卒業論文

サポートベクターマシンを用いた電極材料の欠陥検出における最適なパラメータの検討

F114037　**樋口　昂平**

指導教員　永田寅臣　教授

2021 **年** 3 **月**

山口東京理科大学 工学部 機械工学科

概要

本稿では、製造業が抱える品質管理に関する課題を解決するために、SVM を設計出来るアプリケーションをもとにシステムを設計し簡易的な評価を行うことでシステムの性能を評価する．z のどの方向に配置されているかがすぐわかるようにし，トラッキング (マウス操作による描画領域の回転，拡大縮小平行移動) に関する処理を行うクラスの作成やピッキング (要素選択) を行うクラスの作成を行った．つぎに描画要素の ON/OFF，マウス操作の ON/OFF，ピック要素 (点，線，面) の ON/OFF の機能を持つモデルビューワレンダラクラスをピック OpenGL レンダラクラスの派生クラスとして生成した．その後，CLS データの座標データを数値として取り込む処理を行うクラスの作成を行い，CLS ファイルに含まれている法線ベクトルも同時に描画できるようにした．これにより，ロボットアーム先端の工具の姿勢変化を視覚的にシミュレーションできるようになった．また，近年では Kinect をはじめ安価で手に入る Depth センサが発売されてきており，3D 点群を意味するポイントクラウドの撮影ができるようになっている．本研究ではポイントクラウドをハンドリングできる PCL(Point Cloud Library) と Depth センサを組み合わせることで，3D プリンターで利用可能な STL データを生成させ，簡単なリバースエンジニアリングの実現方法を紹介する．

目次

第 1 章

緒言

多くの課題を抱えている．例えば，作業員による目視検査である．

工業製品の品質に対する要求が社会的に高まっている中, 工場の生産ラインでは品質管理に関する~~多くの課題を抱えており, 品質改善のため様々な手法が検討されている. そのため, 様々な特徴の欠陥をもつ製品を検出するために経験を積んだ作業者によって, 各工程で製品の品質に関わる検査が行われている. しかしながら~~, これらの検査の良否判断は作業者の主観や経験に基づく判断に委ねられており~~その判断結果が必ずしも妥当とは言えず, 人為的な判断ミスや~~判定基準の曖昧さ~~などが問題視される~~事も多い~~. そこで我々は人間の視覚システム行う作業をAI によって自動化する研究を行っている. コンピュータを活用した欠陥検出は従来より研究されており, レーザーやカメラなどの外観検査装置や主成分分析などといった手法をうまく応用する事で不良品の判別がされてきた. ~~その中でも外観検査は, 製品の欠陥を検出する上で有力な手法であり, 信頼性の高い製品を製造する上で欠かせない作業となっている. この AI のシステムを生産ラインに取り入れる事で労働力不足の解消や生産効率のを向上させることが可能になる.~~

が

参考文献

例えば, 大井らは微細化が進んでいる電子部品を SVM を使って欠陥を自動識別するシステムを検討している. 製造現場から得られる少数の欠陥画像から特徴を抽出した画像を大量に生成することで欠陥検出の精度を高めているようである. また, ~~構造物の維持管理にも利用されており~~, 広兼らはコンクリート構造物の健全性評価を SVM を用いて自動化するシステムを開発を試みている.

本研究では製造業が抱える品質管理に関する課題を解決する為に AI を用いて製造ラインになどで発生する多種多様な欠陥をより高い認識率で不良品と検出するための基本システムを設計し, 簡単な識別検査を行うことでシステムの基本システムを評価する. また, 研究室では Python や C++ などの専門知識がなくても SVM の設計と訓練を効率的に行うことができるツールを提案する。このツールを用いて 1 クラス学習による SVM を設計し, 良品と不良品のテスト画像を用いて分類実験により,SVM の性能を評価する.

第 2 章

2.1 人工知能

鋳造製造の砂型を製造するメーカーは，砂型のマスター型として発泡スチロール型や木型を使用している．型メーカーでは NC 工作機械が使われているが，NC 工作機械は主軸ヘッド部が大きいため被加工物と干渉するなど動作自由度が低く，機械自体も高価という問題点がある．リンク機構で NC 工作機械より動作自由度が高く低コストの産業用ロボットでの加工を行っている．産業用ロボットは CLS ファイルにより動かしている．CLS ファイルは cutter location source の略で CAM 内での工具の位置情報である．従来では加工するときにモデルを再確認しようとするとその都度 CAM に戻って加工パスの再計算を行う必要があった．また研究室にある NC ビューワには工具の姿勢を表す法線ベクトルを表示させる機能がなかったため，今回作成した CLS ビューワには法線ベクトルを表示させえる機能も追加した．また，そのプログラムを基に吉本らが開発した STL-CLS コンバータを使用る前に再度 STL データを確認できる STL ファイルビューワの作成も行った．図 2.1 に従来の加工環境と提案する CLS ビューワ，STL ビューワを示す．

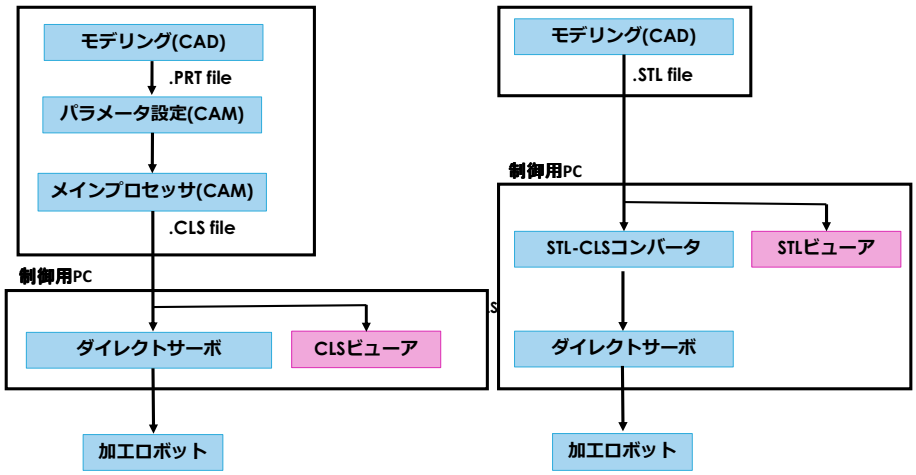

図 2.1. 従来の加工プロセスと産業用ロボットを用いた加工プロセスの比較と，提案する CLS ビューワ．

2.2 機械学習

ソフトウェアは図 2.2 に示すように Visual Studio 2013 Cmmunity C++ 上で Win32 プロジェクトのフレームワークと OpenGL の関数を組み合わせて開発した．OpenGL(Graphics Library) は Khronos グループ [11] が策定している OS に依存しないグラフィックス・ハードウェアのアプリケーションインタフェースである．このインタフェースは数百にも及ぶ豊富な関数群備えており，3 次元オブジェクトの高品質なカラー画像の作成に必要なオブジェクトやクラスを提供する．これにより，プログラマは」オブジェクトや操作を指定することができ，高品質な 2 次元，3 次元オブジェクトのオブジェクトカラー画像を生成することが出来る．OprnGL は種々のモードやシェーダプログラムに従ってプリミティブを描画する．各モードは独立に変更可能で，あるモードの設定はほかのモードの設定に影響を及ぼさない．モードの設定やプリミティブの指定，他の OpenGL の操作は，関数や手続き呼び出しの形で記述されるコマンドを送信することで行われる．

今回のビューワ作成では表 2.1 に示す関数を用いた [4]．glColor3f(GLfloat red, GLfloat green, GLfloat blue) は引数に赤，緑，青の引数をとりそれぞれ 0.0 1.0 の範囲で指定する．glBegin(primitive) は引数に描画したいプリミティブを指定することが出来る．glLineWidth(GLfloat width) は glBegin(const GLfloat *v) で描画した線の太さを指定する．glVertex3fv( ) は描画するモデルの頂点の指定を行う．

図 2.2. CLS ビューワの開発環境

表 2.1. 使用した OpenGL の関数一覧

関数名	機能
glColor3f(GLfloat red, GLfloat green, GLfloat blue)	赤，緑，青の強さを 0.0〜0.1 の範囲で指定する
glBegin(primitive)	プリミティブの指定を行う
glLineWidth(GLfloat width)	描画する線の太さを変更する
glVertex3fv(const GLfloat *v)	3 次元の座標値の指定

2.3　SVM

OpenGL では，頂点属性の集合を指定することで描画コマンドを用いて描画される．点や線，ポリゴンなど様々な幾何学オブジェクトを描画することが出来る．OpenGL のプリミティブによる描画の違いを図 2.3 と表 2.2 に示す．各頂点は 1 つ以上の頂点属性で指定される．頂点属性は頂点シェーダから参照され，後のパイプラインのステージで利用される様々な値を計算するのに利用される．頂点属性とは描画するポリゴンの頂点に定義されている座標，法線ベクトル，色などの属性のことである．頂点シェーダは頂点を処理し，頂点の同時座標系での位置を計算し，頂点シェーダが明示的に書き込む出力用の varying 変数に出力する．図 2.4 に頂点列からプリミティブ (点，線分，ポリゴン) の組み立てを行う処理の流れを示す． 頂点シェーダは 4 成分からなる頂点属性を格納する配列にアクセスする．頂点配列が有効ではない場合，現在設定されている頂点属性の値が頂点属性の値となる．頂点データはサーバーのアドレス空間にある配列にロードされる．これらの配列内のデータブロックは GL のコマンドの実行を通して，複数の幾何学プリミティブを指定するのに用いられる．クライアントは最大で MAX_VERTEX_ATTRIBS 個の配列を用いて，1 つ以上の頂点属性を格納することが出来る．次のコマンド void VertexAttribPointer(uint index, int size, enum type, boolean normalized, sizei stride, constvoid *pointer); はこれらの配列の場所や構成を指定するのに用いる．type は配列に格納される値のデータ型を指定する．size は配列に格納される頂点ごとのデータ数とその成分の順序を指定する．表 2.3 に size や type に指定できる値を

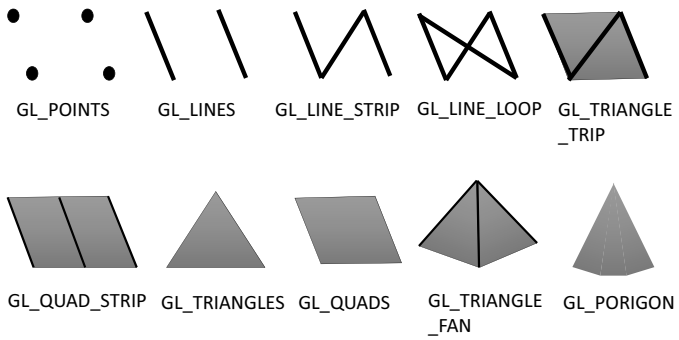

図 2.3. プリミティブ一覧

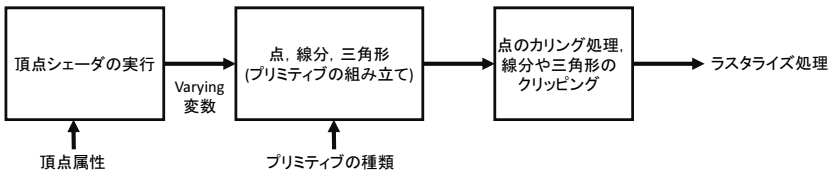

図 2.4. 頂点の処理とプリミティブ組み立て

示しそれぞれ BYTE, SHORT, INT, FLOAT, HALF_FLOAT, DOUBLE の場合，それぞれが byte, short, int, float, half,double 型であることを示す．type の値 UNSIGNED_BYTE, UNSIGNED_SHORT, UNSIGNED_INT の場合，それぞれ ubyte, ushort, uint であることを示す．VertexAttribPointer と VertexAttribIPointer の引数 index は，頂点属性配列を指定する．1 つの頂点に対応する配列の中の 1 つから 4 つの値は，頂点配列の 1 つの要素を構成する．size が BGRA のとき，値が 4 つであることを示す．各配列の要素内の値はメモリ内に順番に格納さ BGR ただし BGR ただし size が BGRA の場合は，配列中の各要素の 1 番目，2 番目，3 番目，4 番目の値は，それぞれメモリ上の 3 番目，2 番目，1 番目，4 番目の値から取得される．頂点シェーダは，頂点の値と頂点に関するデータの処理方法を記述する．頂点シェーダは，処理される各頂点に対して行われる処理方法を記述したソースコードを含む文字配列である．頂点シェーダで用いる言語は OpenGL シェーダ記述言語仕様書で説明されている．頂点シェーダを用いるには，最初にシェーダのソースコードをシェーダオブジェクトにロードし，コンパイルを行う．次に 1 つ以上の頂点シェーダオブジェクトがプログラムオブジェクトにアタッチされる．次にプログラムオブジェクトにアタッチされたすべてのコンパイル後シェーダオブジェクトをリンクすることで，実行コードが生成される．リンク後のプログラムオブジェクトを，カレントのプログラムオブジェクトとして設定することで，そのプログラムオブジェクト持つ頂点シェーダの実行コードが頂点を処理するのに用いられる．シェーダオブジェクトはプログラマブルなステージで実行されるプログラムのソースコードである．プ

表 2.2. 各プリミティブの働き

関数名	機能
GL_POINTS	各頂点を単独の点として扱う
GL_LINES	2 つの頂点をペアとし，それぞれのペアを独立した線分として扱う
GL_LINE_STRIP	描画する線の太さを変更する
GL_LINE_LOOP	すべての頂点を線分で連結する
GL_TRIANGLE_STRIP	一辺を共有しながら帯状に三角形を描く
GL_QUAD_STRIP	一辺を共有しながら帯状に四角形を描く
GL_TRIANGLES	3 点を組にして三角形を描く
GL_QUADS	4 点を組みにして四角形を描く
GL_TRIANGLE_FAN	一辺を共有しながら扇状に三角形を描く
GL_POLYGON	凸多角形を描く

表 2.3. 頂点のサイズとデータ型

コマンド	サイズと成分の順番	整数の扱い	型
VertexAttribPointer	1, 2, 3, 4,BGRA	フラグ	byte, ubyte, short, ushort, int,uint, float, half, double,packed
VertexAttribIPointer	1, 2, 3, 4,	整数	byte, ubyte, short, ushort, int uint

ログラムオブジェクトは GL のプログラマブルなステージで用いられるシェーダオブジェクトのことを言う．頂点シェーダは実行するとき，いくつかの変数を参照する．attribute 変数は頂点ごとに持つ値，uniform 変数はプログラムオブジェクト毎の変数で，このプログラムオブジェクトの実行中にはあまり値は変化しない．サンプラは uniform 変数の特殊な形であり，テクスチャ処理で用いられる．vering 変数は頂点シェーダ実行結果を保存し，パイプラインの続くステージで処理される．カリング処理とは描画する必要のないポリゴンの描画を行わないことで，この処理を行うことにより OpenGL における描画コストを減らすことが出来る．後ろ向きのポリゴンを描画しない背面カリング，カメラの視錐台内に存在しないモデルを描画しないようにする視錐台カリング，不透明モデルによって見えない位置にあるモデルの描画を行わないオクル-ジョンカリングなどの種類がある．ポリゴンの表裏は頂点インデックスをたどる順番により決められる．反時計回りに頂点インデックスを指定すると表になり，時計回りに頂点インデックスを指定していくと裏になる．クリッピングとは画像の指定した一部分を切り抜く処理である．カメラ座標に投影行列を行い得られる座標系をクリッピング座標系という．返還後のクリッピング座標系では同時座標を用いた視体積内外の判定が容易になるように射影行列を決定する．ラスタライズ処理はプリミティブを 2 次元画像に変換する処理のことである．プリミティブのラスタライズ処理は 2 パートある．1 つ目はウィンドウ座標系の整数格子のどの正方形がプリミティブによっておおわれているか決める処理である．2 つ目はデプス値と一つ以上の色の値を各正方形に割り当てる処理である．

2.4　a

OpenGL ではモデルデータにビュ-ポート変換を行って 3 次元モデルをディスプレイ上の 2 次元面の描画領域に写像する．モデルの頂点位置を画面上の位置に変換するバーテックスシェーダから出力された図形の描画領域からはみ出た部分はクリッピングによって削り取られる．したがってバーテックスシェーダは画面に表示する時，クリッピング空間に収まるように投影変換する必要がある．この変換には一般的に平行投影変換と透視投影変換の 2 種類がある．図 2.5 に示すように，透視投影変換では視点は特定の位置に定め投影面からの距離によって見かけの大きさが変わる．平行投影変換では物体の各点などの要素ごとに多数の視点を持ち投影面の距離によって大きさを変えて描画する必要がない．透視投影と平行投影による図形の表示の違いを図 2.6 に示した．

図 2.6 のように透視投影では遠近感のある画像を得ることが出来る．平行投影は遠近感をつかむことは難しいが図の大きさを変えて描画していないので図の大きさを比べるとき時に役立つ．今回作成したビューワでは透視投影変換を用いて描画した．

2.5　CLS ファイルと STL ファイル

次に表示対象である CLS ファイルについて述べる．CLS ファイルは cutter location source の略称で CAM 内での工具の位置情報を格納している．CLS ファイルは以下のように各行

GOTO 文の後に，そのモデルが作成されたときに設定されたワーク座標系における座標値 X, Y, Z 座標の値と正規化された法線ベクトルが記述されている．

GOTO/0.998,0.496,25.774,-0.460,-0.420,0.781
GOTO/1.925,0.496,26.290,-0.418,-0.429,0.800
GOTO/2.881,0.496,26.757,-0.373,-0.437,0.818

Win32 プロジェクトで作成した CLS ビューワは 7 個のソースファイルと，10 個のヘッダーファイルで構成されており，プログラムの核となる ModelViewer.cpp を中心に残りの 6 個のソースファイルがそれぞれの処理を行っている．STL とは 3D SYSTEMS によって開発された三角パッチの集合体としてモデルを表現するシステムで，データの構造が簡単であるため特にラピッドプロトタイピングシステム [5] の標準フォーマットとなっている．CAD/CAM ソフトや photoshop，3D スキャナーなどで生成でき，3D プリンタなど多くのソフトが対応している．STL ファイルのデータ形式には ASCII 形式とバイナリ形式の 2 種類があり，生成されるファイルサイズの点ではバイナリ形式が有利である．例として図 2.7 には ASCII 形式の STL ファイルに含まれる三角パッチのブロックの構成を図 2.8 にバイナリ形式の STL ファイルの構成を示す．ASCII 形式のブロック内には，三角パッチの各頂点の位置座標と法線ベク

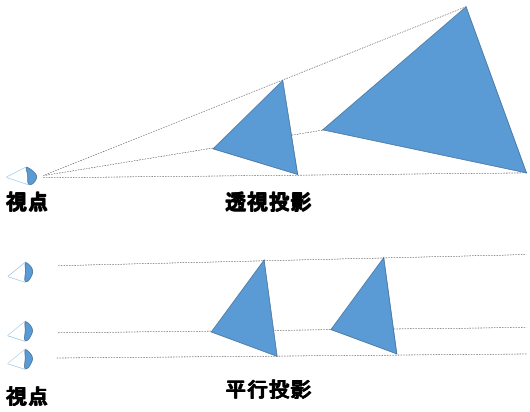

図 2.5. 透視投影と平行投影の視点の違い

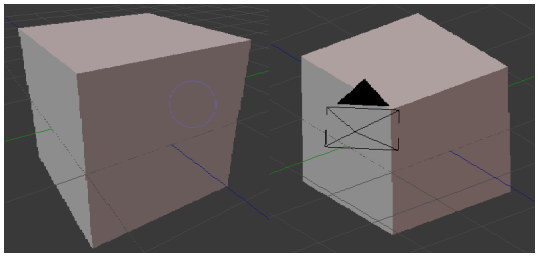

図 2.6. 透視投影と平行投影

トルが記述されている．

2.6　CLS ファイルビューワ

CLS ファイルはテキスト形式であるため，fgets( ) 関数を使用して 1 行ずつリードした．fgets( ) はテキストファイル内の各行の改行コードまでを読み込む関数である．位置・姿勢情報は “GOTO” コードを含む行のみに記述されているため，関数 _stricmp( ) 関数により対象となる行を抽出している．なお，“GOTO” コードを抽出する際にはその数をカウントし，総行数を確認できるようにしている．全ての行の位置・姿勢情報を格納するための領域を宣言し，そこへ格納する処理を行った．線の描画処理について説明する．具体的には格納されている位置情報に対して線と点の描画を行う．まず，線の描画は glVertex3fv( ) により連続する 2 つの位置座標を指定後に，glBegin(GL_LINES) をコールすることで行う．IndexVertex には CLS データの行数がカウントされており，for 文により全ての位置情報に対してこの処理を適用することで，その CLS ファイルが構成する曲面を表現できる．また，各位置情報を点で明

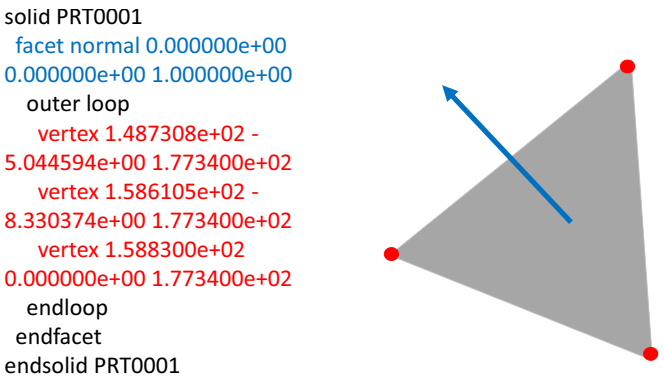

図 2.7. ASCII 形式の STL の構成

バイト数	データ型	データ内容
80	char	任意の文字列
4	unsingned int	三角形の枚数
4, 4, 4	float	三角形の法線ベクトルのx, y, z成分
4, 4, 4	float	三角形1点目の頂点のx, y, z成分
4, 4, 4	float	三角形2点目の頂点のx, y, z成分
4, 4, 4	float	三角形3点目の頂点のx, y, z成分
2	float	未使用データ
⋮	⋮	⋮

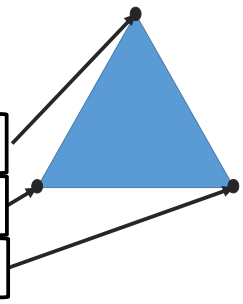

図 2.8. バイナリ形式の STL の構成

確に示すために glBegin(GL_POINTS) 関数で点を描画させている．

```
//線の描画処理
for(counter=0; counter<IndexVertex; counter++)
 {
  glBegin(GL_LINES);
  glVertex3fv((float*)&m_model.Vertex[counter]);
  glVertex3fv((float*)&m_model.Vertex[counter+1]);
  glEnd( );
 }
```

次にベクトルデータの表示方法について述べる．ワーク座標系における座標値 X, Y, Z 座標の値の次に記述されている正規化された法線ベクトルの値を参照して法線ベクトルの描画を行う．法線ベクトルの位置座標の値と CLS データの位置座標の値を glVertex3fv( ) により指定後，glBegin(GL_LINES) をコールして描画を行った．

```
//線の描画処理
for(counter=0; counter<IndexVertex; counter++)
 {
  glBegin(GL_LINES);
  glVertex3fv((float*)&m_model.Vector[counter]);
  glVertex3fv((float*)&m_model.Vertex[counter+1]);
  glEnd( );
 }
```

CLS ビューワの動作を確認するために描画実験を行った．図 2.9 と図 2.10 には CLS ビューワによる表示結果を示す．CLS ファイルに格納されていたワーク座標系における位置ベクトル $\boldsymbol{p}(i) = [p_x(i)\ p_y(i)\ p_z(i)]^T$ を青い点と黒い線で表し，正規化された法線ベクトル $\boldsymbol{n}(i) = [n_x(i)\ n_y(i)\ n_z(i)]^T$ を赤い線で描画している．ただし，i は CLS ファイル内の行番号を表す．

2.7 STL ファイルビューワ

まず，表示処理の前にファイルを読み込むための領域を確保しなければならない．ASCII 形式の場合 CLS と同様に _stricmp( ) 関数を用いて，一つの三角パッチに必ず含まれている facet の行だけを抽出することで三角パッチの総数を知ることができる．一つの三角パッチには図 2.7 に示すように法線ベクトルと 3 つの頂点の座標が格納されているため，それらをもとに必要となる領域の宣言を行った．次に，各三角パッチの位置情報と姿勢情報がそれぞれ vertex を含む行と facet を含む行に記述されているので，それぞれ _stricmp( ) 関数によって抽出することができる．全ての三角パッチから抽出した位置と姿勢の情報はさきほど確保した領域に格納した．バイナリ形式では図 2.8 に示すようにまず任意の文字列が 80 バイトがあ

り，その次に三角形の数を記述した行が存在するのでその行を参照してファイルを読み込む領域を read( ) 関数を用いて確保した．次に三角形の法線ベクトルの値と三角形の各頂点座標を read( ) 関数で読み込みさきほど確保した領域に格納した．

次に具体的な表示方法について述べる．STL ビューワが行う線の描画処理のプログラムの一部下に示す．

```
//線の描画処理
    for(IndexTriangle=0;IndexTriangle
      m_model.vIndexedTriangle.size();Triangle++)
    {
       for(tri3;tri3¡3;tri3++)
      {
        glBegin(GL_LINES);
        unsigned int IndexVertex0=m_model.vTriangle[Triangle]
        .Index[tri3];
        unsigned int IndexVertex0=m_model.vTriangle[Triangle]
        .Index[(tri3+1)%3];
```

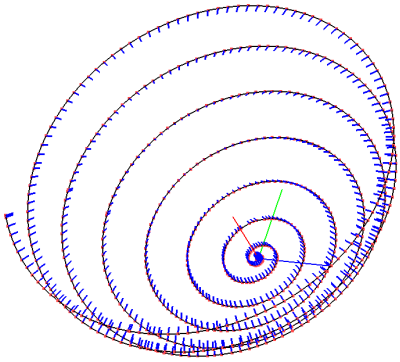

図 2.9. CLS ビューワによるスパイラルパスの表示結果

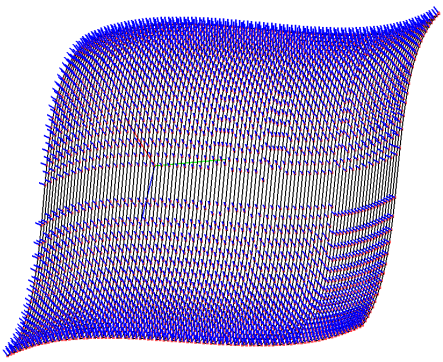

図 2.10. CLS ビューワによるジグザグパスの表示結果

```
        glVertex3fv((float*)&m_model.Vertex[IndexVertex0]);
        glVertex3fv((float*)&m_model.Vertex[IndexVertex1]);
        glEnd();
      }
    }
```

格納されている 3 つの頂点の座標から二つを選択し，glVertex3fv( ) により連続する 2 つの位置座標を指定後に，glBegin(GL_LINES) をコールする．これを for 文で 3 回行うこで三角パッチを描画する．これを STL ファイル内の全てのデータに対して繰り返し実行することでモデルの描画を行う．また，工具のパスのみを表す CLS データとは異なり STL データは 3 次元のモデルデータであるため，点，線の描画に加えて面の描画を行う必要がある．glVertex3fv( ) により連続する 3 点を指定後，glBegin(GL_TRIANGLES) をコールすることで三角パッチの面を描画できるようにした．

```
//線の描画処理
for(counter=0; counter¡IndexVertex; counter++)
 {
   glBegin(GL_TRIANGLES);
   glVertex3fv((float*)&m_model.Vertex[counter]);
   glVertex3fv((float*)&m_model.Vertex[counter+1]);
   glEnd( );
 }
```

法線ベクトルの描画は STL データの facet が書かれている行の値を参照して行った．三角パッチの重心位置の値と法線ベクトルの位置座標の値を glVertex3fv( ) により指定後，glBegin(GL_LINES) をコールして描画を行った．

表示結果の例を図 2.11，図 2.12, 図 2.13 に示す．ビューワは STL の点や面，線，法線ベクトルを描画している．

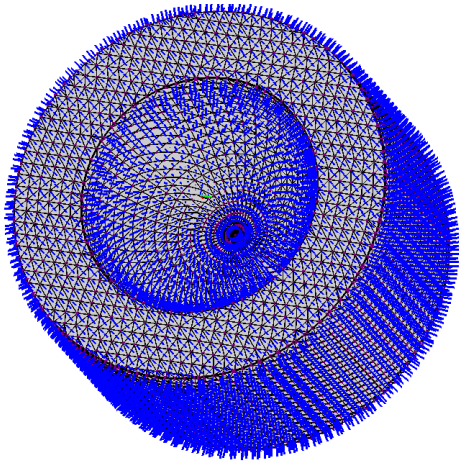

図 2.11. STL ビューワによる臼型モデルの表示結果

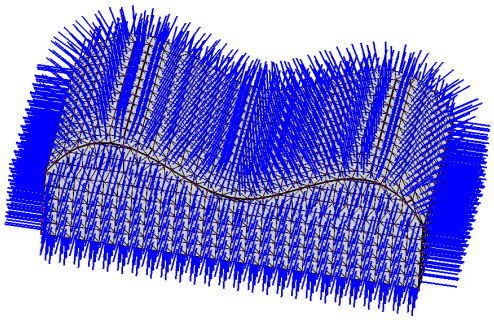

図 2.12. STL ビューワによる曲面の表示結果

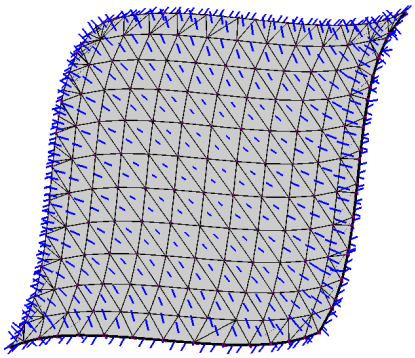

図 2.13. STL ビューワによるバイナリ形式の表示結果

第 3 章

Kinect を用いたリバースエンジニアリング

3.1 Kinect

今回研究で使用する Kinect V2 は 2014 年にマイクロソフト社から発売され，カラーセンサやデプスセンサ，マイクを搭載し様々な情報を取得することが出来る．Kinect V1 と比べ全体的に性能が向上しており解像度の改善，骨格を取得できる人数及び関節数の増加等が挙げられる．図 3.1 に Kinect V2 本体，表 3.1 に Kinect V2 の仕様を示す．

表 3.1 より解像度が 1920×1080(Full HD) に向上したことにより，従来 Full HD 画像を得るために Web カメラを併設することが多かったサイネージなどの分野で Kinect V2 センサ単

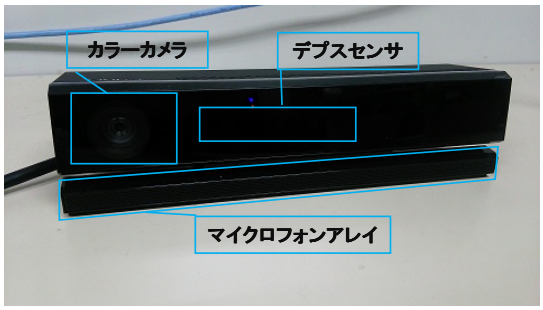

図 3.1. Kinect の各種センサ

表 3.1. Kinect の仕様

カラーセンサの解像度	1920×1080 pixel
デプスセンサの解像度	512×424 pixel
一度に取得できる人物	6 人
デプスセンサの取得範囲	0.5〜8.0 m
人物の検出範囲	0.5〜4.5 m
角度	水平:70 度，垂直:60 度

体での活用が出来るようになった．Depth センサは計測方法が赤外線光のパルスをカメラ側から照射し，その光パルスが対象物に反射して戻ってくるまでの飛行時間を計測する方法である ToF(Time of Flight) 方式を採用することで，より精度よく距離情報を習得することが出来るようになった．骨格の検出も検出できる人数が 6 人検出できる関節が 25 個と増加し対象が座った状態でも骨格を検出することが出来るようになった．

3.2　OpenCV

OpenCV(Intel Open Source Computer Vision Library) は Intel 社が開発したオープンソース，マルチプラットフォームの C/C++ ライブラリであり現在は Itseez が開発，管理を行っている．OpenCV を使用することでコンピュータ・ビジョンに必要な各種機能を使用することが出来る．具体的には
線形代数や統計など，コンピュータビジョンに必要な各種数学関数
直線や曲線，テキストなど画像への描画関数
OpenCV で使用したデータを読み込み/保存するための関数
エッジ等の特徴抽出や画像の幾何学変換，カラー処理等の画像処理関数
物体追跡や動き推定などの動画処理用の関数
物体検出などのパターン認識関数
3 次元復元のためのカメラ位置や姿勢検出などのカメラキャリブレーション関数
コンピュータにパターン学習させるための機械学習関数
画像の読み込みや保存，表示，ビデオ入出力などインターフェース用関数
等の機能を利用することが出来る． OpenCV によって高度な画像処理プログラムやリアルタイムの画像処理プログラムを簡単に作成することが出来るようになった．また，多くの人が利用しているため，より効率的なアルゴリズムの提案やバグ修正が頻繁に行われている．今回の研究では Kinect から取得した画像データを扱うために OpenCV を使用する．OpenCV を利用するための環境構築にはいくつかの方法があり，今回は環境構築が容易である公式パッケージを使ったインストールにより OpenCV の導入を行った．

3.3　PCL

PCL(Point Cloud Library) は 3D ポイントクラウドと呼ばれる 3 次元の点の集まり (点群) の処理を行うオープンソースフレームワークである．OpnCV の姉妹プロジェクトでありポイントクラウドデータに対する 3D 的なコンピュータビジョンの最先端のアルゴリズムをパッケージ化している．Kinect が登場するまでは精密な計測を行う高価な 3D センサしか存在せず 3D 点群処理の需要は土木建築での測量やロボットでのマシンビジョン，医療用の 3D スキャンなど限られたものだった．しかし Kinect が発売以降安価なデプスセンサによる 3D 点群処理が手軽に実現できるようになり，PCL のような 3D 点群処理をまとめたオープンソースライブラリの必要性が高まった．PCL では以下のようなポイントクラウドデータを用いた

処理が，モジュールごとに分けられた形で C++ のクラス郡として提供されている．
ポイントクラウドデータの可視化
ポイントクラウドデータの入出力
フィルタリング
法線方向の取得
位置合わせ
サーフェスの表示
等の機能が使用できる．PCL では PCL 専用の点群データ形式である PCD ファイルがサポートされている．従来の CAD や 3D ソフトがサポートしているデータ形式には Wavefont 社の OBJ ファイルや，3DSystem 社の STL ファイル，スタンドフォード大学が公開している PLY 形式などがあるが PCD ファイルはそれらのファイルを踏襲したファイル形式である．テキスト形式で点の数や法線の数，メッシュの数などが書かれており，次に 1 行ずつ点や法線，メッシュの値が書かれている．本実験では PCL の機能を使用して Kinect から取得したデータを STL 形式に変換する処理を行う．

3.4 PCD データのスムージング処理

PCL には 3 D 点群データを保存する独自のフォーマット PCD 形式をサポートしている．PCD データは保存している PCD の情報を格納しているヘッダー部と実際の点の持っている値が記述されているデータ部にわかれて記述されている．PCD データの中身を図 3.2 と表 3.2 に示す． Kinect から取得した X, Y, Z, 座標の値や色情報を PCD 形式で保存するには PCL の関数である pcl::io::savePCDFile( ) を呼び出して行う．保存した PCD データは KinectV2 の Depth データ取得方式が TOF 方式のためノイズが多く表面が荒くなっている．これを解決するために PCD データのスムージング処理を行った．PCL にはスムージングを行うための関数が用意されているが今回はスムージングの処理の原理が理解するために独自にスムージ

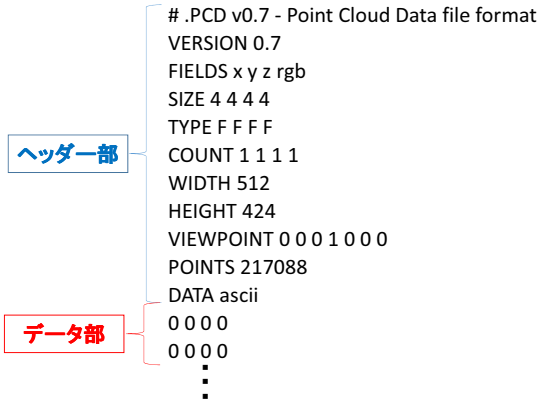

図 3.2. Kinect の各種センサ

ングを行うプログラムを作成した．このスムージング処理には 2 種類のパラメータで行うことが出来る．一つはスムージングの小ウィンドウの大きさ l でもう一つはスムージング処理を行うために求めた標準偏差の係数 η である．小ウィンドウの大きさが大きく，標準偏差の係数が小さいほどスムージングの効果が大きくなる．今回作成したスムージング処理の概要を図 3.3 に示す．図のようにウィンドウの大きさを決定した後，以下の式よりウィンドウ内の点の z の値の平均を求める．$z_{i,j}$ は小ウィンドウの中心点の z 値である．

$$\bar{z}_{x,y} = \frac{1}{l^2} \sum_{j=y}^{y+l-1} \sum_{i=x}^{x+l-1} z_{i,j} \tag{3.1}$$

次に，以下の式より標準偏差を求める．

$$\sigma_{x,y} = \sqrt{\frac{1}{l^2} \sum_{j=y}^{y+l-1} \sum_{i=x}^{x+l-1} (z_{i,j} - \bar{z}_{x,y})} \tag{3.2}$$

以下の式より，求めた標準偏差 $\sigma_{x,y}$ よりウィンドウの中心点の z 値 $z_{i,j}$ と $\bar{z}_{x,y}$ の差が大きいときスムージングを行う．この処理を小ウィンドウをスライドさせながら繰り返す．

$$|z_{x,y} - \bar{z}_{x,y}| \geq \eta \sigma_{x,y} \tag{3.3}$$

図 3.4 にスムージング前，スムージング後の PCD データを示す．

3.5　PCD-STL コンバータと加工実験

保存した PCD データを加工するために現在 3D プリンタのフォーマットとして広く普及しており研究室の加工ロボットにも使用できる STL データに変換した．PCD データは図 3.2 に示した通り縦方向に 424 個横方向に 512 個の点が並んでおり計 217008 個の点で画像を表現している．図 3.5 のように 4 つの点を選択し 2 つの三角形を作成し，その処理を繰り返し行うことにより PCD データから三角パッチベースの STL データを生成した．スムージング処理を

表 3.2. 各ヘッダー部の記述の意味

VERSION	PCD データのヴァージョン
FIELDS	PCD データの持っている次元
SIZE	次元の持っているバイト数
TYPE	保存したデータ型
COUNT	データの持っている要素数
WIDTH	点群の行数
HEIGHT	点群の列数
VIEWPOINT	点群の視点位置
POINTS	点群の点の総数
DATA	PCD のフォーマット形式

行っていない STL データと行っていない STL データの比較を図 3.6 に示す． 変換した STL データは研究室の加工用ロボットで加工するために吉本らが開発した STL-CLS コンバータにより CLS データに変換した．変換した CLS データの比較を図 3.7 に示す．今回の加工ではピックフィード 0.2 のジグザグパスの工具パスを生成した．加工用ロボットで加工したモデルを図 3.8 に示す．

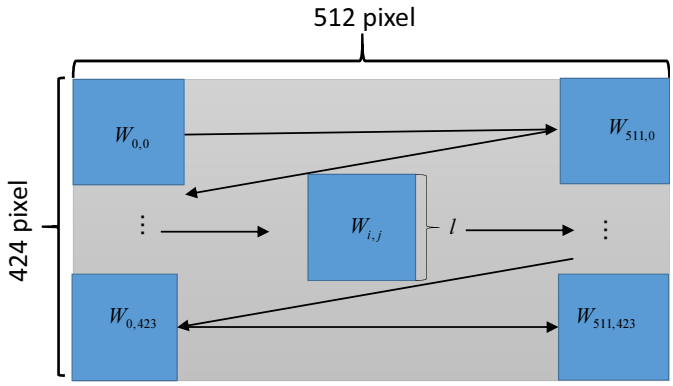

図 3.3. スライドする小ウィンドウを用いたスムージング処理

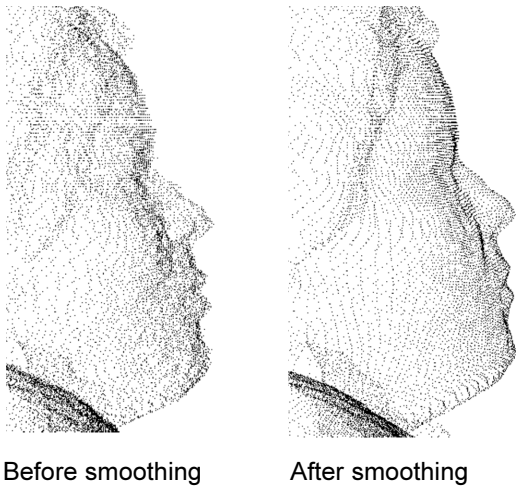

図 3.4. スムージング処理を行った PCD データ

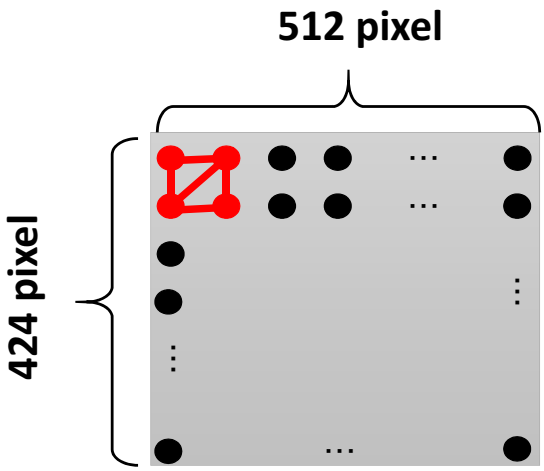

図 3.5. PCD から STL を生成する概要

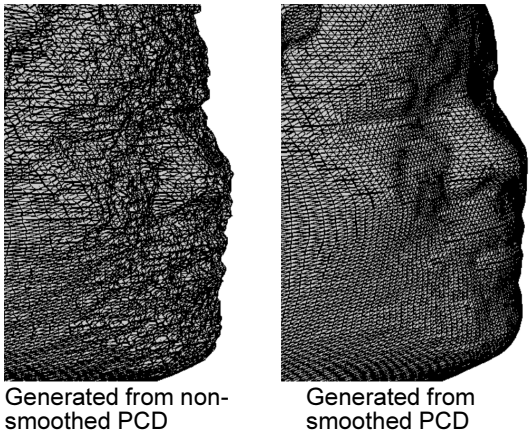

図 3.6. PCD から生成した STL ファイル

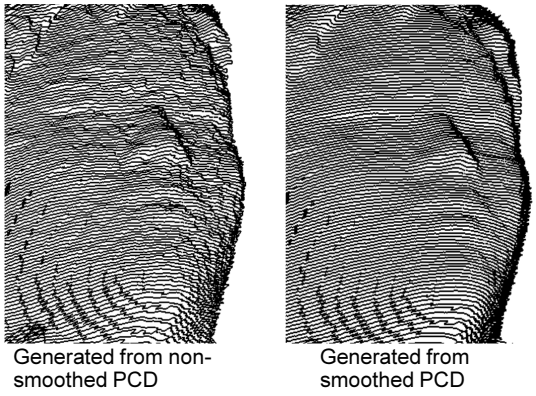

図 3.7. STL から生成した CLS ファイル

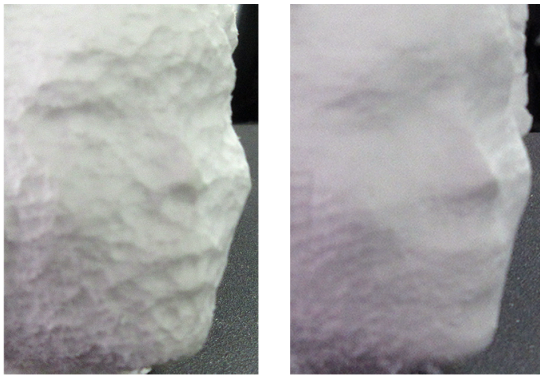
Machined using non-smoothed CLS data
Machined using smoothed CLS data

図 3.8. 加工したモデル

第 4 章

結言

従来の研究室の作業環境では，加工を行う前に CLS ファイルや STL ファイルを確認したい場合，その都度 CAD/CAM システム側で再計算させる必要があり，制御 PC 側では表示させることができなかった．これを解決するために OpenGL を利用した CLS, STL ファイルビューワを作成した．このビューワを利用することによって，作業者が加工を行う前に制御 PC 側で CLS ファイルの加工パスや STL ファイルのモデルを簡易・迅速に表示できることを確認できた．Kinect から加工対象のデータを取得して PCD データとして保存し，保存した PCD データを STL データに変換して加工が行えることが確認できた．Kinect から得たデータを基に保存した PCD データは KinectV2 の Depth 情報を得る方法が TOF だったためノイズが多く表面が荒かったが，スムージング処理を行うことで表面が滑らかなデータを生成することが出来た．今回の研究では加工したモデルの大きさに対して工具の方が大きかったため元のデータに対して削りすぎてしまった．そのため加工するモデルをより大きくすることにより，加工する精度を上げることが出来ると考えられる．また，一方向からの画像のみで PCD を生成したので側面のデータは正確に得られていないので多方向からの画像を結合させることでより正確なデータの加工が行えると考えられる．

謝辞

本研究は，山口東京理科大学大学院基礎工学研究科基礎工学専攻で行われたものである．本研究の遂行ならびに本論文のまとめに際し，終始懇切丁寧な御指導と多大なる御助言を頂きました山口東京理科大学工学部機械工学科永田寅臣教授に心よりお礼申し上げます．また，本研究を遂行するに当たりご協力いただきました大塚先生をはじめ永田研究室の皆さんに心より感謝いたします．

参考文献

[1] 中村航輔, 永田寅臣, 渡辺桂吾, “産業用ロボットのための「ロボット言語を用いない CAM システム」の提案と設計,” 第 14 回計測自動制御学会システムインテグレーション部門学術講演会 (SI2013) 講演論文集 , **58**, pp. 1477-1482, 2013.

[2] D. Shreiner, M. Woo, J. Neider and T. Davis, “OpenGL Programming Guide (Sixth Edition) – The Official Guide to Learning OpenGL, Version 2.1,” Addison-Wesley Professional, 2007.

[3] “Khronos Group クロノスグループ日本語サイト” http://jp.khronos.org/, 2015,10/23 閲覧.

[4] R. Kempf, C. Frazier, “OpenGL リファレンスマニュアル– The Official Guide to Learning OpenGL, Version 1.1”, ピアソン・エデュケーション, 1999.

[5] 一見大輔 “入門 立体形状のラピッドプロトタイピング”, オーム社, 2013.

[6] F. Nagata, A. Otsuka, K. Watanabe and M. K. Habib, “Machining robot for foamed polystyrene materials using fuzzy feed rate controller,” *International Journal of Mechatronics and Automation*, Vol. 5, No. 1, pp. 34-43, 2015.

[7] F. Nagata, S. Yoshitake, A. Otsuka, K. Watanabe and M. K. Habib, “Development of CAM System Based on Industrial Robotic Servo Controller without Using Robot Language,” *Robotics and Computer-Integrated Manufacturing*, Vol. 29, No. 2, pp. 454–462, 2013.

[8] 福原元一, “JIS ハンドブック　工作機械,” 日本規格協会, pp.378-379, 1994.

[9] 中村薫, 杉浦司，高田智広，上田智章，“KINECT for Windows SDK プログラミング–Kinect for Windows v2 センサー対応版,” 秀和システム, 2015.

[10] 小枝正直，上田悦子，中村恭之 “OpenCV による画像処理入門”, 講談社, 2015.

[11] “itseez-vision that works!,” http://itseez.com/, 2015, 10/29 閲覧.

[12] 河内基樹，木村元, “深度計測デバイスによるデータからの 3D 図面自動生成に関する研究,” 日本船舶海洋工学会講演会論文集, pp. 489-492, 2013.

[13] 高橋将平，岡哲資, “KINECT を用いた 3D スキャナの開発”，日本大学生産工学部第 45 回学術講演会概要, pp. 499-500, 2012.

[14] Mark Segal, Kurt Akeley, Jon Leech, “OpenGL4.0 グラフィックスシステム,” 株式会社カットシステム, 2010.

[15] F. Nagata, K. Takeshita, S. Yoshimoto, A. Yshinaga, S. Kurita, K. Watanabe, Maki K. Habib, “Viewer, Converter and Preprocessor for Smart Machining Process Using an Industrial Robot,”Proceedings of 21st International Symposium on Artificial Life and Robotics,” pp. 441-445, 2016.

[16] 山本晃, “Kinect と PCL を用いた立体物の 3 次元モデリング,” 卒業論文, 2013.

[17] 西浦直樹 “Kinect を用いた立体物の計測と PCL による 3 次元点群処理,” 卒業論文, 2012

[18] 中村薫, 斎藤俊太, 宮城英人, “KINECT for Windows SDK プログラミング C++ 編,” 秀和システム, 2012

[19] 谷尻豊寿, “体の動きがコントローラ C++ で Kinect プログラミング Kinect センサー 画像処理プログラミング,” カットシステム, 2011

[20] 伊藤大生, 田村仁, 高塚崇文, 大川涼, ダビデミンナイ , “RGB-D カメラによって得た頭部の情報の三次元モデル化手法,” 第 74 回全国大会講演論文集, pp. 245-246, 2012.