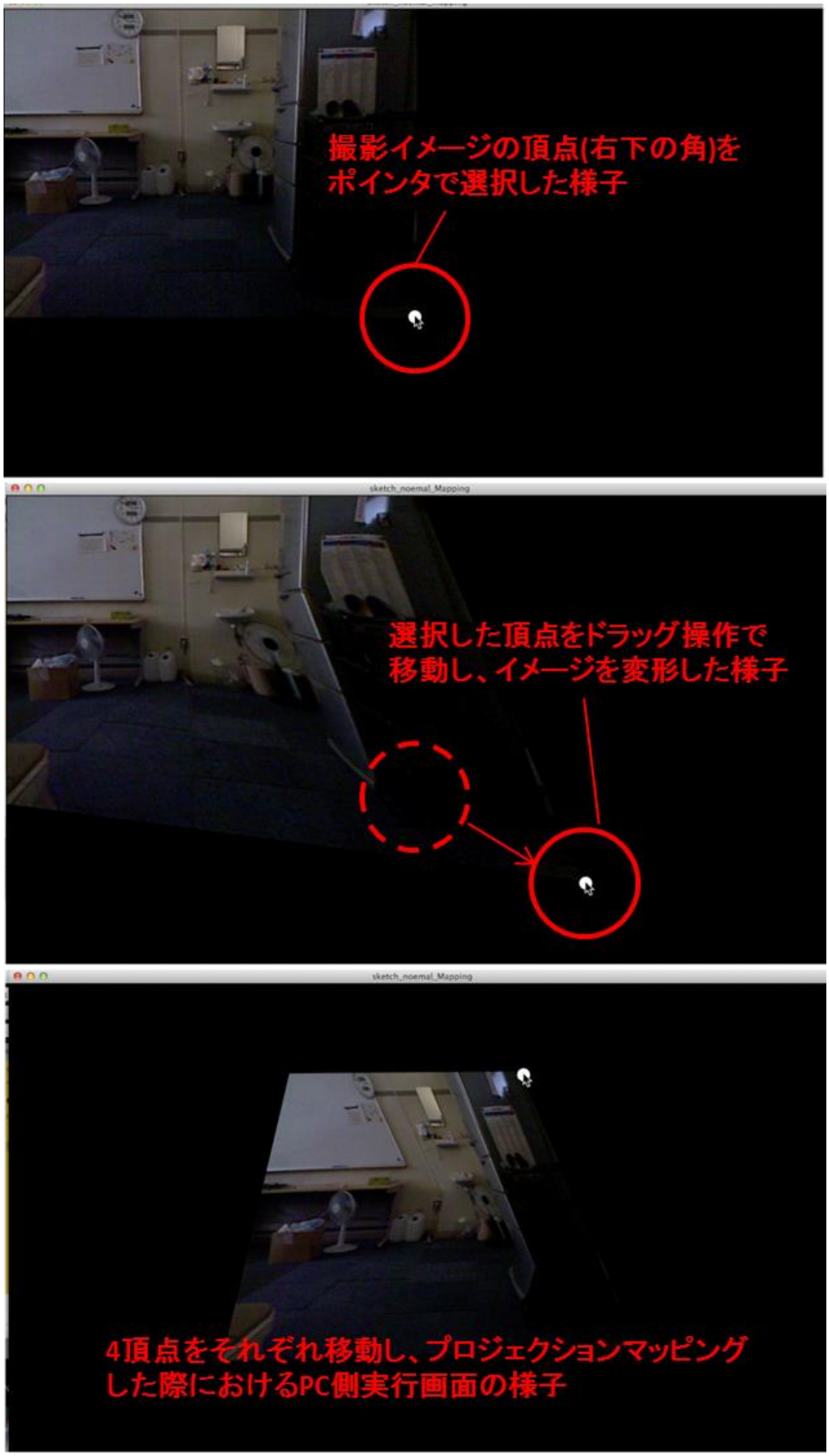
内で行い，vertex によりテクスチャ頂点を指定し形状を決定する。vertex では，テクスチャとして設定するイメージ内の頂点として設定したいピクセル座標とそれに該当するテクスチャの座標を対応させる。これによりテクスチャの頂点に操作を加えたとき元のイメージ内でも同様の操作が行われ動的な変形等が可能となる。本研究では，Kinect から取得する 640×480 のイメージ(ただし，前工程で人物領域と背景領域に分離した 2 つのイメージ)をそれぞれプロジェクションマッピングするため，img1 と img2 それぞれのテクスチャを設定した。

テクスチャ頂点操作について

本研究におけるプロジェクションマッピングは直観的な操作をめざし，実際の投影状況を目視しながら mac 搭載のトラックパッドにてドラッグ操作を行う構成にした。このフェイズでは大まかにクリックをしている場合とクリックをしていない場合の 2 つの状況におけるポインタ情報を使用している。本研究で作成したプログラムでは冒頭においてテクスチャの頂点情報を格納する配列を用意しているため，クリ

ックをしている場合の処理はポインタの座標と頂点の座標を置き換えるという単純なものである。クリックをしていない場合は，マウスポインタと各頂点(本研究では 2 つのイメージの合計 8 つの頂点)との距離を計算し，その距離が一定の距離内にあるとき次の処理に移る。この一定の距離を吸着距離とし，本研究では 25 と設定した。この距離の値が大きくなれば，マウスポインタがより遠くの場合でも頂点付近に近づいたと判断されるようになるため，実際の投影状況等に合わせて変更する必要がある。吸着距離内にある場合の処理は，ポインタとの距離を吸着距離に設定し，選択頂点を 1 つの頂点に特定している。また，クリックしている場合もクリックしていない場合も頂点に近づいた際にはポインタの位置に円の表示を行い視覚的に認識しやすくし，操作性の向上を行っている。

実際にこれらの用いたプロジェクションマッピングを行っている際の実行画面を(図 4.4)に示す。



（図 4.4 プロジェクションマッピング時の実行画面）

第 5 章　実験

本章では，前章までで製作した装置・作成したプログラムを実際に組み合わせて撮影と投影を行った実験について述べる。

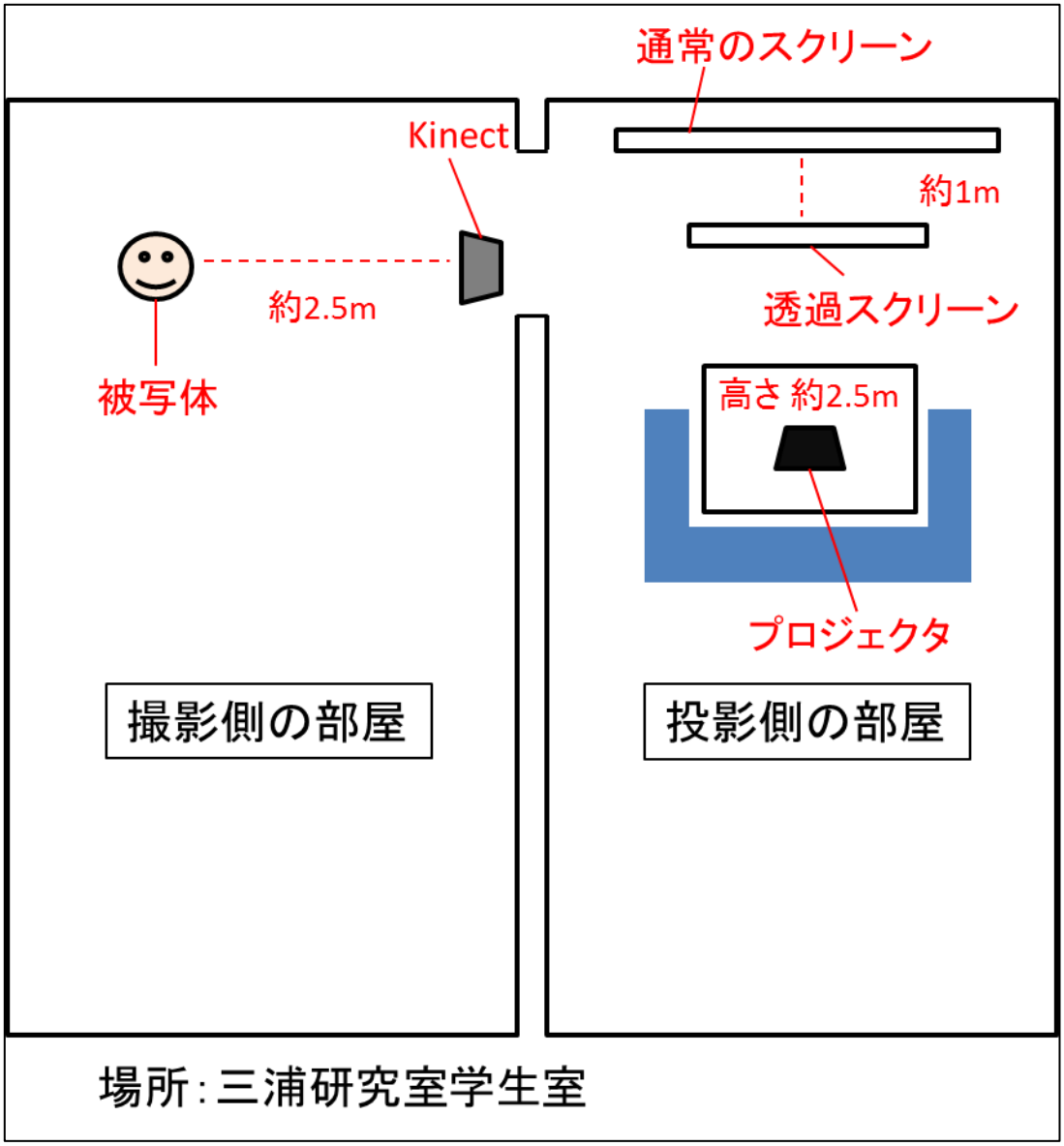
5.1　実験概要

本研究の前提として，臨場感を得るためにはデスクトップライブステージのように映像内の人物と背景を分離しそれぞれを投影することで立体感を持たせれば良いと仮定した。また，小ライブハウス等において大がかりな装置，セットを用いなくても簡単に実現できることを目標とした実装を行った。よって投影環境は室内と想定する。

実験時の環境条件は，

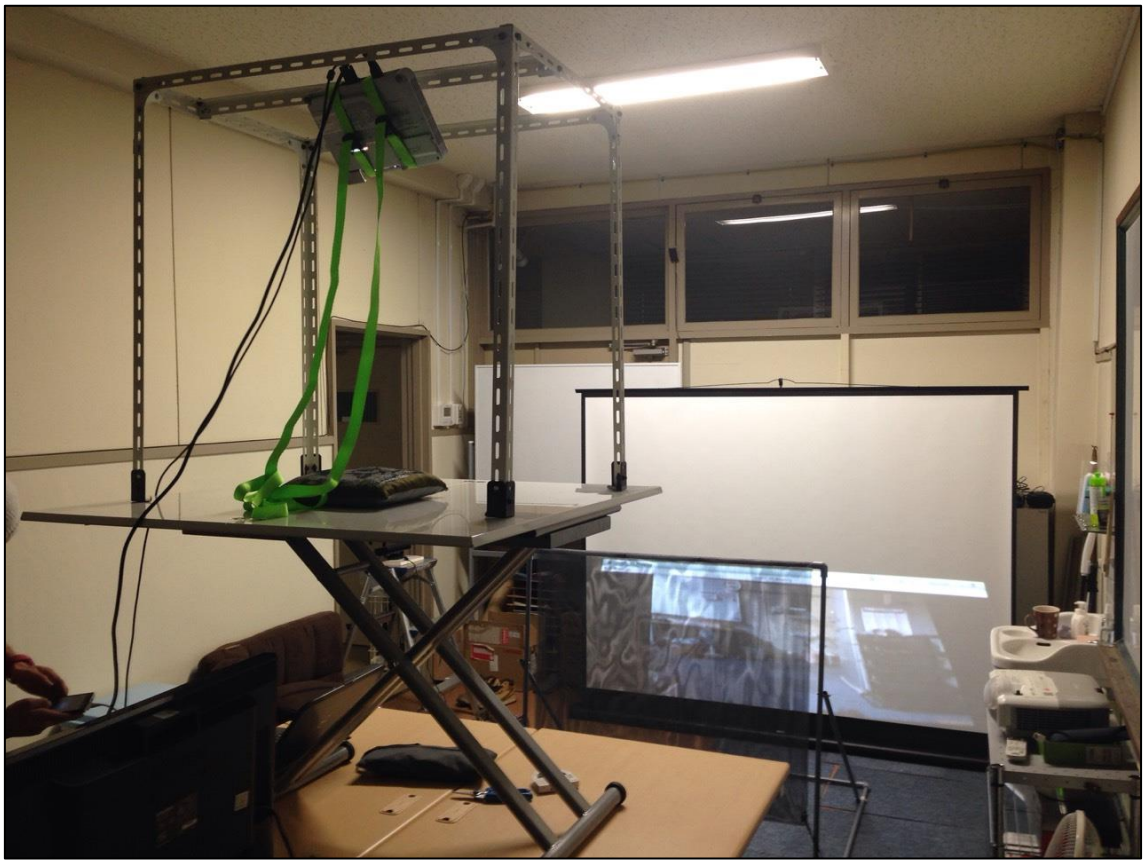
- 撮影側は RGB 映像の撮影を行うため照明で明るさを確保した上で，人物とその背景が映るように注意しながら Kinect から約 1m～4m の範囲に人物が待機する。
- 投影側はプロジェクタを用いるため部屋を暗くした上でプロジェクタを部屋の上部に設置し，投影光がアミッドスクリーンと背景投影部の両方を含むように調整する。この時プロジェクタの光量を強く設定し台形補正等が入らないように注意する。

とする。またプロジェクションマッピングを行う際は投影状況を目視しながら各イメージの頂点を調整していく。実験時における設置状況の概略図を(図 5.1)に，設置した装置の実際の様子を(図 5.2)に示す。



(図 5.1 実験実施時における室内環境の概略図)

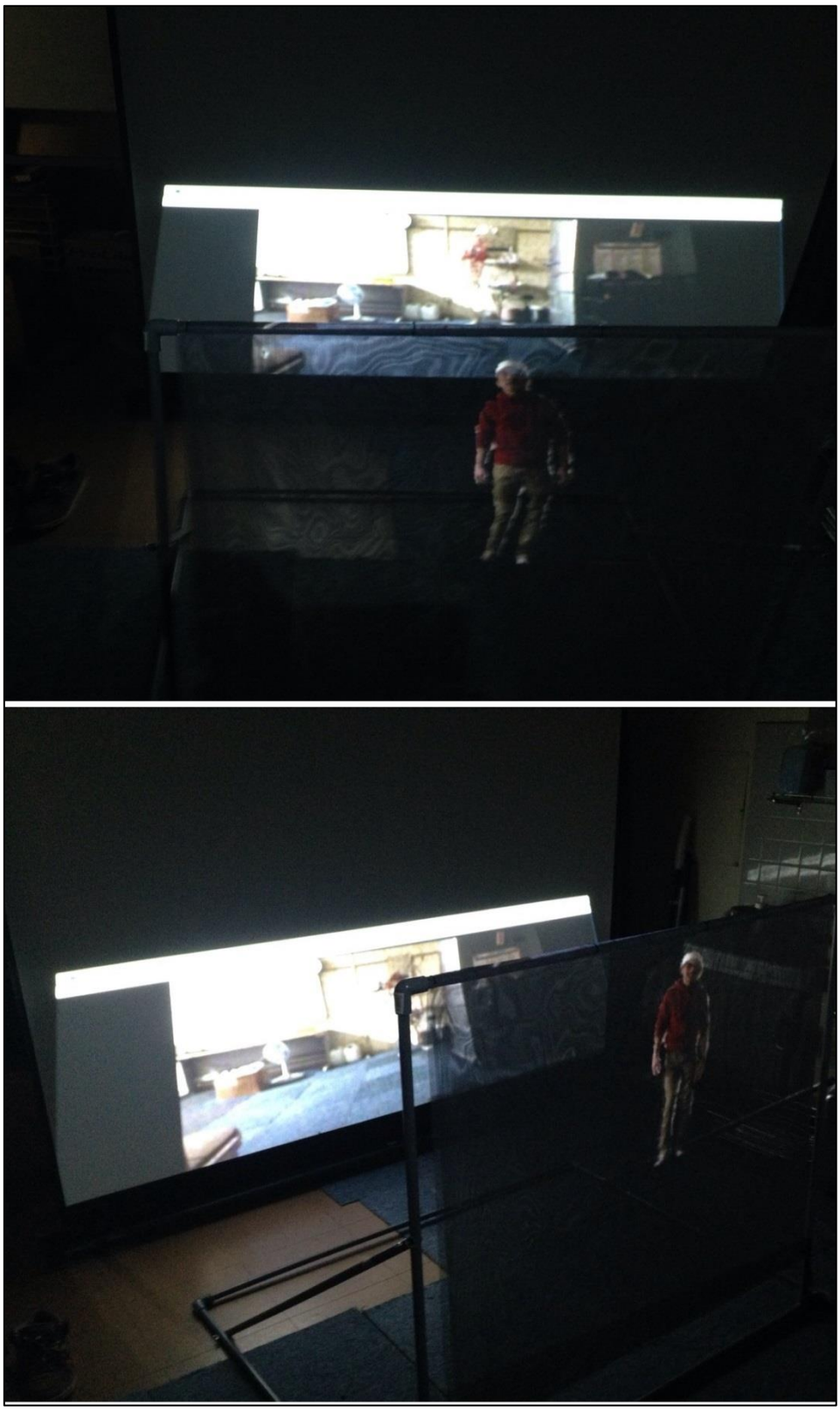
投影，撮影の部屋を分け，被写体と Kinect の距離を約 2.5m，プロジェクタを床から約 2.5m，透過スクリーンと背景投影用の通常のスクリーンを約 1m の距離とした。観測は(図 5.5)内の青領域でおこなった。



(図 5.2 設置した装置の実際の様子)

5.2　結果

今回の実験には同研究室所属 22 歳～28 歳の学生 4 人に被験者として協力してもらった。被験者は全員九州工業大学所属の男性である。実験方法は，単純にカメラ映像を投影した様子と本研究の手法を用いたプロジェクションマッピングによる投影の様子の 2 つのパターンをプロジェクションマッピングするところから見てもらいコメントをもらうというものである。実験時のアミッドスクリーンにプロジェクションマッピングしている様子を(図 5.3)に示す。



(図 5.3 実験時の様子 上：視聴領域正面側 下：視聴領域側面側)

実験の結果は，本研究の手法を用いた抽出映像のプロジェクションマッピング自体は成功したが，問題点も多く被験者となってもらった方々からのコメントでも同問題点を指摘された。コメントを以下に述べる。

- プロジェクションマッピングにおける画像の位置合わせ等の簡単さは伝わった。
- 立体的になっているということは伝わったが，臨場感と言われると印象が薄かった。
- 人物抽出精度におけるノイズが目立っていた。
- 真正面から見たとき背景映像の光によって手前の人物が見えづらかった。
- デスクトップライブステージを単純に等身大に近いセットサイズに適応したことは正かったのか。
- そもそもデスクトップライブステージによって臨場感が得られるという仮定を実証する予備実験が必要だったのではないか。
- 実行時のタイトルバーやシステムバーが気になる。

5.3　改善案

上記した問題点を解決するためにはコメントでも指摘があったように，そもそも臨場感が得られるという確証を得るための事前実験を実施する必要があったと感じる。また，それに付随してより立体感を表現できる投影方法(例：プロジェクタの適正な高さ，スクリーンの配置

位置，プロジェクタの投影光量)を割り出し，本研究手法に用いる際の適正な方法を考える必要があった。

人物検出精度については，本研究手法では簡単な導入方法を想定し可能な限り標準機能を用いて行うために Kinect 性能に依存した検出を行っていたので，精度を確保するには Kinect の深度情報を用いた人物及び境界の特定[7]を併用したり，既存の境界・特徴点検出アルゴリズム(SIFT や SURF など)を用いた人物抽出にしたりする必要がある。もしくは Kinect v2 といった高機能なセンサーに切り替えるという方法も考えられる。また本研究では背景領域に床の映像も含まれてしまうためアミッドスクリーンに投影される人物が浮いて見えてしまうという問題点もあったため，本来のデスクトップライブステージのように人物領域，背景領域，床面領域といった 3 面プロジェクションマッピングにする必要性があると感じた。この為にはやはり，Kinect の機能だけの検出ではなく，境界を検出する他のアルゴリズムを導入し背景と床面も切り離す必要がある。

第 6 章　結論

本研究ではライブビューイングのように人物の映像を投影する状況を想定し，撮影地さながらの臨場感を表現するための立体感を再現するために Kinect を用いた人物映像の抽出とそのプロジェクションマッピングという手法を提案した。

研究に関するシステムの開発にはビジュアル操作に強い Processing を用いて Kinect からの情報の操作やプロジェクションマッピングを行ったが，今回のように複雑な処理を標準機能だけで再現するのは厳しいことがわかった。またデスクトップライブステージでは 3 面プロジェクションマッピングだったのに対し，安易に 2 面プロジェクションマッピングにしたことで立体感ではなく浮遊感が生まれてしまった。やはりデスクトップライブステージの手法に従い 3 面プロジェクションマッピングにするため，人物検出だけでなく境界検出による人物，背景，床面の切り離しが必要だと感じた。

また研究背景としてこの手法により臨場感が得られると恣意的に判断し仮定していたが，この過程を立証するものはなくこれについての研究が事前に必要ではないかと思う。

この様な分野は VOCALOID や MMD を使用する一般ユーザによる発展を遂げているためより活発な研究・開発が行われることを期待する。

謝辞

本研究にあたり，研究内容・進行法など，様々な面でご指導してくださりました三浦准教授に，この場を借りて心より御礼申し上げます。加えて，研究期間中に御教授していただいた情報セクションの先生方，先輩方に御礼申し上げます。また，研究室の方々を始めとする，実験に協力していただいた方々に，重ねて御礼申し上げます。

参考文献

[1] Mine, M. R., van Baar, J., Grundhöfer, A., Rose, D., & Yang, B. (2012). Projection-Based Augmented Reality in Disney Theme Parks. IEEE Computer, 45(7), 32-40.

[2] 初音ミク公式ブログ,(記事投稿日時)2015/02/10/ 20:40
http://blog.piapro.net/2015/02/u1502101_1.html
(アクセス日) 2015/02/13

[3] 佐藤雄隆, 金子俊一, 丹羽義典, & 山本和彦. (2003). Radial reach filter (RRF) によるロバストな物体検出. 電子情報通信学会論文誌 D, 86(5), 616-624.

[4]橋本直 (2012) AR プログラミング―Processing でつくる拡張現実感のレシピ― オーム社

[5]中村薫 (2011) KINECT センサープログラミング 秀和システム

[6] はつねみくみく 出た！プロジェクションマッピング利用のお手軽デスクトップライブステージ, (記事投稿日時)2013/03/01 20:25
http://vocaloid.blog120.fc2.com/blog-entry-14596.html
(アクセス日時) 2014/04/16

[7]Xia, L., Chen, C. C., & Aggarwal, J. K. (2011, June). Human detection using depth information by kinect. In Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on (pp. 15-22). IEEE.

付録

付録として，事件に用いた人物抽出映像のプロジェクションマッピングに関するプログラムコードと撮影した映像を単純にプロジェクションマッピングするプログラムコードを添付する。

```
/*****************************************************************/
//Title:人物抽出映像と背景マッピング　プログラムコード
//
//Content：２映像テクスチャの頂点(選択時円表示)をドラッグして映像を貼付ける
//
//Name : Myazaki Aoi
//Date:2015/2/12
/*****************************************************************/
import SimpleOpenNI.*;//ver 0.20

SimpleOpenNI kinect;
color[] backpixel = null;//背景保存用色配列

/*-------マッピング用頂点移動要素定義-------*/
int    selected = -1;  // 選択されている頂点代入変数(頂点は 0～7,-1 は非選択状態)
int    pos[][] =
{{0,0},{640,0},{640,480},{0,480},{640,50},{1280,50},{1280,530},{640,530}};// 初期頂点ピクセル情報座標配列
float minD;//基準吸着距離変数
float D;//マウスポインタと頂点距離格納変数

void setup(){

/*---------------初期設定フェイズ----------------*/
    size(1280,1024,P3D);//P3D 有効

    kinect = new SimpleOpenNI(this);
    kinect.enableRGB(); //RGB カメラ有効化
    kinect.enableDepth();//深度カメラ有効化
    kinect.alternativeViewPointDepthToImage();//2カメラの視点一致
    kinect.enableUser(SimpleOpenNI.SKEL_PROFILE_ALL);//骨格検出有効化(全身判定)
```

```
    backpixel = new color[kinect.rgbWidth()*kinect.rgbHeight()];

}

void draw(){

    background(0);//画面初期化　0=黒背景
    kinect.update();//キネクトカメラの更新

//image(kinect.rgbImage(),0,0);//テスト表示用

/*-----------------人物領域判定フェイズ-----------------*/
    int[] usermap = null;    //人物情報保存用
    int usercount = kinect.getNumberOfUsers();    //ユーザがいるかどうかの判断用

//人物がいると判断されたとき
    if(usercount>0){
      usermap = kinect.getUsersPixels(SimpleOpenNI.USERS_ALL);//ユーザマップ取得関数による情報格納
    }

    PImage img1 = createImage(640, 480, RGB);//人物描画テクスチャ用
    PImage img2 = createImage(640, 480, RGB);//背景描画テクスチャ用

//ピクセル読み込み
    img1.loadPixels();
    img2.loadPixels();

//キネクト取得映像から各ピクセルを割り振る
    for (int y=0;y<kinect.rgbHeight();y++){
        for(int x=0;x<kinect.rgbWidth();x++){

            int index = x+y*kinect.rgbWidth();//(x,y)座標のピクセル検索方法
```

```
            if (usermap != null && usermap[index]>0){
                img1.pixels[index] = kinect.rgbImage().pixels[index];//人物部分の
処理
            }else{
                backpixel[index] = kinect.rgbImage().pixels[index];//非人物部分の
処理(一度 backpixel に退避)
            }
        img2.pixels[index] = backpixel[index];
        }
    }
//ピクセルの更新
    img1.updatePixels();
    img2.updatePixels();

/*--------------2 面マッピング設定フェイズ------------------*/
    noStroke();//縁なしを設定

// img1(人物側)をテクスチャとして設定し頂点指定
    beginShape();
    texture(img1);
        vertex(pos[0][0],pos[0][1], 0, 0); // 左上頂点指定
        vertex(pos[1][0],pos[1][1], img1.width, 0); // 右上頂点指定
        vertex(pos[2][0],pos[2][1], img1.width, img1.height); // 右下頂点指定
        vertex(pos[3][0],pos[3][1], 0,img1.height); // 左下頂点指定
    endShape(CLOSE);
// img2(非人物側)をテクスチャとして設定し頂点指定
    beginShape();
    texture(img2);
        vertex(pos[4][0],pos[4][1], 0, 0); // 左上頂点指定
        vertex(pos[5][0],pos[5][1], img2.width, 0); // 右上頂点指定
        vertex(pos[6][0],pos[6][1], img2.width, img2.height); // 右下頂点指定
        vertex(pos[7][0],pos[7][1], 0,img2.height); // 左下頂点指定
    endShape(CLOSE);
```

```
/*------------------テクスチャ頂点操作フェイズ----------------------*/
//ある点がクリックされていたら
    if( mousePressed && selected >= 0 )
    {
        pos[selected][0] = mouseX;
        pos[selected][1] = mouseY;
        ellipse( mouseX, mouseY, 20, 20 );//選択円の表示
    }
//頂点がクリックされていないとき
    else
    {
        minD = 25; //基準吸着距離(25)
        selected = -1; //選択頂点の初期化
        for (int i=0; i<8; i++) {
            D = dist( mouseX, mouseY, pos[i][0], pos[i][1] );  //ポインタ位置と頂
点の距離を格納
//ある頂点とポインタが基準距離より近いとき
            if ( D < minD )
            {
                minD = D;
                selected = i;
                ellipse( mouseX, mouseY, 20, 20 );//選択円の表示
            }
        }
    }

}
```

```
/*****************************************************************
**/
//Title:キネクト取得画像(RGB 画像)マッピング　プログラムコード
//
//Content:テクスチャの頂点(選択時円表示)をドラッグして映像を貼付ける
//
//Name:Myazaki Aoi
//Date:2015/2/12
/*****************************************************************
**/

import SimpleOpenNI.*;//ver 0.20

SimpleOpenNI kinect;


/*----マッピング用頂点移動要素定義----*/
int    selected = -1;  // 選択されている頂点変数(頂点は 0～3)
int    pos[][] = {{0,0},{640,0},{640,480},{0,480}};// 初期頂点ピクセル情報座標配
列
float minD;//基準吸着距離変数
float D;//ポインタと頂点距離格納変数


void setup(){

/*---------------初期設定フェイズ----------------*/
   size(1280,1024,P3D);//P3D 有効

   kinect = new SimpleOpenNI(this);
   kinect.enableRGB(); //RGB カメラ有効化
   kinect.enableDepth();//深度カメラ有効化
   kinect.alternativeViewPointDepthToImage();//2カメラの視点一致
}
```

```
void draw(){

    background(0);//画面初期化
    kinect.update();//キネクトカメラの更新

//image(kinect.rgbImage(),0,0);//テスト表示用


    PImage img1 = createImage(640, 480, RGB);//キネクト RGB 画像格納用

    img1.loadPixels();
//各ピクセルをコピー
    for (int y=0;y<kinect.rgbHeight();y++){
        for(int x=0;x<kinect.rgbWidth();x++){

            int index = x+y*kinect.rgbWidth();//(x,y)座標のピクセル検索方法
            img1.pixels[index] = kinect.rgbImage().pixels[index];//人物部分の処
理

        }
    }

    img1.updatePixels();

/*---------------マッピングフェイズ------------------*/
    noStroke();
    beginShape();
    texture(img1); // img1(RGB 画像)をテクスチャとして設定
        vertex(pos[0][0],pos[0][1], 0, 0); // 左上頂点指定
        vertex(pos[1][0],pos[1][1], img1.width, 0); // 右上頂点指定
        vertex(pos[2][0],pos[2][1], img1.width, img1.height); // 右下頂点指定
        vertex(pos[3][0],pos[3][1], 0,img1.height); // 左下頂点指定
    endShape(CLOSE);
```

```
/*-----------------頂点操作フェイズ----------------------*/
//ある点がクリックされていたら
    if( mousePressed && selected >= 0 ){
        pos[selected][0] = mouseX;
        pos[selected][1] = mouseY;
        ellipse( mouseX, mouseY, 20, 20 );//選択円の表示

//頂点がクリックされていないとき
    }else{
        minD = 25; //基準吸着距離(25)
        selected = -1; //選択頂点の初期化
        for (int i=0; i<4; i++) {
            D = dist( mouseX, mouseY, pos[i][0], pos[i][1] );  //ポインタ位置と頂
点の距離を格納
//頂点選択を判断
            if ( D < minD ) {
                minD = D;
                selected = i;
                ellipse( mouseX, mouseY, 20, 20 );//選択円の表示
            }
        }
    }


}
```