2000年度学士論文

カラー画像の距離画像へのマッピング

指導教授 : 中西 正和 教授
斎藤 博昭 専任講師

慶應義塾大学 理工学部 情報工学科
中西・斎藤研究室 学部4年

学籍番号:69725320

吉藤 伸幸

目 次

第1章　序論

1.1　はじめに

コンピュータビジョンは，2 次元画像を解析することによって画像に写った 3 次元物体の形状や運動を求めること，すなわち 2 次元物体から 3 次元情報を復元することを主要なテーマとして研究が進められてきた．これはカメラによる 3 次元情報から 2 次元情報への幾何学的・光学的変換 (投影) の逆問題を解くことに相当し，投影によって失われた奥行情報をいかにして安定かつ高精度に求めるかという本質的に困難な問題を含んでいる．投影の逆問題を解くには，画像の生成過程を正確に記述する必要があるため，カメラのモデル化 (カメラモデル) と各種パラメータの決定法がコンピュータビジョン研究の当初から議論され現在も盛んに研究が行われている．[1]

1.2　本論文の目的

近年，レンジセンサの性能の向上もあり，以前に比べ容易に正確なレンジデータが測定できるようになった．また，カメラの性能も向上している．これまでは，光投影式距離計測センサーという 1 つの機器によって色と距離を求めて、コンピュータ上で物体を再現していたが，先に述べた 2 つの機器によって測定したものをアラインメントするといった研究についてはほとんど行われていない．本研究では，カメラとレーザレンジセンサデータのアラインメントの新しい手法について提案する．カメラから得られるカラー画像と別の位置にあるレーザレンジセンサから得られる距離画像の位置合わせのために，対象物体とカメラの間のカメラパラメータを求め，その結果を利用し距離画像にカラー画像から得られる色情報をテクスチャとして貼るというシステムの構築を目指す．

1.3 本研究の概要

本研究では図 1.1 のようにカメラとレンジセンサのアラインメントを行う.

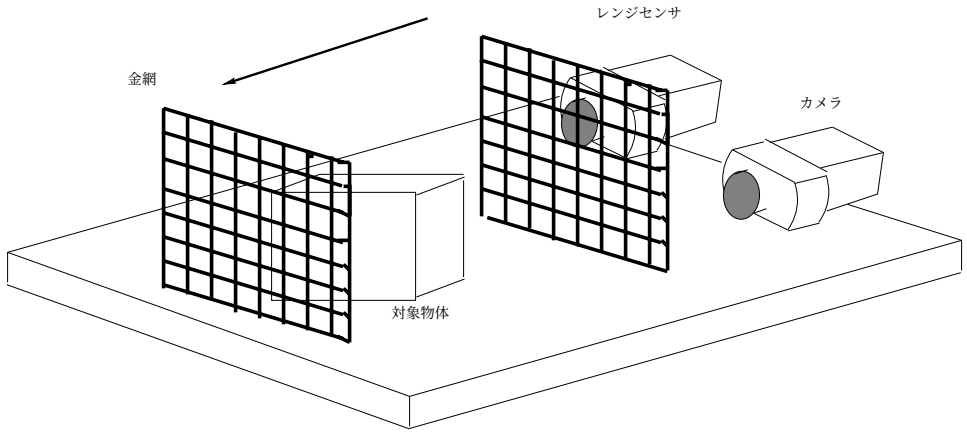

図 1.1: カメラとレンジセンサのアラインメント

カメラとレンジセンサのアラインメントとは，レンジデータにおけるある点がカラー画像のどのピクセルにあたるかといった対応づけを行うことである．そのためには，2 つの機器の間のカメラパラメータの推定が必要となる．カメラパラメータとは，レンジデータ (x_w,y_w,z_w) と画像座標 (X_f,Y_f) を対応づけるパラメータである．カメラパラメータを求めることで，3 次元座標から，画像座標を予測したり，逆に画像から 3 次元座標が予測できるようになる．

カメラパラメータを推定するためには，レンジデータと画像の間で，特徴点の対応が必要となる．本研究では，白い金網の格子の交点をカメラパラメータを推定する際の特徴点として用いる．理由としては，

- レンジセンサでの形状の計測が容易.
- カメラで撮影したときに，背景との分離が容易.

というものが挙げられる.

以上より，カメラとレンジセンサのアラインメントが行われたので，それを利用して，距離画像にテクスチャを貼る.

1.4　本論文の構成

本論文の構成を以下に示す.

2 章: カメラパラメータ推定に関する理論について述べる.

3 章: 特徴点抽出の方法とそれに関するに理論について述べる.

4 章: テクスチャマッピングについて述べる.

5 章: 実験結果について述べる.

6 章: 結論と今後の課題について述べる.

第2章　カメラパラメータの推定

本研究で用いるカメラパラメータとは以下の 11 のパラメータで構成される．

- 外部パラメータ
  - x 軸基準の回転角 (radian)
  - y 軸基準の回転角 (radian)
  - z 軸基準の回転角 (radian)
  - 平行移動ベクトル T の x 成分
  - 平行移動ベクトル T の y 成分
  - 平行移動ベクトル T の z 成分
- 内部パラメータ
  - 焦点距離
  - 画像原点 C_x
  - 画像原点 C_y
  - レンズひずみ係数
  - スケール係数

ここでは，カメラパラメータを推定する手法として，Tsai の手法 [2] について説明する．

2.1　世界座標と画像座標の関係

まず準備として，点 P の世界座標系における 3 次元座標 (x_w,y_w,z_w) とコンピュータ上の画像座標 (X_f,Y_f) の関係を考える．そのために以下の 4 段階の変換を行う．

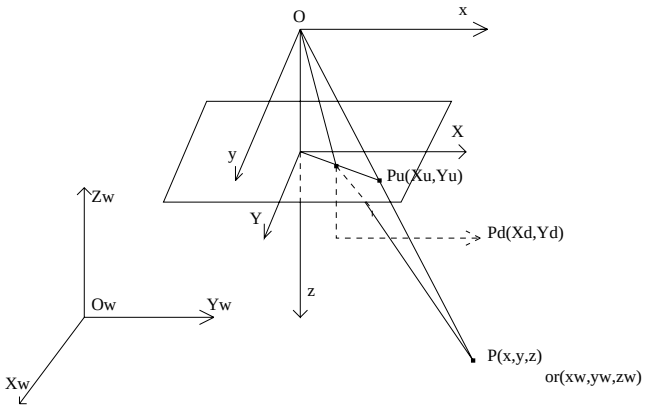

図 2.1: 世界座標と座像座標の関係

Step1

(x_w,y_w,z_w) から (x,y,z) への変換: 回転行列 R, 平行移動ベクトル T を用いて表す.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T \tag{2.1}$$

ただし,

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix},\ T = \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \tag{2.2}$$

Step2

透視変換による (x,y,z) から (X_d,Y_d) への変換: 焦点距離 f を用いて表す.

$$X_u = f\frac{x}{z},\ Y_u = f\frac{y}{z} \tag{2.3}$$

Step3

(X_d,Y_d) から (X_d,Y_d) への変換:

$$X_d + D_x = X_u,\ Y_d + D_y = Y_u \tag{2.4}$$

ただし，D_x,D_y はレンズの半径方向のひずみ係数 k_1，k_2 を用いて次のように表わす.

$$D_x = X_d(k_1 r^2 + k_2 r^4) \tag{2.5}$$

$$D_y = Y_d(k_1 r^2 + k_2 r^4) \tag{2.6}$$

$$r = \sqrt{X_d^2 + Y_d^4} \tag{2.7}$$

Step4

(X_d,Y_d) から (X_f,Y_f) への変換:

$$X_f = s_x d_x^{'-1} X_d + C_x \tag{2.8}$$

$$Y_f = d_y^{'-1} Y_d + C_y \tag{2.9}$$

ただし，s_x はスケール係数，(C_x,C_y) はディジタル画面上の原点座標，d_x,d_y はそれぞれ X 方向 Y 方向の CCD 素子の間隔を表す．なお，$d_x^{'}$ は X 方向の CCD 素子数 N_{cx} と，1 走査線のサンプル数 N_{fx} を用いて d_x を補正したものである．スケール係数 s_x とする．

$$d_x^{'} = d_x \frac{N_{cx}}{N_{fx}} \tag{2.10}$$

以上の関係を整理すると，

$$X = X_f - C_x,\ Y = Y_f - C_y \tag{2.11}$$

を用いて

$$s_x^{-1} d_x^{'} X + s_x^{-1} d_x^{'} X(k_1 r^2 + k_2 r^4) = f \frac{r_1 x_w + r_2 y_w + r_3 z_w + T_x}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \tag{2.12}$$

$$d_y Y + d_y Y(k_1 r^2 + k_2 r^4) = f \frac{r_4 x_w + r_5 y_w + r_6 z_w + T_y}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \tag{2.13}$$

が得られる．ただし，

$$r = \sqrt{(s_x^{-1} d_x' X)^2 + (d_y Y)^2} \tag{2.14}$$

である．

2.2 パラメータの決定

a) ひずみのある画像座標 (X_d,Y_d) の計算

1) それぞれの特徴点 i の座標を求め，(X_{fi},Y_{fi}) とする．
2) カメラおよび A/D 変換器の仕様から N_{cx},N_{fx},d'_x,d_y を求める．
3) 画像の中央を画像原点 (C_x,C_y) とする．
4)N 点の (X_{di},Y_{di}) を求める．

$$X_{di} = s_x^{-1} d'_x (X_{fi} - C_x) \tag{2.15}$$

$$Y_{di} = d_y (Y_{fi} - C_y) \tag{2.16}$$

b) 5 つの未知数を計算する．

(x_{wi},y_{wi},z_{wi}) と (X_{di},Y_{di}) の組から $T_y^{-1}r_1$,$T_y^{-1}r_2$, $T_y^{-1}T_x$,$T_y^{-1}r_4$,$T_y^{-1}r_5$ を未知数とする線形方程式を解く．

$$\begin{bmatrix} Y_{di}x_{wi} \\ Y_{di}y_{wi} \\ Y_{di} \\ -X_{di}x_{wi} \\ -X_{di}y_{wi} \end{bmatrix}^t \begin{bmatrix} T_y^{-1}r_1 \\ T_y^{-1}r_2 \\ T_y^{-1}T_x \\ T_y^{-1}r_4 \\ T_y^{-1}r_5 \end{bmatrix} = X_{di} \tag{2.17}$$

c) $(T_y^{-1}r_1,\ T_y^{-1}r_2,\ T_y^{-1}T_x,\ T_y^{-1}r_4,\ T_y^{-1}r_5)$ から $(r_1,\ \cdots,\ r_9,T_x,T_y)$ を計算する．

1)$T_y^{-1}r_1$,$T_y^{-1}r_2$,$T_y^{-1}T_x$,$T_y^{-1}r_4$,$T_y^{-1}r_5$ から $|T_y|$ を計算する．
$T_y^{-1}r_1$,$T_y^{-1}r_2$,$T_y^{-1}T_x$,$T_y^{-1}r_4$,$T_y^{-1}r_5$ がすべて 0 でない場合は，以下のように T_y^2 は求まる．

$$T_y^2 = \frac{S_r - \sqrt{S_r^2 - 4(r'_1 r'_5 - r'_4 r'_2)^2}}{2(r'_1 r'_5 - r'_4 r'_2)^2} \tag{2.18}$$

ただし，$S_r = r_1'^2 + r_2'^2 + r_4'^2 + r_5'^2$ とする．
そうでないときは，

$$T_y^2 = (r_i'^2 + r_j'^2)^{-1} \tag{2.19}$$

となる．r'_i，r'_j は $r_i(i$ =1,2,4,5) のなかで 0 でないものとする．

2) まず，T_y の符号を正とする．

3) 以下の計算をする．

$$r_1 = (T_y^{-1} r_1) T_y,\ r_2 = (T_y^{-1} r_2) T_y$$

$$r_4 = (T_y^{-1} r_4) T_y,\ r_5 = (T_y^{-1} r_5) T_y$$

$$T_x = (T_y^{-1} T_x) T_y$$

$$x = x_1 x_w + r_2 y_w + T_x,\ y = x_4 x_w + r_5 y_w + T_y$$

4)x と X の符号が等しくかつ y と Y の符号が等しいなら T_y の符号を正，それ以外では T_y の符号を負とする．

5) 以下の式より，R を計算する．

$$R = \begin{bmatrix} r_1 & r_2 & \sqrt{1 - r_1^2 - r_2^2} \\ r_4 & r_5 & s\sqrt{1 - r_4^2 - r_5^2} \\ r_7 & r_8 & r_9 \end{bmatrix} \tag{2.20}$$

r_7,r_8,r_9 は Rが直交行列であるという性質を用いて求める．

6) このあと求める焦点距離 f が負となる場合は R を以下の式を用いる．

$$R = \begin{bmatrix} r_1 & r_2 & -\sqrt{1 - r_1^2 - r_2^2} \\ r_4 & r_5 & -s\sqrt{1 - r_4^2 - r_5^2} \\ -r_7 & -r_8 & r_9 \end{bmatrix} \tag{2.21}$$

ただし，$s = -sgn(r_1 r_4 + r_2 r_3)$ とする．

d) レンズひずみを無視して f,T_z の初期値を求める

それぞれの点 i について，f,T_z を未知数とする以下の 1 次方程式を解く．

$$\begin{bmatrix} y_i & -d_y Y_i \end{bmatrix} \begin{bmatrix} f \\ T_z \end{bmatrix} = w_i d_y Y_i \tag{2.22}$$

ただし，

$$y_i = r_4 x_{wi} + r_5 y_{wi} + T_y$$

$$w_i = r_7 x_{wi} + r_8 y_{wi}$$

とする．

e) f,T_z,k_1 の正確な答えを求める

d) で求めた f,T_z および k_1=k_2=0 を初期値として，(2.13) においてもっとも誤差が少なくなるようにそれぞれの値を決定する．

以上の手法により，空間中のある点のおける世界座標とそれに対応する画像座標が分かれば，カメラパラメータを決定できることが分かる．次章では，本研究で用いる白い金網の格子の交点について，それらの座標を求める方法について考える．

第3章　特徴点抽出

本研究では，白い金網の格子の交点をカメラパラメータの推定を行う際の特徴点として用いる．しかし，それらの点を直接求めるのは容易でないので，まず金網の線の部分の直線の方程式を求めてそれらの交わる点を交点と考える．

コンピュータビジョンにおいて直線を検出する手段としては，Hough 変換 [3]，直線近似の方法としては最大 2 乗最適化法がある．まず，これらの原理について述べる．

3.1　直線抽出

3.1.1　Hough 変換

まず，Hough 変換の原理について述べる．Hough 変換は 1962 年に Hough，P．V．C によって考案され，Rosenfeld や，Duta and hart によって画像処理の世界に紹介されて以来，雑音を含む画像からの安定な直線検出法およびパターン検出・照合法として定着した．

始めに 2 次元画像からの直線検出を例に Hough 変換の原理を説明する．X-Y 平面に存在する直線は，傾き a_0，Y 切片 b_0 とすると

$$y = a_0 x + b_0 \tag{3.1}$$

と表せる．図 3.1(a) のような画像中の点 $(x_i,\ y_i)$ を通る直線は

$$y_i = a x_i + b \tag{3.2}$$

と表され，これを a，b に関する方程式と考えてその軌跡を図 3.1(b) のように a-b 平面に描く．(このような軌跡を Hough 曲線という) 画像中の全特徴点 (エッジ点) に対して同様の作業を行い，a-b 平面内で多くの軌跡が交わる座標 $(a_0,\ b_0)$ を検出することによって入力画像中に存在する直線を検出することができる．

しかし，有限の大きさの画像中でも直線が Y 軸に対して平行になる場合，傾き a と Y 切片は無限大となってしまう．そこで図 3.2(a) のように極座標表現をもちいることにより，直線は

$$\rho = x\cos\theta + y\sin\theta \tag{3.3}$$

のように記述することができ，有限のパラメータ空間全ての直線を表現することが可能となる．このとき Hough 曲線は図 3.2(b) のように正弦曲線となる．ここでパラメータ ρ は原点からの距離，θ は直線の法線ベクトルが X 軸となす角を示す．

実際の画像処理では以下の手順で直線検出を行う．まず，ρ-θ 空間を適当なサンプル間隔で量子化する．次に画像上の点 $(x_i,\ y_i)$ を式 (3.3) によって ρ-θ 空間へ変換する．この際，式 (3.3) によって表される Hough 曲線が通過する量子化されたセル $(\rho,\ \theta)$ の値を 1 増やす．この作業を投票と呼ぶ．この変換を画像上の全ての点について行うと，ρ-θ 空間において度数の極大値となっている座標を検出することによって，X-Y 平面に存在する直線を検出することができる．

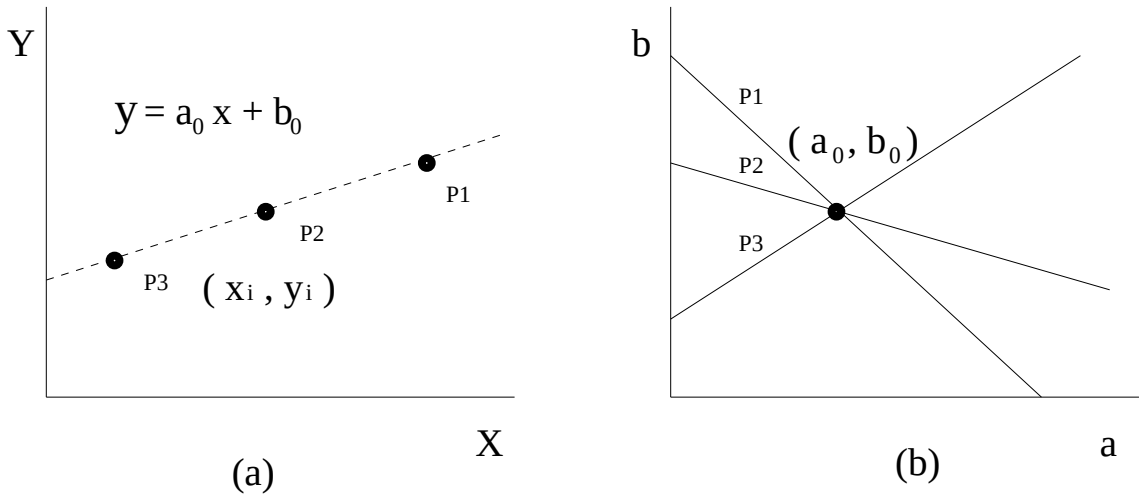

図 3.1: a-b 空間を用いた Hough 変換

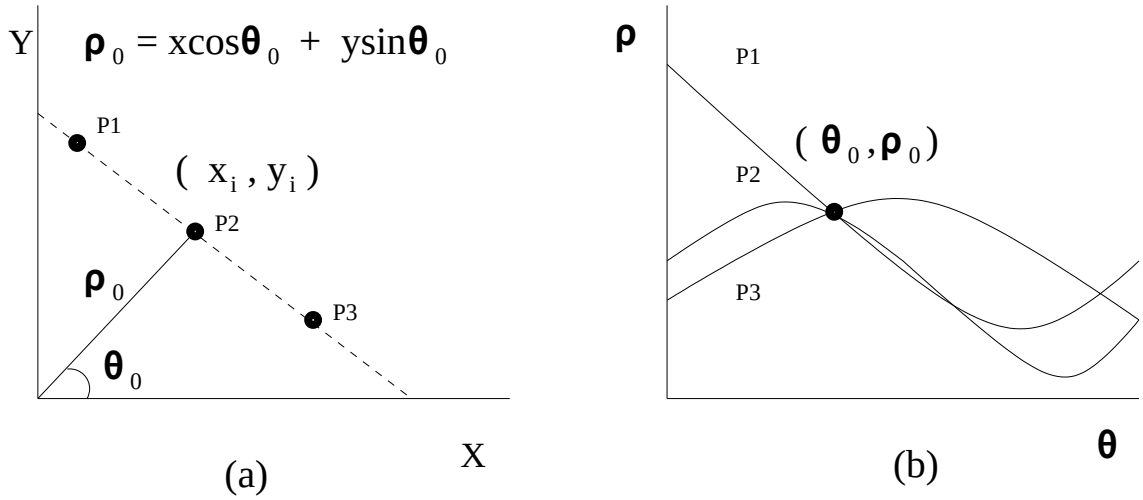

図 3.2: ρ-θ 空間を用いた Hough 変換

3.1.2 逐次検出型

Hough 変換は特頂点の誤りや欠陥に対して安定な図形検出を行うことが出来るが，検出される図形パラメータの精度は必ずしも高いとはいえない．この原因としては，ノイズ・隠蔽・量子化誤差の他に，画像中に複数の図形が含まれる場合，他からの図形の投票が重畳され，図計間の干渉が生じることがあげられる．そのために，従来の方法により検出を行うと．正しい抽出図形の近辺に多数の誤った検出が起こってしまい．検出画像で見ると抽出図形がぶれておりその正確な位置がわからないような結果がわからないような結果となる．

ここでは，その誤検出を抑制する方法として逐次検出型を用いる．これは，入力画像中に多数の検出対象図形を含む場合に，その１つ１つを投票の多いものから順に抽出しつつ．抽出画面から画像を消去しながら計算を繰り返して行く方法であり，図形間の干渉及び欠損率の違いによる個々の図形間の投票数の隔たりを克服することが出来る．その結果，誤った検出を抑制できると共に，検出図形の数が正確に求められるという特徴がある．しかし，抽出対象図形の数だけ計算を繰り返すので時間がかかるという問題点もある．

逐次検出型の手順

逐次検出型 Hough 変換は以下の手順で行う．

(1) 入力画像に対して計算を行い，パラメータ空間に投票を行う．
(2) 投票値が最大となるパラメータを検出し，そのパラメータ群で表される図形を抽出図形として検出する．
(3) その画像を入力画像から消去し，中間画像とする．
(4) 中間画像に対して再度計算を行い．パラメータ空間に投票をおこなう．
(5) (2) 以降の操作を検出対象図形が無くなるまで繰り返す．

3.1.3 最小2乗最適化法

Hough 変換で求められた直線を $y = ax + b$ または $x = c$ とし，その近辺にある点の座標を $(x_i,\ y_i)$ とし，その点の数を n 個あったとする．

$\sum x^2 = \sum_{i=1}^{n} x_i^2$，$\sum = \sum_{i=1}^{n} x_i$，$\sum xy = \sum_{i=1}^{n} x_i y_i$，$\sum y = \sum_{i=1}^{n} y_i$，と表すとすると，

$$\begin{cases} a\sum x^2 + b\sum x - \sum xy = 0 \\ a\sum x + bn - \sum y = 0 \end{cases} \tag{3.4}$$

が得られる．これを解いて，

$$\begin{aligned} a &= \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2} \\ b &= \frac{\sum x \sum xy - \sum y \sum x^2}{(\sum x)^2 - n\sum x^2} \end{aligned} \tag{3.5}$$

と求められ，近似直線が決定する．

3.1.4 本研究における直線抽出

ρ-θ 空間を，ρ を x-y 平面における $\sqrt{x^2 + y^2}$ の最大値の 1000 分割 θ を 3600 分割で量子化する．

投票の最も大きい座標を検出された直線とし，x-y 空間においてその直線場にある点をデータから消去する．更にその画像においてまた投票を行う．この繰り返しによって画像中の直線を全て抽出することが出来る．

Hough 変換と最大 2 乗最適化法によって，2 つの画像中に存在する直線をまず求める．距離画像 (図 3.3) に関しては，$z = 0$ の平面中で直線 (図 3.4) を求め，その交点の x，y 座標と最も近い x，y の値を持つ 3 次元の点を空間中の交点とする．

カラー画像に関しては撮影した画像 (図 3.5) を二値化して，そのピクセルの値をそれぞれ (x,y) として金網を抽出しそれを利用し直線を抽出した (図 3.6)．

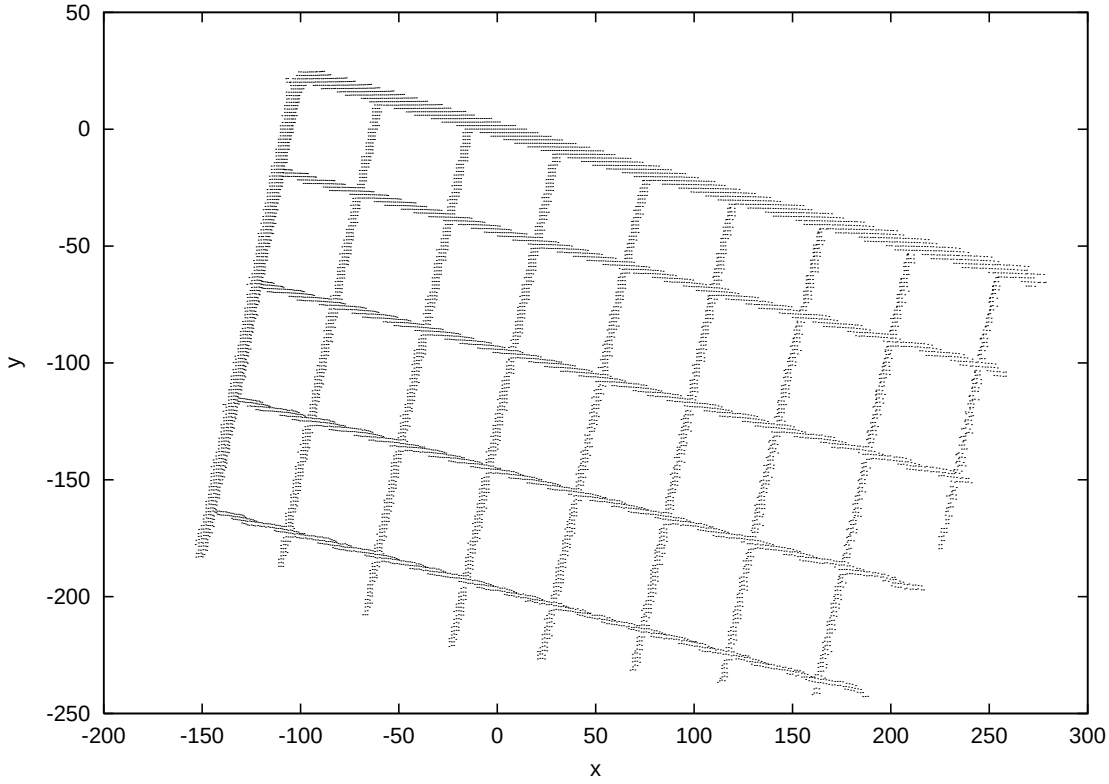

図 3.3: レンジセンサで撮影した網の距離画像 (z=0)

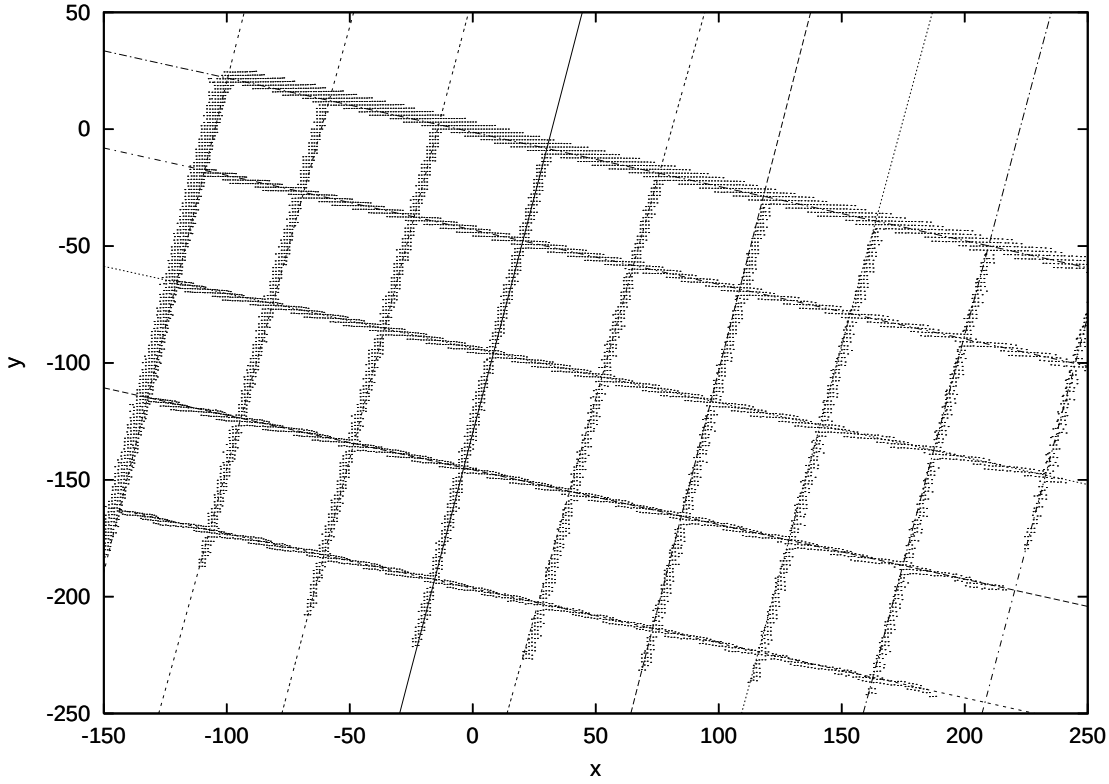

図 3.4: レンジセンサで撮影した網の距離画像から直線を抽出 (z=0)

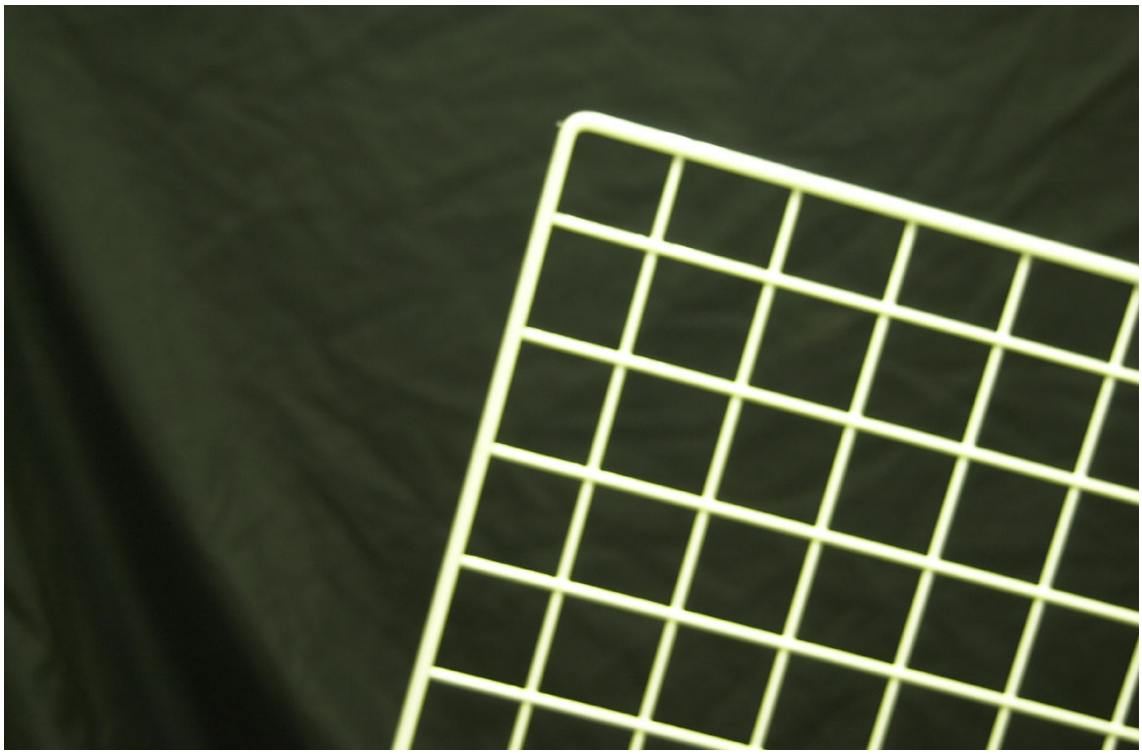

図 3.5: カメラで撮影した網の画像

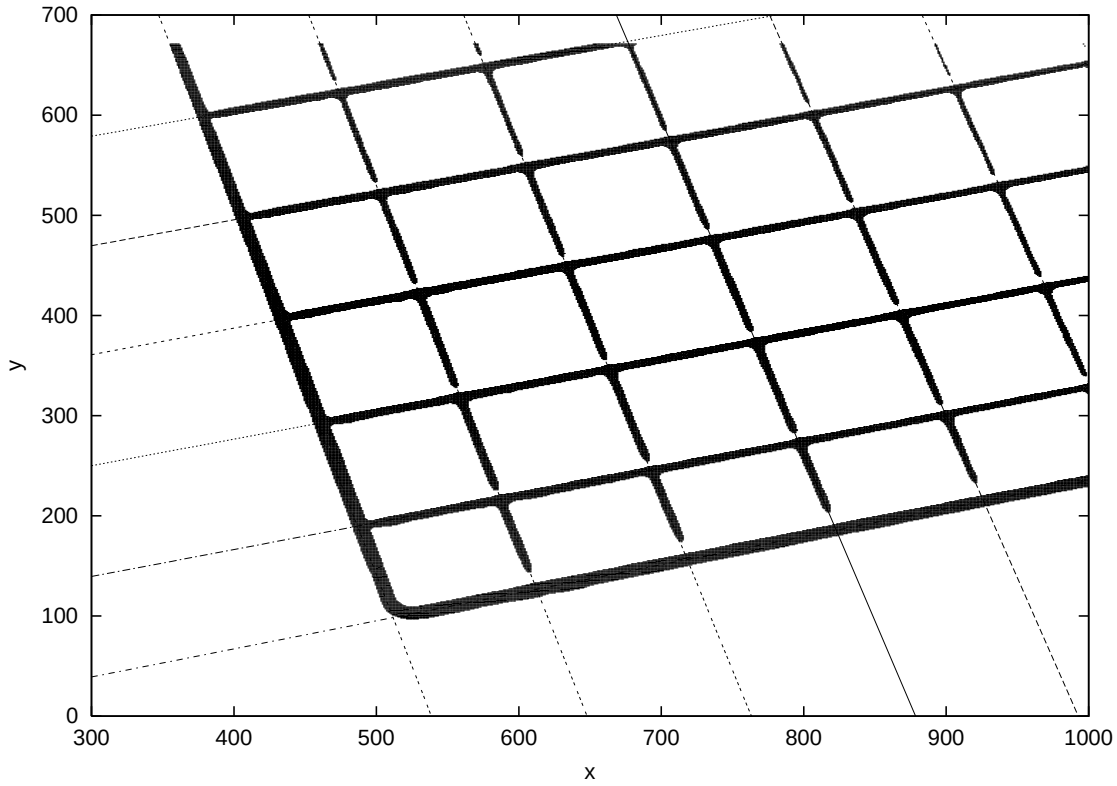

図 3.6: ピクセルの値を x，y とし直線を抽出

3.2 交点の対応付け

3.1で述べたような方法で金網の線の部分のそれぞれの直線は求められるが，カメラパラメータの推定を行うには，各点において距離画像とカラー画像で対応がとれていなければならない．ここではその方法について説明する．

まず，求めたそれぞれの直線を傾きの絶対値でソートする．ここで金網の形状から，傾きの絶対値が1未満の直線を横方向の直線，1以上の直線を縦方向の直線と仮定する．更に，横方向の直線においてはx=(画像の中心)におけるy座標，縦方向の直線においてはy=(画像の中心)におけるx座標でそれぞれソートする．これにより，横方向の直線の本数をm，縦方向の直線の本数をnとすると，それぞれの画像において求められた直線に，横方向の直線a1 〜 am ，縦方向の直線b1 〜 bnと番号をつけることができる(図3.7).

求める交点は両方の画像中にある点だけなのでそれぞれの方向の直線の数を各々の小さいほう(m，n)とし，一番上の直線と一番左の直線の交点を1番目の点(左上の角)，同様に一番上の直線と左から2番目の直線の交点を2番目の点…とし，更に縦向きの直線が最後までいったら，上から2番目の直線と一番左の直線の交点を求めるというふうにして，それぞれの画像で対応させながら交点を求める．

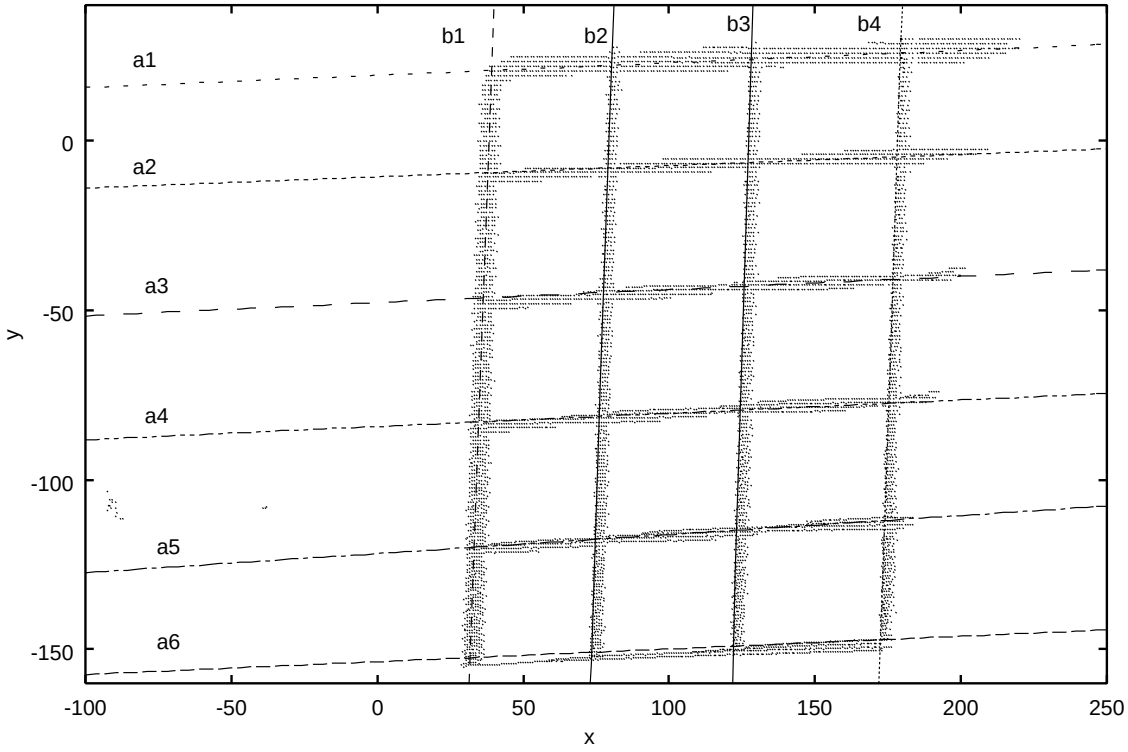

図 3.7: 直線に番号をつける

第4章　テクスチャマッピング

現実感のある立体画像を簡易的に生成する方法であるテクスチャマッピングは様々な場面で広く利用されている．ここでは，その画像生成法について述べる．

テクスチャマッピングの手法は，写真などの実写の映像を立体の上に貼りつけて，より現実的な立体画像を生成させるのもである．例えば，地球の立体画像を作成したい場合，スキャナーなどを用いて，あらかじめディジタル平面の上に地形図を入力しておき，この 2 次元ディジタル画像を球上へマッピングすれば，現実に極めて忠実な地球が作成されることになる．この際マッチングされるディジタル画像をテクスチャマップという．

テクスチャマッピングには，様々な手法があり，目的に応じて使いわけられる．代表的なテクスチャマッピング手法には以下のものがある．

4.1　テクスチャマッピング

狭い意味でのテクスチャマッピングは，写真などの 2 次元平面の画像を 3 次元物体の表面にマッピングする手法であり，一版にテクスチャマッピングというとこの手法を示すことが多い．これにも様々な手法がある．テクスチャには通常は，2 次元のディジタル画像を用いることが多いが，関数によって与える場合もある．

4.2　ソリッド テクスチャマッピング

ソリッド テクスチャマッピングはある点の色 $color$ を

$$color = F(p) \tag{4.1}$$

のように対象物体上の点 p の 3 次元座標の関数として与え，対応するテクスチャの色を得る手法である．

この手法は複雑な形状の物体にも容易にマッピングできるという利点がある．本研究ではこの手法を用いる．

本研究においては，(2.12)(2.13) をそれぞれ変形した，

$$X = f\frac{r_1x_w + r_2y_w + r_3z_w + T_x}{s_x^{-1}d_x'(r_7x_w + r_8y_w + r_9z_w + T_z)(1 + k_1r^2 + k_2r^4)} \tag{4.2}$$

$$Y = f\frac{r_4x_w + r_5y_w + r_6z_w + T_y}{d_y(r_7x_w + r_8y_w + r_9z_w + T_z)(1 + k_1r^2 + k_2r^4)} \tag{4.3}$$

を用いて，3 次元座標 $p(x_w, y_w, z_w)$ から，画像座標 (X, Y) を求め，そのピクセルにおける色を $color$ とする．

4.3 3 角形メッシュの作成

本研究では，モデル曲面を 3 角形メッシュとみなし，それぞれの 3 角形の頂点の点の色情報 (RGB 値) をカメラで撮影したカラー画像から得ることで，モデル表面の描写を行う．ここでは，モデル曲面のレンジデータから 3 角形メッシュを作成する方法について述べる．

対象物体のレンジデータをレンジセンサが測定範囲の左上から図 4.1 のように測定を行うという性質から，データの始めから，x 座標が減少する寸前までの点列を 1 つの配列に，その次の点列も同じようにして 2 つ目の配列に格納する．

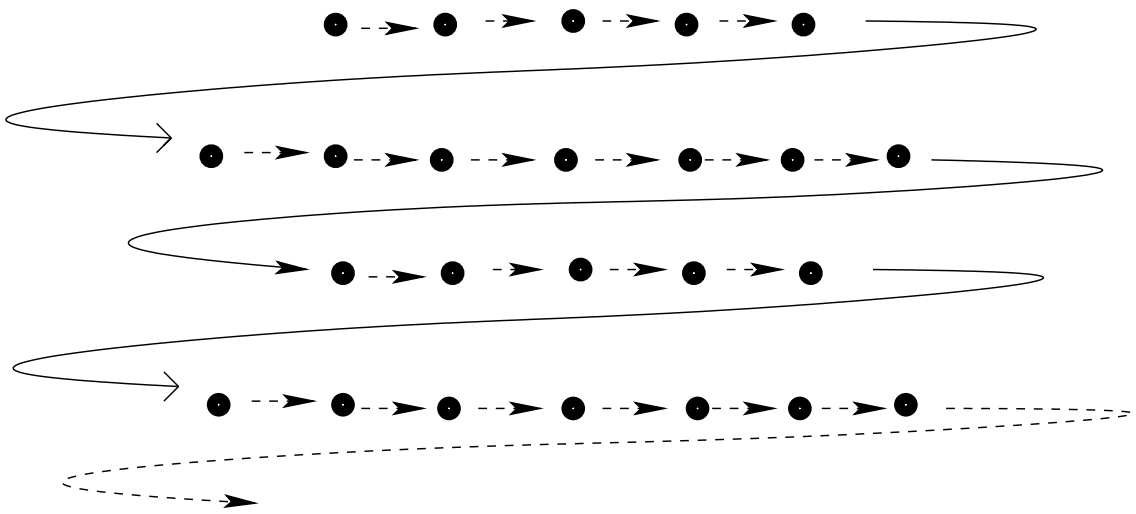

図 4.1: レンジセンサーの測定方法

そして，図 4.2 のように 1 番目の点列のほうが点の数が少ないときは，1 番目の点列の左端 (a1) と 2 番目の点列の左端 (b1) と左から 2 番目の点 (b2) で 1 つの三角形を作り，もし，2 番目の点列の 3 番目の点 (b3) の x 座標が 1 番目の点列の左端の点 (a1) の x 座標より小さいときは (図 4.2(b))，その後に a1，b2，b3 で三角形を作る．a1 の x 座標より小さい値を持つ 2 番目の点列がなくなったら，つぎに a1，a2，そして a1 より小さいかつ最大の x 座標を持つ 2 番目の点列で三角形を作る．以下同じ ようにして 2 つの点列から三角形を作って行く．片方の点列が最後までいったら最後の点ともう片方の点列の点で三角形をつくる．

1 番目の点列のほうが数が大きいときも，方法は同じである．

そして次に今の 2 番目の点列を 1 番目の点列とし，レンジデータ上の次の点列を 2 番目の点列とし，今の作業を繰り返す．

最後にそれらの三角形の頂点の座標とともにその点の RGB の値を与えて，OpenGL で表示する．なお，OpenGL では三角形の頂点座標と色情報を与えることで，三角形の辺や，三角形の内部の色も補正して表示される．

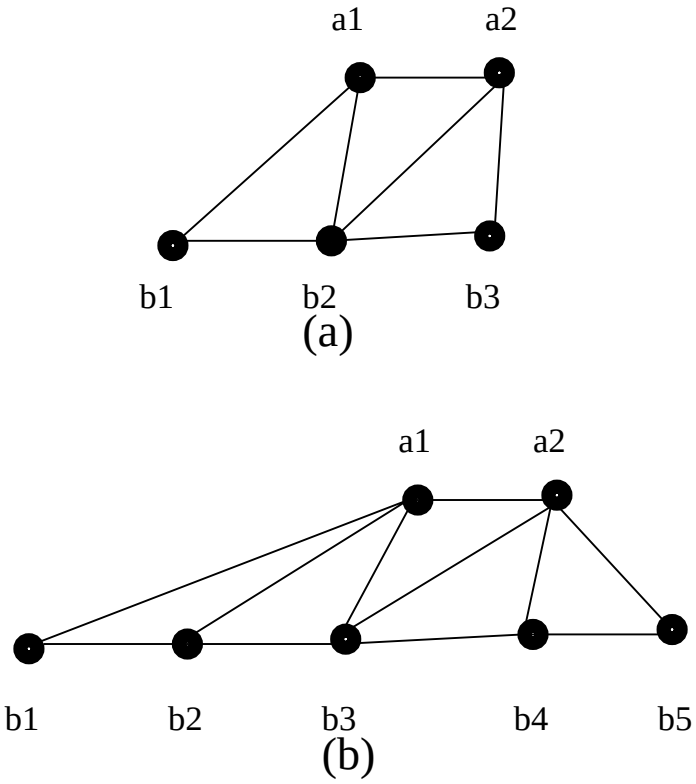

図 4.2: 3 角形メッシュの作成

第5章　実験

5.1　カメラパラメータの推定

5.1.1　実験装置

図 5.1 の実験装置を用いて実験を行った.

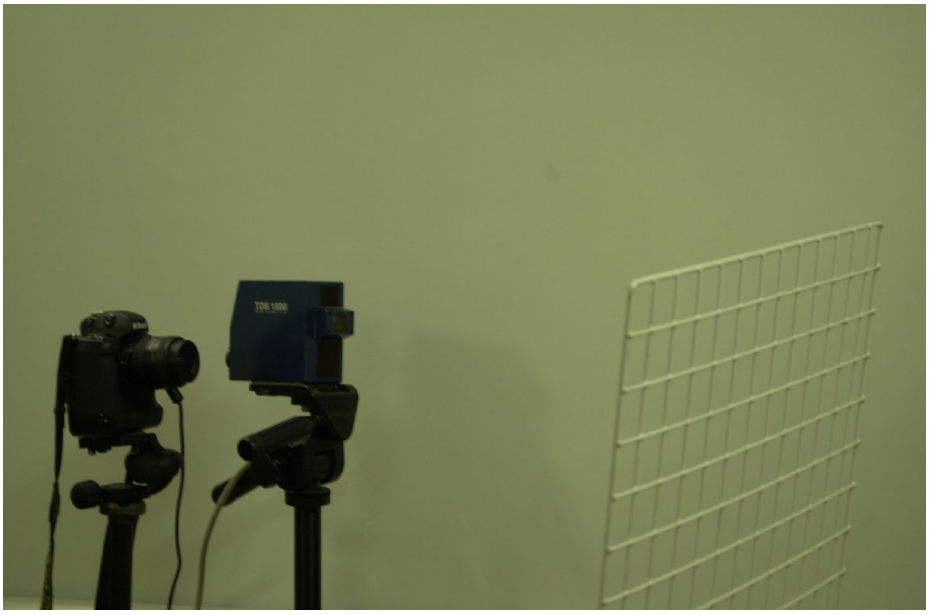

図 5.1: 実験装置

実験器具は以下のとおりである.

- レンジセンサ (TDS-1500)
- カメラ (NIKON-D1)
- 金網 (80cm×40cm，1つの格子 5cm×5cm)

5.1.2 実験結果

求められたカメラパラメータを表 5.1 に示す.

表 5.1: カメラパラメータ

x 軸基準の回転角 (radian)	-3.08852
y 軸基準の回転角 (radian)	0.28593
z 軸基準の回転角 (radian)	0.07153
平行移動ベクトル T x 成分 (mm)	-202.20840
平行移動ベクトル T y 成分 (mm)	-64.78072
平行移動ベクトル T z 成分 (mm)	86.20050
焦点距離 (mm)	47.38317
画像原点 C_x	646.14105
画像原点 C_y	370.01452
レンズひずみ係数	-6.13302e-05
スケール係数	1.00329

このカメラパラメータを用いて，特徴点の世界座標から対応する画像座標を求めた時の誤差は平均で 2.807[pix]，最大で 7.870[pix] となった．(画像の大きさは 1024[pix] × 672[pix])

5.2　テクスチャマッピング

5.2.1　実験装置

図 5.2 の実験装置を用いて実験を行った.

図 5.2: 実験装置 2

実験器具は以下のとおりである.

- レンジセンサ (TDS-1500)
- カメラ (NIKON-D1)

5.2.2 実験

対象物体 1 の距離画像を図 5.3，カメラで撮影した画像を図 5.4，同様に対象物体 2 の距離画像を図 5.5，カメラで撮影した画像を図 5.6 とする．

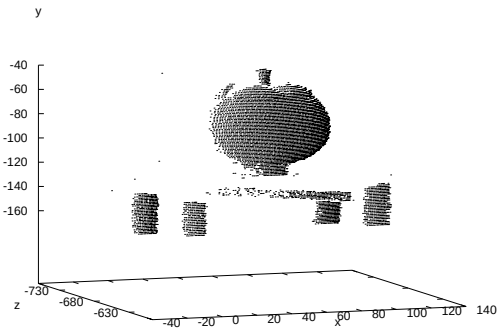

図 5.3: 対象物体 1 をレンジセンサで計測したもの

図 5.4: 対象物体 1 をカメラで撮影したもの

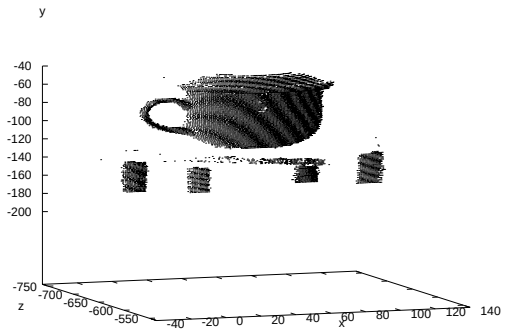

図 5.5: 対象物体 2 をレンジセンサで計測したもの

図 5.6: 対象物体 2 をカメラで撮影したもの

5.2.3 実験結果

5.1で求めたカメラパラメータを用いてレンジセンサデータの各点に対応するカラー画像のピクセルの位置のRGB値を求め，それを元にテクスチャを貼った結果が図5.7，図5.8(対象物体1)，図5.9，図5.10(対象物体2)である．

図 5.7: 距離画像にテクスチャを貼った対象物体1

図 5.8: 図5.7を上から見て時計回りに回転させたもの

図 5.9: 距離画像にテクスチャを貼った対象物体 2

図 5.10: 図 5.9 を上から見て反時計回りに回転させたもの

5.3 金網がおけないような位置にある物体の場合

奈良の大仏の前に金網をおき，本実験の手法により，カメラパラメータを求めた後に，大仏にテクスチャを貼ることを試みた．

まず，求められたカメラパラメータを表 5.2 に示す．

表 5.2: カメラパラメータ

x 軸基準の回転角 (radian)	-3.09116
y 軸基準の回転角 (radian)	0.01955
z 軸基準の回転角 (radian)	-0.05525
平行移動ベクトル T x 成分 (m)	-0.31143
平行移動ベクトル T y 成分 (m)	0.09723
平行移動ベクトル T z 成分 (m)	-0.04403
焦点距離 (m)	29.94935
画像原点 C_x	535.30189
画像原点 C_y	401.20311
レンズひずみ係数	-4.43533e-05
スケール係数	0.99892

このカメラパラメータを用いて，特徴点の世界座標から対応する画像座標を求めた時の誤差は平均で 0.953[pix]，最大で 5.030[pix] となった．(画像の大きさは 1024[pix] × 672[pix])

そして，金網と同時に大仏をレンジセンサーで測定 (図 5.11)，またカメラでも撮影し (図 5.12)，表 5.2 に示したカメラパラメータを用いて，距離画像にテクスチャを貼ったのが，図 5.13 である．

また，同様に大仏をレンジセンサーで測定 (図 5.14)，またカメラでも撮影し (図 5.15)，表 5.2 に示したカメラパラメータを用いて，距離画像にテクスチャを貼ったのが，図 5.16 である．

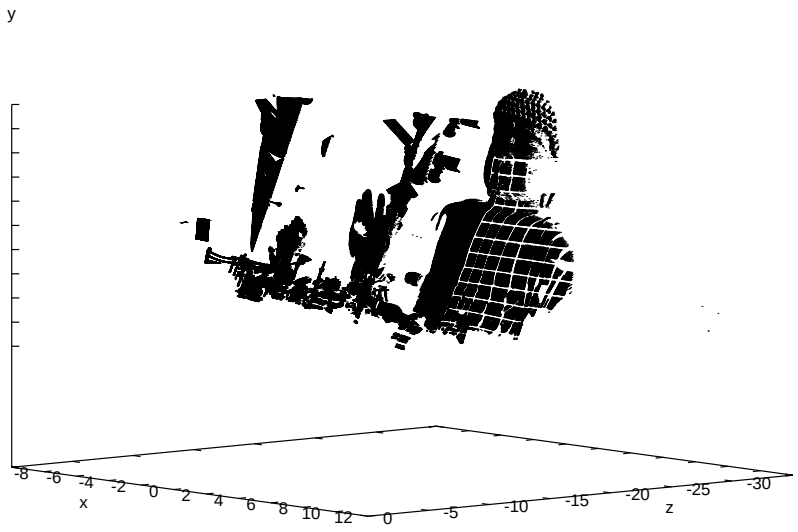

図 5.11: 対象物体をレンジセンサで計測したもの

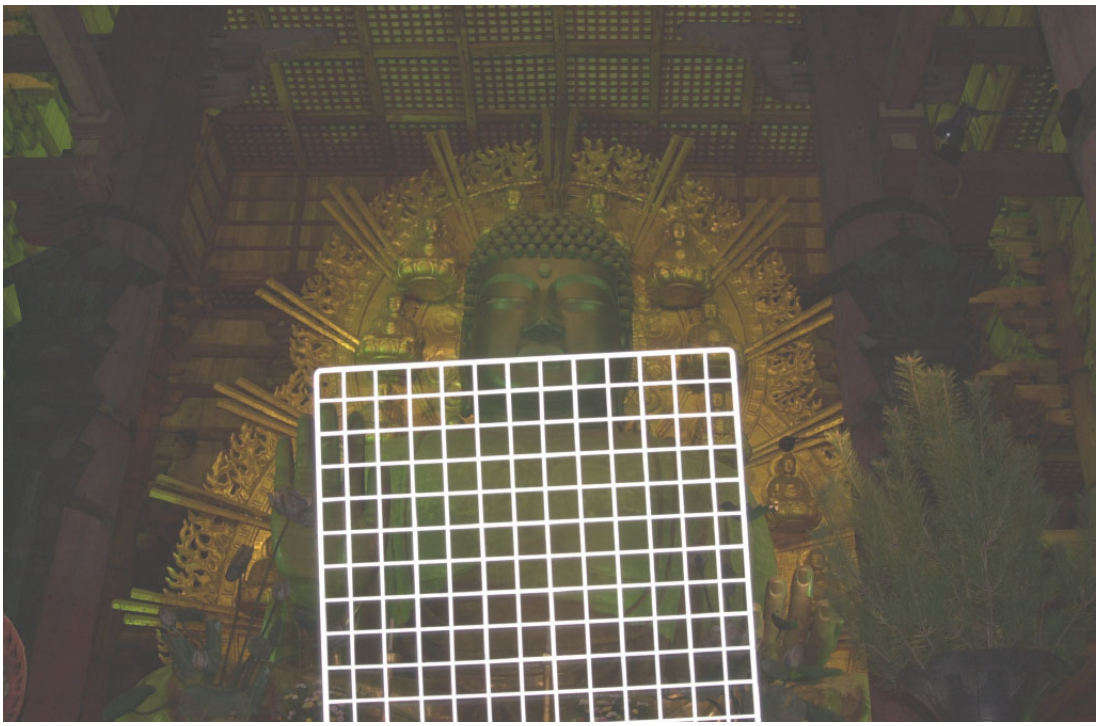

図 5.12: 対象物体をカメラで撮影したもの

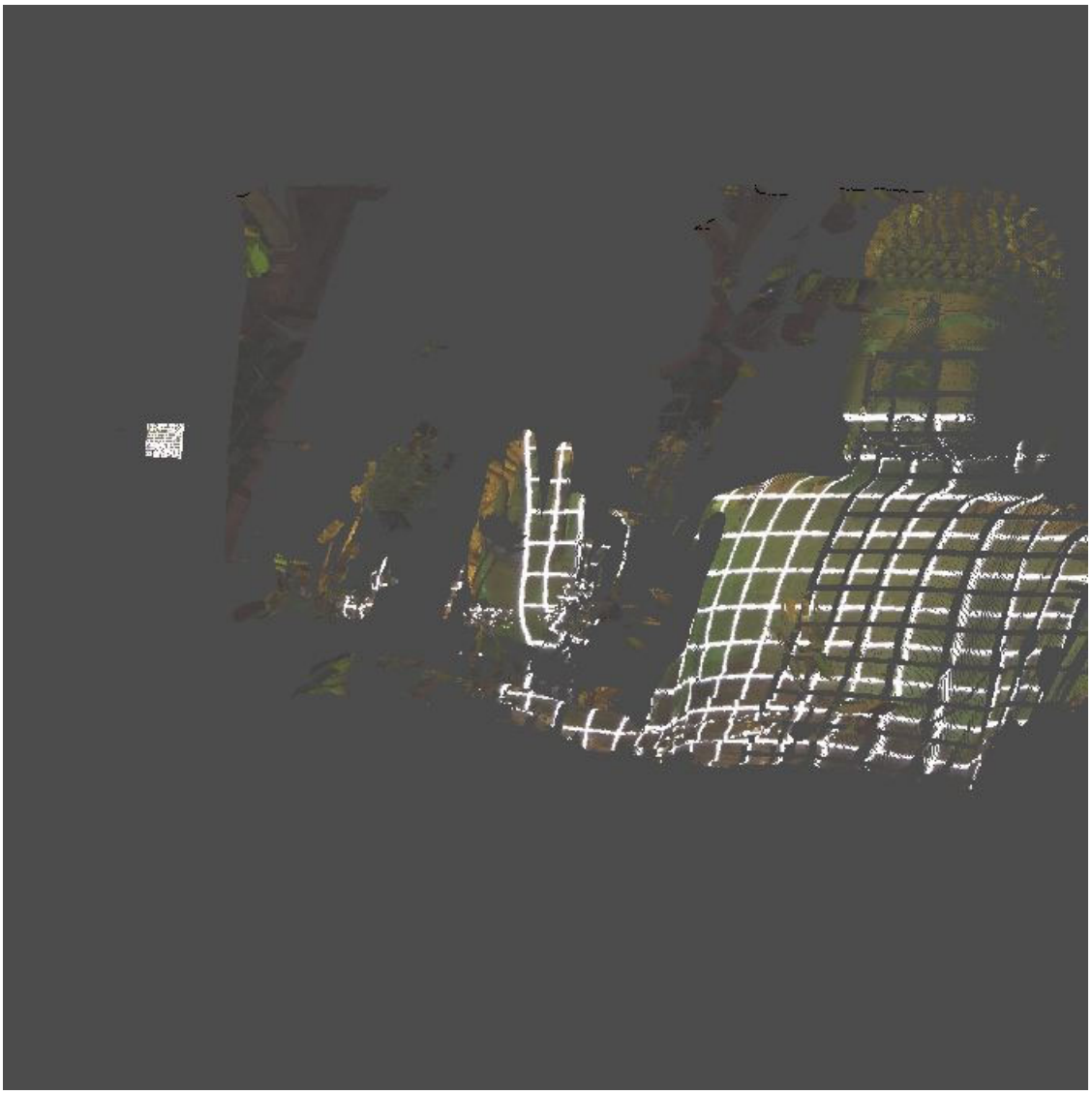

図 5.13: 距離画像にテクスチャを貼った対象物体

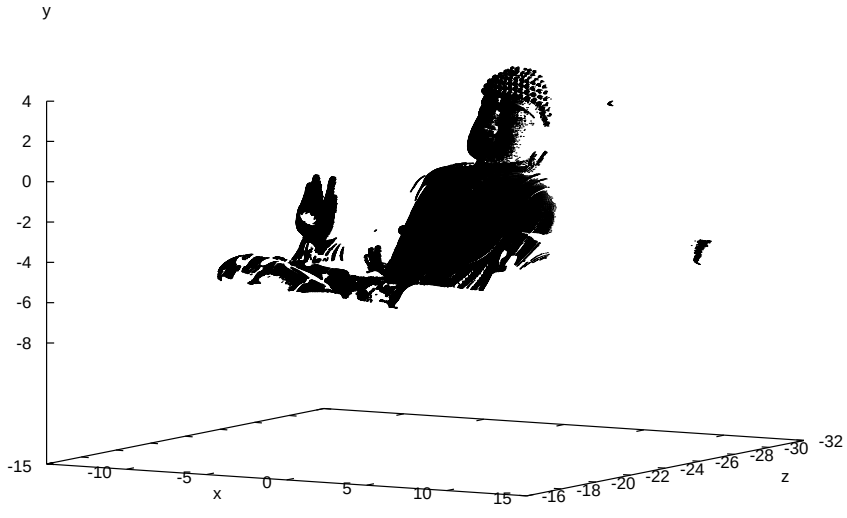

図 5.14: 対象物体をレンジセンサで計測したもの

図 5.15: 対象物体をカメラで撮影したもの

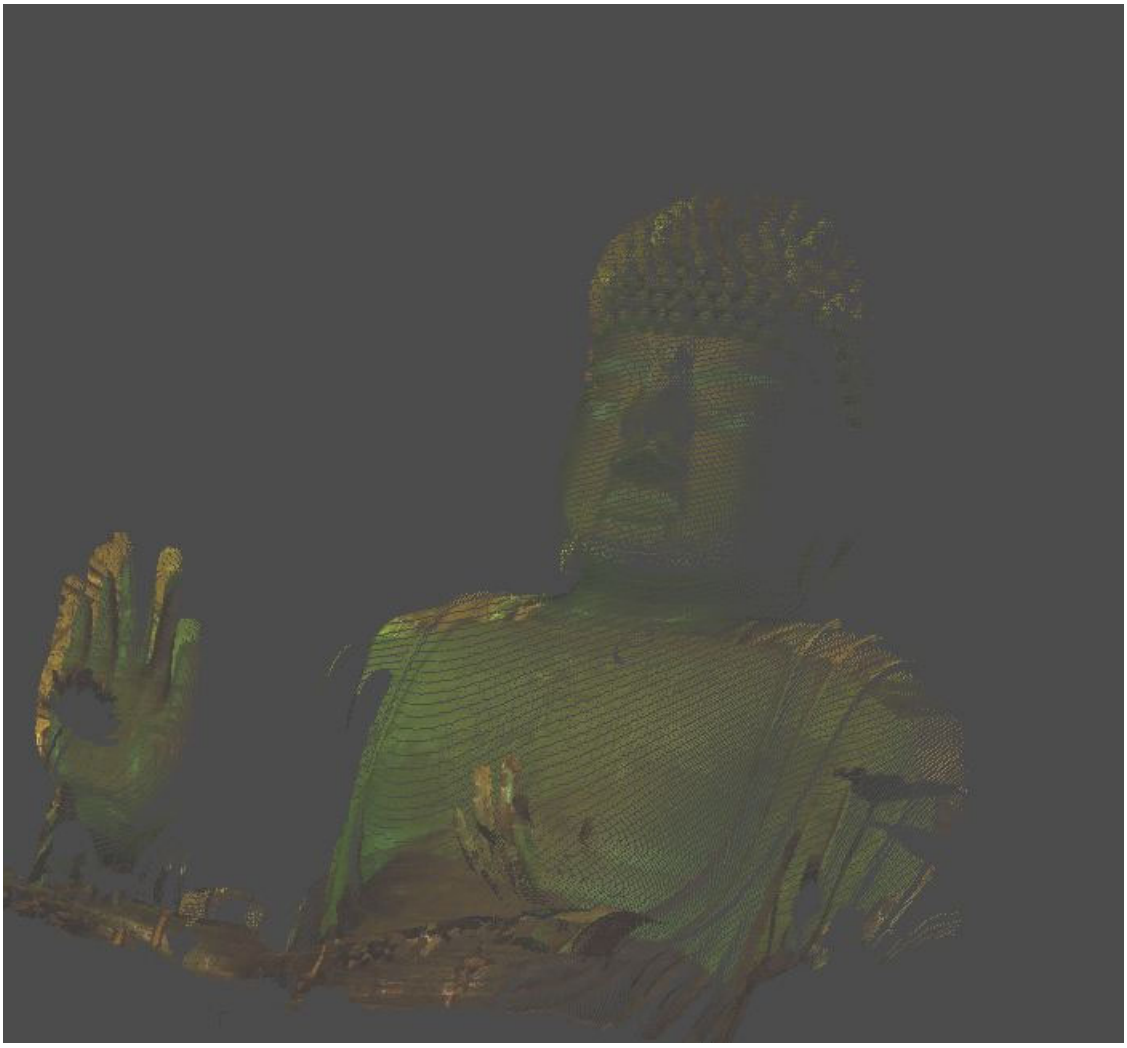

図 5.16: 距離画像にテクスチャを貼った対象物体

第6章　結論

6.1　まとめ

カメラとレンジセンサデータのアラインメントの新しい手法として，カメラから得られるカラー画像と別の位置にあるレンジセンサから得られる距離画像の位置合わせのために，対象物体とカメラの間のカメラパラメータを求め，その結果を利用し距離画像にカラー画像から得られる色情報をテクスチャとして貼るというシステムの構築を行った．これにより，距離画像の計測とカメラによる画像の撮影を同一視点から行わなくても，対象物体をコンピュータ上に再現することが可能となった．また，本研究の手法ならばレンジセンサの種類に関わらず，カラー画像を距離画像へマッピングすることができる．

6.2　問題点

- 網の太さや閾値のような値の設定が困難

  金網の格子の交点を求める際，逐次型 Hough 変換を用いるため，求められた直線付近にある点をデータから削除する．この際に網の太さ，つまり削除する範囲をうまく設定しないと，別の直線上の点を削除してしまい，正確に直線を求めることができない．しかし，実験環境によって画像中における網の太さが違ってしまうのは仕方ないことである．

  また，カラー画像から白い部分である金網を抽出する際も，わずかでも暗い場所で撮影を行うと，二値化する際の閾値によっては，全く白い部分が抽出されないといったことが起こる．

  以上のことから，このシステムを完全に自動化するのは困難である．

- 対象物体を再現できる環境が限られる．

  カメラパラメータ推定を行う際に金網を置いた平面間のみにおいて正確に世界座標と画像座標の対応がとれる．しかし，図 5.13 を見ると分かるように，金網においては正確に 2 つの座標系間の関係がとれていて，白く表示されているが，それ以外の場所，つまり大仏の表面上においては，対応が正しくとれていない．ある点の世界座標からカメラパラメータを用いて，その点の画像座標を求めた場合に実際と違う点を求めてしまうために，このようなことが起こる．

  これは，図 5.16 においても同様で，全体的に，距離画像に対してそれに対応する画像座標が実際よりも上にずれているためこのような結果となると思われる．

  つまり，建造物などのように，物体が固定されている場合，物体のある位置に金網が置けないときはこの手法は使えない．

謝辞

まず始めに，私の我儘により東京大学生産技術研究所池内研究室での研究を薦めて頂いた斎藤博昭専任講師に深く感謝致します．おかげで素晴らしい研究生活だけでなく，新たな多くの出会いを経験することができました．これらの出会いは今後の私の人生においてかけがえの無い財産になると思います．

東京大学生産技術研究所池内研究室においては，十分すぎるほどの研究環境を提供して頂くとともに，お忙しい日々を送っていらっしゃる中で時間を割いて様々な助言を与えて下さった池内克史教授に心から感謝致します．また，日頃から私に様々な手助け，温かい励ましを頂いた池内研究室の諸先輩方にはこの場をかりて深く御礼申し上げます．

中西・斎藤研究室の諸先輩方には接する機会は少し減ってしまいましたが，素晴らしい先輩ばかりで中西・斎藤研究室の一員として誇りを持って研究に打ち込むことができました．最後まで温かく見守って頂き本当にありがとうございました．同期のみんなにも，本当にくだらない質問，簡単な質問にいつも答えてくれてありがとう．

また，遠くから常に私の健康を気遣ってくれた両親，大きな心の支えとなってくれた素晴らしい友人達にこの場を借りて御礼を言いたいと思います．

最後に，御指導して頂く機会は少なかったですが，故中西正和教授には生前とてもお世話になりました．心より御冥福をお祈りします．

2000 年冬

参 考 文 献

[1] 浅田 尚紀：カメラキャリブレーション “ コンピュータビジョン， 技術評論と将来展望 第３章, 新技術コミュニケーションズ, 1998

[2] Tsai,R.Y.：A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses.，IEEE Journal of Robotics and Automation 3(4):pp.323-344, August 1987

[3] 松山隆司：Hough 変換とパターンマッチング，情報処理 vol30, no.9, pp1035-1046, 1989

[4] 岩波書店：グラフィックスとマンマシンシステム,1995

[5] 星雲社：Open GL プログラミングガイド第２版 The official Guide to Learnig OpenGL,Version1.1，,1997

付録

外部仕様
内部仕様
ソースプログラム

付録 A 外部仕様

以下に外部仕様を示す．なお，vista 形式のファイルの場合のみ，必ずファイル名は拡張子を v とすること．

A-1　対応点抽出

A-1.1　実行

Linux 上で point_search を実行する．

A-1.2　入力ファイル

- Sample.asc

  金網の形状を示すレンジデータ

  レンジセンサで測定した結果を出力したもの

  金網の各点のレンジセンサを原点とする世界座標 $(X_w,\ Y_w,\ Z_w)$

- Sample.pgm

  金網をカメラで撮影したときの jpg ファイルを pgm ファイルに変換したもの．各ピクセルの値を白黒の濃淡値の 256 段階で示している．

A-1.3　出力ファイル

- point.txt(自動生成)

  金網の交点の世界座標 $(X_w,\ Y_w,\ Z_w)$，それに対応する画像座標 $(x_f,\ y_f)$

- camera.txt(自動生成)

  カメラの仕様に関するデータ

A-2　カメラパラメータの推定の準備1

A-2.1　実行

Linux 上で Create_Pointfile を実行する.

A-2.2　入力ファイル

- point.txt

  A-1.3 で出力したもの

A-2.3　出力ファイル

- point.v

  金網の交点の世界座標 $(X_w,\ Y_w,\ Z_w)$, それに対応する画像座標 $(x_f,\ y_f)$ を vista 形式になおしたもの.

A-3　カメラパラメータの推定の準備2

A-3.1　実行

Linux 上で Create_Camerafile を実行する.

A-3.2　入力ファイル

- camera.txt

  A-1.3 で出力したもの

A-3.3　出力ファイル

- camara.v
  カメラの仕様に関するデータを vista 形式になおしたもの.

A-4　カメラパラメータの推定

A-4.1　実行

Linux 上で vcamcal を実行する.

A-4.2　入力ファイル

- point.v
  A-2.3 で出力したもの
- camera.v
  A-3.3 で出力したもの

A-4.3　出力ファイル

- calib.v
  カメラパラメータ

A-5　テクスチャを貼る準備 1(対象物体の世界座標から対応する画像座標を求める)

A-5.1　実行

Linux 上で check_WtoS を実行する.

A-5.2　入力ファイル

- Sample_range.asc

コンピュータ上で再現したい物体のレンジデータ

物体の各点のレンジセンサを原点とする世界座標 $(X_w,\ Y_w,\ Z_w)$

- calib.v

  A-4.3 で出力したもの

A-5.3　出力ファイル

- Sample_color.txt

  対象物体の世界座標 $(X_w,\ Y_w,\ Z_w)$，それに対応する画像座標 $(x_f,\ y_f)$

A-6　テクスチャを貼る準備 2(レンジデータを正規化する)

A-6.1　実行

Linux 上で point を実行する.

A-6.2　入力ファイル

- Sample_range.asc

  コンピュータ上で再現したい物体のレンジデータ

  物体の各点のレンジセンサを原点とする世界座標 $(X_w,\ Y_w,\ Z_w)$

A-6.3　出力ファイル

- Sample_range.txt

  コンピュータ上で再現したい物体のレンジデータを正規化したもの

A-7　テクスチャを貼る準備3(対象物体のカラー画像から，RGBの値を求める)

A-7.1　実行

Linux 上で readbmp を実行する．

A-7.2　入力ファイル

- color.bmp

  カメラで対象物体を撮影した画像の jpg ファイルから変換したもの

- Sample_color.txt

  A-5.3 で出力したもの

- Sample_range.txt

  A-6.3 で出力したもの

A-7.3　出力ファイル

- final.txt

  対象物体のレンジデータの正規化したものとそれに対応する画像座標の RGB の値

A-8　テクスチャを貼る準備4(レンジデータに三角パッチをあてる)

A-8.1　実行

Linux 上で txt2ply を実行する．

A-8.2　入力ファイル

- Sample_range.txt

A-6.3 で出力したもの

A-8.3　出力ファイル

- ply.txt

  レンジデータの各点に番号をつけ，どの点が三角形を作るかをしめす

A-9　テクスチャを貼る

A-9.1　実行

Linux 上で texture を実行する．

A-9.2　入力ファイル

- final.txt

  A-7.3 で出力したもの

- ply.txt

  A-8.3 で出力したもの

A-9.3　出力

- 対象物体の画像

付録 B 内部仕様

B-1 point_search.c

B-1.1 主な変数・配列

point point[]	金網の各点の世界座標を格納する構造体
box bunkatu[]	左端と上端を格納する構造体
line line[]	求めた直線の傾きと y 切片を格納する構造体
point2 point2	金網の各点の画像座標を格納する構造体

B-1.2 主要関数

int main(void)

引数 なし
返戻値 なし
機能 金網の交点の抽出を開始する

void readData_range(char)

引数 ファイル名
返戻値 なし
機能 レンジデータの x 座標，y 座標のそれぞれ最大値，最小値を求める

int readData_range2(char)

引数 ファイル名
返戻値 読み込んだ点の数
機能 レンジデータの x 座標，y 座標を構造体に格納

int readData_color(char)

引数　ファイル名
返戻値　読み込んだ点の数
機能　カラー画像の白い部分のピクセルの位置を x 座標，y 座標として構造体に格納

`void pow_min(double, double, int)`

引数　直線の傾きと y 切片，レンジデータ数
返戻値　なし
機能　与えられた直線付近の点に対して最小自乗法を行う

`void bunkatu_box(double)`

引数　$\rho - \theta$ 平面における ρ の最大値
返戻値　なし
機能　$\rho - \theta$ 平面の量子化

`void bunkatu_plus(double, int, double)`

引数　ある点を Hough 変換したときの θ，ρ，ρ の最大値
返戻値　なし
機能　量子化されたセルへの投票

`void s_line(int)`

引数　点のデータ数
返戻値　なし
機能　Hough 変換の開始

`int max(int)`

引数　点のデータ数
返戻値　正しく直線が求められているかのチェックを行う値
機能　投票の最も大きいセルから直線の決定

`int line_point(int, int, int, int)`

引数　横方向の直線の数，縦方向の直線の数，直線の総数，点の数
返戻値　求められる交点の数
機能　直線の方程式から交点を決定

B-2 point.c

B-2.1 主な変数・配列

`point point[]` 対象物体のレンジデータを格納する構造体

B-2.2 主要関数

`int main(void)`

引数 入力ファイル名，出力ファイル名
返戻値 なし
機能 対象物体のレンジデータの正規化

B-3 txt2ply.c

B-3.1 主な変数・配列

`point point[]`	対象物体のレンジデータを格納する構造体
`point2 triangle[]`	どの点が三角形を作るかのデータを格納する構造体
`int dummy[1000], dummy2[1000]`	レンジデータを一旦格納する配列

B-3.2 主要関数

`int main(void)`

引数 なし
返戻値 なし
機能 三角形を作る点の番号の決定

B-4 readbmp.c

B-4.1 主な変数・配列

`unsigned char ref_r[]`	読む込む画像のそれぞれのピクセルにおけるRの値を格納する配列
`unsigned char ref_g[]`	読む込む画像のそれぞれのピクセルにおけるGの値を格納する配列
`unsigned char ref_b[]`	読む込む画像のそれぞれのピクセルにおけるBの値を格納する配列
`unsigned char ref[]`	読む込む画像のあるそれぞれのピクセルにおけるRGBのいずれかの値を格納する配列
`int point`	世界座標に対応するピクセルの位置

B-4.2 主要関数

`int main(void)`

引数 なし
返戻値 なし
機能 画像座標のピクセルの位置におけるRGBの値の読み取り

B-5 texture.c

B-5.1 主な変数・配列

`point point`	対象物体の正規化したレンジデータとそれに対応するカラー画像のピクセルの画像座標におけるRGBの値を格納する構造体
`point2 triangles`	三角パッチを作るレンジデータの番号を格納する構造体

B-5.2 主要関数

`int main(int atrc , char** argv)`

引数　入力ファイル
返戻値　なし
機能　対象物体の表示

`void readData(char)`

引数　入力ファイル名
返戻値　なし
機能　ファイルからデータを読みだして，配列に格納

`void triangle(void)`

引数　なし
返戻値　なし
機能　各々の三角形の座標と色を与える

付録 C ソースコード

point_search.c

```
#include "math.h"
#include "stdio.h"

#define MAX 50000
#define MAX2 1000
#define MAX3 3600
#define PI 3.141592
#define SHIKII 100
#define r_width 11
#define c_width 10

/*レンジデータの各点の座標*/
struct point{
  double X;
  double Y;
  double Z;
} point[MAX];

/*ハフ変換のそれぞれの箱*/
struct box{
  double X; //箱の左端
  double Y; //箱の上端
  int sum;
} bunkatu[MAX3][MAX2];

/*直線の傾きと Y 切片*/
struct line{
  double a;
  double b;
  double c;
} line[200];

/*カラーデータの各点の座標*/
struct point2{
  int X;
```

```
  int Y;
} point2[MAX];

struct point point_range[1000];
struct point point_color[1000];
struct line line_color[200];

int all_range, all_color, z = 0, X_co, Y_co;
double x_min, y_min, x_max, y_max,p_max = 0, p_max2 = 0, range, range2;

void make_camera_file(void){

  FILE *fp;

  if((fp = fopen("camera.txt", "w")) == NULL){
    printf("error\n");
    exit(1);
  }

  fprintf(fp, "%s %d %d\n\n", "CameraNSels", 2012 ,1324);
  fprintf(fp, "%s %d %d\n\n", "CameraNPixels", X_co, Y_co);
  fprintf(fp, "%s %f %f\n\n", "CameraSelSize", 23.7 / 2012, 15.6 / 1324);
  fprintf(fp, "%s %d %d\n\n", "CameraImageCenter", X_co / 2, Y_co / 2);
  fprintf(fp, "%s %d\n\n", "CameraScaleX", 1);

  fclose(fp);
}
void readData_range(char *filename) {
  FILE *fp;
  int i;
  float x, y, z;
  char ch, ch2[50];

  if((fp = fopen(filename, "r"))==NULL) {
    printf("Error");
    exit(1);
  }
  for(i = 0; i < 68; i++)
    fscanf(fp,"%s", ch2);

  x_min = 999999.00;
  y_min = 999999.00;
  x_max = -999999.00;
  y_max = -999999.00;

  while((ch = fgetc(fp)) != EOF) {
    fscanf(fp,"%s", ch2);

    fscanf(fp,"%f%f%f", &x, &y, &z);
```

```
    if(x != 999999.00 && y != 999999.00 && z != 999999.0){
      if(x_min > x)x_min = x;
      if(y_min > y)y_min = y;
      if(x_max < x)x_max = x;
      if(y_max < y)y_max = y;
    }
  }
  x_min = x_min - 100;
  y_min = y_min - 100;
  x_max = x_max + 100;
  y_max = y_max + 100;
  fclose(fp);
}/*レンジデータの読み込み 1*/

int readData_range2(char *filename) {
  FILE *fp, *fq;
  float x, y, z;
  int i;
  char ch, ch2[50];

  if((fp = fopen(filename, "r"))==NULL) {
    printf("Error2");
    exit(1);
  }

  if((fq = fopen("point_range1212.txt", "w+"))==NULL) {
    printf("Error2");
    exit(1);
  }

  for(i = 0; i < 68; i++)
    fscanf(fp,"%s", ch2);

  i = 0;

  while((ch = fgetc(fp)) != EOF) {
    fscanf(fp,"%s", ch2);

    fscanf(fp,"%f%f%f", &x, &y, &z);

    if(x != 999999.00 && y != 999999.00 && z != 999999.00) {
      fprintf(fq, "%f %f %f \n", x, y, z);
      point[i].X = x - x_min;
      point[i].Y = y - y_min;
      point[i].Z = z;
      i++;
    }
  }
  fclose(fp);
```

```
  fclose(fq);
  return i;
}/*レンジデータの読み込み 2*/

int readData_color(char *filename) {
  FILE *fp;
  int i = 0, k, x = 0, y = 0, Z;
  char ch, ch2[50];

  if((fp = fopen(filename, "r"))==NULL) {
    printf("Error");
    exit(1);
  }

  for(k = 0; k < 11; k++)
    fscanf(fp,"%s", ch2);

  fscanf(fp,"%d %d", &X_co, &Y_co);
  make_camera_file();

  fscanf(fp,"%s", ch2);

  while((ch = fgetc(fp)) != EOF && i < MAX) {
    fscanf(fp,"%d", &Z);

    if(Z > SHIKII){
      point2[i].X = x;
      point2[i].Y = y;
      i++;
    }

    x++;
    if(x >= X_co){
      x = 0;
      y++;
    }

  }
  fclose(fp);
  return i;
}/*カラーデータ読み込み*/

int partition_a(int m, int n, struct line *line){

  int i, k;
  double pivot;
  struct line tmp;

  pivot = fabs(line[n].a);
```

```
  i = m - 1;
  k = n;

  for(;;){
    while(fabs(line[++i].a) < pivot)
      ;
    while(i < --k && pivot < fabs(line[k].a))
      ;
    if(i >= k)
      break;

    tmp = line[i];
    line[i] = line[k];
    line[k] = tmp;
  }
  tmp = line[i];
  line[i] = line[n];
  line[n] = tmp;

  return i;
}

void quick_sort_a(int m, int n, struct line *line){
  int v;

  if(m >= n)
    return;

  v = partition_a(m, n, line);

  quick_sort_a(m, v - 1, line);

  quick_sort_a(v + 1, n, line);

}


int partition_b(int m, int n, struct line *line){

  int i, k;
  double pivot;
  struct line tmp;

  pivot = line[n].c;

  i = m - 1;
  k = n;
```

```
  for(;;){
    while(line[++i].c > pivot)
      ;
    while(i < --k && pivot > line[k].c)
      ;
    if(i >= k)
      break;

    tmp = line[i];
    line[i] = line[k];
    line[k] = tmp;
  }
  tmp = line[i];
  line[i] = line[n];
  line[n] = tmp;

  return i;
}

void quick_sort_b(int m, int n, struct line *line){
  int v;

  if(m >= n)
    return;

  v = partition_b(m, n, line);

  quick_sort_b(m, v - 1, line);

  quick_sort_b(v + 1, n, line);

}

int partition_c(int m, int n, struct line *line){

  int i, k;
  double pivot;
  struct line tmp;

  pivot = line[n].c;

  i = m - 1;
  k = n;

  for(;;){
    while(line[++i].c < pivot)
      ;
    while(i < --k && pivot < line[k].c)
      ;
```

```
    if(i >= k)
      break;

    tmp = line[i];
    line[i] = line[k];
    line[k] = tmp;
  }
  tmp = line[i];
  line[i] = line[n];
  line[n] = tmp;

  return i;
}

void quick_sort_c(int m, int n, struct line *line){
  int v;

  if(m >= n)
    return;

  v = partition_c(m, n, line);

  quick_sort_c(m, v - 1, line);

  quick_sort_c(v + 1, n, line);

}

int min(int a, int b){
  if(a >= b)
    return(b);
  else
    return(a);
}

void pow_min(double a, double b, int sum){

  int i, N = 0;
  double aa, bb, l, X, Y, x = 0, y = 0, x2 = 0, xy = 0;

  for(i = 0; i < sum; i++){
    l = ((point[i].Y + y_min) - a * (point[i].X + x_min) - b)
      * cos(atan2(a, 1));
    if(fabs(l) < r_width){
      X = point[i].X;
      point[i].X = 99999.0;
      Y = point[i].Y;
      x = x + X;
      y = y + Y;
```

```
      x2 = x2 + pow(X, 2);
      xy = xy + X * Y;
      N++;
    }
  }

  aa = (N * xy - x * y) / (N * x2 - pow(x, 2));
  bb = (x2 * y - x * xy) / (N * x2 - pow(x, 2)) - x_min * aa + y_min;

  line[z].a = aa;
  line[z].b = bb;
  z++;

}/*最小自乗法*/

void pow_min_color(double a, double b, int sum){

  int i, N = 0;
  double aa, bb, l, X, Y, x = 0, y = 0, x2 = 0, xy = 0;

  for(i = 0; i < sum; i++){
    l = (point2[i].Y - a * point2[i].X - b)
      * cos(atan2(a, 1));
    if(fabs(l) < c_width){
      X = point2[i].X;
      point2[i].X = 99999.0;
      Y = point2[i].Y;
      x = x + X;
      y = y + Y;
      x2 = x2 + pow(X, 2);
      xy = xy + X * Y;
      N++;
    }
  }

  aa = (N * xy - x * y) / (N * x2 - pow(x, 2));
  bb = (x2 * y - x * xy) / (N * x2 - pow(x, 2));

  line_color[z].a = aa;
  line_color[z].b = bb;
  z++;

}/*最小自乗法*/

double max_p(int sum){
  int i;
  double p;

  for(i = 0; i < sum; i++){
```

```
    p = sqrt(pow(point[i].X, 2) + pow(point[i].Y, 2));
    if(p_max < p)
      p_max = p;
  }
  return(p_max);
}/*p の最大値を求める*/

double max_p_color(int sum){
  int i;
  double p;

  for(i = 0; i < sum; i++){
    p = sqrt(pow(point2[i].X, 2) + pow(point2[i].Y, 2));
    if(p_max2 < p)
      p_max2 = p;
  }
  return(p_max2);
}/*p の最大値を求める*/

void init_bunkatu(){
  int i, k;

  for(i = 0; i < MAX3; i++)
    for(k = 0; k < MAX2; k++)
      bunkatu[i][k].sum = 0;
}

void bunkatu_plus(double a, int k, double p_max){
  double b, l;
  int m;

  b = p_max - a;
  l = b / range;
  m = floor(l);
  bunkatu[k][m].sum++;
}

void bunkatu_box(double p_max){
  int i, k;

  range = p_max / MAX2;
  range2 = PI * 2 / MAX3;

  for(i = 0; i < MAX2; i++)
    for(k = 0; k < MAX3; k++){
      bunkatu[k][i].X = k  * range2;
      bunkatu[k][i].Y = p_max - i * range;
    }
}/*箱の左上の座標を求める*/
```

```
void s_line(int sum){
  int i, k, n;

  double a, atan, theta, l;

  init_bunkatu();

  for(i = 0; i < sum; i++){
    if(point[i].X < 10000){
      atan = atan2(point[i].Y, point[i].X);
      theta = 180 * atan / PI * 10;
      n = floor(theta);
      if(theta - n >= 0.5)
        n = n + 1;
      for(k = 0; k < n + 901; k++){
        a = point[i].X * cos(k * PI / 1800) +  point[i].Y * sin(k * PI / 1800);
        bunkatu_plus(a, k, p_max);
      }
      for(k = n + 2700; k < MAX3; k++){
        a = point[i].X * cos(k * PI / 1800) + point[i].Y * sin(k * PI / 1800);
        bunkatu_plus(a, k, p_max);
      }
    }
  }
}/*それぞれの箱に通る直線分だけ加算*/

void s_line_color(int sum){
  int i, k, n;

  double a, atan, theta, l;

  init_bunkatu();

  for(i = 0; i < sum; i++)
    if(point2[i].X < 10000){
      atan = atan2(point2[i].Y, point2[i].X);
      theta = 180 * atan / PI * 10;
      n = floor(theta);
      if(theta - n >= 0.5)
        n = n + 1;
      for(k = 0; k < n + 901; k++){
        a = point2[i].X * cos(k * PI / 1800) + point2[i].Y * sin(k * PI / 1800);
        bunkatu_plus(a, k, p_max2);
      }
      for(k = n + 2700; k < MAX3; k++){
        a = point2[i].X * cos(k * PI / 1800) + point2[i].Y * sin(k * PI / 1800);
        bunkatu_plus(a, k, p_max2);
      }
```

```
    }
}/*それぞれの箱に通る直線分だけ加算*/

int max(int sum){
  int i, k, bunkatu_max = 0, i_max = 0, k_max = 0,  check = 0;
  double a, b, l;

  for(i = 0; i < MAX3; i++)
    for(k = 0 ; k < MAX2; k++)
      if(bunkatu_max < bunkatu[i][k].sum){
        bunkatu_max = bunkatu[i][k].sum;
        i_max = i;
        k_max = k;
      }

  if(bunkatu_max < 30){
    printf("return\n");
    return(1);
  }

  if(i_max == 0){
    a = 999999.00;
    b = bunkatu[0][k_max].Y;
    for(i = 0; i < sum; i++){
      l = point[i].X - bunkatu[0][k_max].Y;
      if(fabs(l) < r_width){
        point[i].X = 99999.0;
        check = 1;
      }
    }
  }
  else{
    a = (-1) * cos(bunkatu[i_max][k_max].X) /
      sin(bunkatu[i_max][k_max].X);

    b = (bunkatu[i_max][k_max].Y - range / 2) /
      sin(bunkatu[i_max][k_max].X) - x_min * a + y_min;

    if(fabs(a) < 1 ){
      pow_min(a, b, sum);
      check = 2;
    }
    else{
      for(i = 0; i < sum; i++){
        l = ((point[i].Y + y_min) - a * (point[i].X + x_min) - b)
          * cos(atan2(a, 1));
        if(fabs(l) < r_width){
          point[i].X = 99999.0;
          check = 1;
```

```
}
      }
    }
  }
  if(check == 1){
    line[z].a = a;
    line[z].b = b;
    z++;
    return(0);
  }
  if(check == 2)
    return(0);

  bunkatu[i_max][k_max].sum = 0;
  return(2);

}/*直線を求める*/

int max_color(int sum){
  int i, k, bunkatu_max = 0, i_max = 0, k_max = 0,  check = 0;
  double a, b, l;

  for(i = 0; i < MAX3; i++)
    for(k = 0 ; k < MAX2; k++)
      if(bunkatu_max < bunkatu[i][k].sum){
        bunkatu_max = bunkatu[i][k].sum;
        i_max = i;
        k_max = k;
      }

  if(bunkatu_max < 30){
    printf("return\n");
    return(1);
  }

  if(i_max == 0){
    a = 999999.00;
    b = bunkatu[0][k_max].Y;
    for(i = 0; i < sum; i++){
      l = point2[i].X - bunkatu[0][k_max].Y;
      if(fabs(l) < c_width){
        point2[i].X = 99999.0;
        check = 1;
      }
    }
  }
  else{
    a = (-1) * cos(bunkatu[i_max][k_max].X) /
      sin(bunkatu[i_max][k_max].X);
```

```
    b = (bunkatu[i_max][k_max].Y - range / 2) /
      sin(bunkatu[i_max][k_max].X);

    if(fabs(a) < 1 ){
      pow_min_color(a, b, sum);
      check = 2;
    }
    else{
      for(i = 0; i < sum; i++){
        l = (point2[i].Y - a * point2[i].X - b)
          * cos(atan2(a, 1));
        if(fabs(l) < c_width){
          point2[i].X = 99999.0;
          check = 1;
}
      }
    }
  }

  if(check == 1){
    line_color[z].a = a;
    line_color[z].b = b;
    printf("[%d] %f %f\n", z, line_color[z].a, line_color[z].b);
    z++;
    return(0);
  }

  if(check == 2)
    return(0);

  bunkatu[i_max][k_max].sum = 0;
  return(2);
}

int line_point(int small, int big, int range_small, int sum, int num1){
  int i, k, l, l_min = 0;
  double x, y, xy, xy_min;

  for(i = 0; i < small; i++)
    for(k = range_small; k < range_small + big; k++){
      if(line[i].a < 9999.00 && line[k].a < 9999.00){
        xy_min = 999999.00;
        x = (line[i].b - line[k].b) / (line[k].a - line[i].a);
        y = line[i].a * x + line[i].b;
        for(l = 0; l < sum; l++){
          xy = pow(((point[l].X + x_min) - x), 2) + pow(((point[l].Y + y_min) - y), 2);
          if(xy < xy_min){
            xy_min = xy;
```

```
            l_min = l;
          }
          point_range[num1].X = point[l_min].X + x_min;
          point_range[num1].Y = point[l_min].Y + y_min;
          point_range[num1].Z = point[l_min].Z;
        }
      }
      num1++;
    }
  return(num1);
}/*交点を求める*/

int line_point_color(int small, int big, int color_small, int num2){
  int i, k, l, l_min = 0;
  double x, y, xy, xy_min;
  for(i = 0; i < small; i++)
    for(k = color_small; k < color_small + big; k++){
      if(line_color[i].a < 9999.00 && line_color[k].a > 99999.00){
        point_color[num2].X = line_color[i].b;
        point_color[num2].Y = line_color[k].b;
        num2++;
      }

      if(line_color[i].a < 9999.00 && line_color[k].a < 9999.00){
        x = (line_color[i].b - line_color[k].b) / (line_color[k].a - line_color[i].a);
        y = line_color[i].a * x + line_color[i].b;
        point_color[num2].X = x;
        point_color[num2].Y = y;
        num2++;
      }
    }
  return(num2);
}


int main(void) {
  int i, k, l, m = 0, n, o, range = 0,
    color = 0, range_sum, color_sum, range_num, color_num,
    range_min, color_min, big_min, small_min, pointrange, pointcolor;
  double p;
  char range_file[10][30], color_file[10][30];
  FILE *fp, *fq;

  printf("?plane?\n");
  scanf("%d", &l);
  for(i = 0; i < l; i++){
    printf("input rangedata[%d](.asc):\n", i + 1);
    scanf("%s", range_file[i]);
    printf("input colordata[%d](.pgm):\n", i + 1);
```

```
  scanf("%s", color_file[i]);
}

for(n = 0; n < l; n++){
  x_min = 99999.00;
  y_min = 99999.00;
}

/*レンジデータ処理*/
readData_range(range_file[n]);
range_sum = readData_range2(range_file[n]);
p = max_p(range_sum);
bunkatu_box(p);
z = 0;

for(;;){
  s_line(range_sum);
  o = max(range_sum);
  if(o == 2)
    for(;;){
      o = max(range_sum);
      if(o == 1)
        break;
    }
  if(o == 1)
    break;
}

quick_sort_a(0, z - 1, line);
/*それぞれの直線を傾きの絶対値で昇順にソート*/

i = 0;

while(fabs(line[i].a) < 1)
  i++;

quick_sort_b(0, i - 1, line);/*降順にソート*/

for(k = i; k < z; k++){
  if(line[k].a != 999999.00)
    line[k].c = (((y_max - y_min) / 2) - line[k].b) / line[k].a;
  else
    line[k].c = line[k].b;
}

quick_sort_c(i, z - 1, line); /*昇順にソート*/

range_min = i;/*距離画像中の傾きの小さい直線の総数*/
```

```
range_num = z; /*距離画像中の直線の総数*/
z = 0;

/*カラーデータ処理*/
color_sum = readData_color(color_file[n]);
p = max_p_color(color_sum);
bunkatu_box(p);

for(;;){
  s_line_color(color_sum);
  o = max_color(color_sum);
  if(o == 2)
    for(;;){
      o = max_color(color_sum);
      if(o == 1)
        break;
    }
  if(o == 1)
    break;
}

quick_sort_a(0, z - 1, line_color);
/*それぞれの直線を傾きの絶対値で昇順にソート*/

i = 0;
while(fabs(line_color[i].a) < 1)
  i++;

for(k = 0; k < i; k++)
  line_color[k].c = X_co * line_color[k].a + line_color[k].b;

quick_sort_c(0, i - 1, line_color);

for(k = i; k < z; k++){
  if(line_color[k].a != 999999.00)
    line_color[k].c = (Y_co - line_color[k].b) / line_color[k].a;
  else
    line_color[k].c = line_color[k].b;
}

quick_sort_c(i, z - 1, line_color);

color_min = i; /*カラー画像中の傾きの小さい直線の総数*/

color_num = z; /*カラー画像中の直線の総数*/

big_min = min(range_num - range_min, color_num - range_min);
/*傾きの大きい直線の総数*/
```

```
  small_min = min(range_min, color_min); /*傾きの小さい直線の総数*/

  readData_range2(range_file[n]);
  pointrange = line_point(small_min, big_min, range_min, range_sum, range);
  /*距離画像の交点を求める*/
  pointcolor = line_point_color(small_min, big_min, color_min, color);
  /*カラー画像の交点を求める*/

  range = pointrange;
  color = pointcolor;

  if((fp = fopen("point.txt", "w")) == NULL){
    printf("error\n");
    exit(1);
  }

  /*ファイル出力*/
  fprintf(fp, "%s %d\n", "point" , range);
  for(i = 0; i < range; i++)
    fprintf(fp, "%.2f %.2f %.2f %f %f\n",
    point_range[i].X, point_range[i].Y, point_range[i].Z,
    point_color[i].X, point_color[i].Y);
  printf("point.txt & camera.txt out\n");
  fclose(fp);
}
```

point.c

```
#include "math.h"
#include "stdio.h"

#define MAX 50000

/*各点の座標*/
struct point{
  double x;
  double y;
  double z;
} point[MAX];

int main(argc, argv)
int argc;
char *argv[];
{
  FILE *fp, *fq;
  float x, y, z;
```

```
int i = 0, j;
char ch, ch2[50];
double x_min, y_min, z_min, z_max, x_max, y_max;

/*ファイルの読み込み*/
if( argc != 3 ){
  fprintf(stderr,"usage: point asc_file outfile\n");
  exit(1);
}

if((fp = fopen(argv[1], "r"))==NULL) {
  printf("Error2");
  exit(1);
}

if((fq = fopen(argv[2], "w+"))==NULL) {
  printf("Error2");
  exit(1);
}

for(i = 0; i < 68; i++)
  fscanf(fp,"%s", ch2);

x_min = 999999.00;
y_min = 999999.00;
z_min = 999999.00;
z_max = -999999.00;
x_max = -999999.00;
y_max = -999999.00;

while((ch = fgetc(fp)) != EOF) {
  fscanf(fp,"%s", ch2);

  fscanf(fp,"%f%f%f", &x, &y, &z);

  if(x != 999999.00 && y != 999999.00 && z != 999999.00){
    if(x_min > x)x_min = x;
    if(y_min > y)y_min = y;
    if(x_max < x)x_max = x;
    if(y_max < y)y_max = y;
    if(z_min > z)z_min = z;
    if(z_max < z)z_max = z;
    point[i].x = x;
    point[i].y = y;
    point[i].z = z;
    i++;
  }
}
```

```
  x_min = x_max - x_min;
  y_min = y_max - y_min;
  z_min = z_max - z_min;

  if(x_min > y_min){
    if(x_min > z_min){
      y_min = x_min;
      z_min = x_min;
    }
    else{
      y_min = z_min;
      x_min = z_min;
    }
  }
  else{
    if(y_min > z_min){
      x_min = y_min;
      z_min = y_min;
    }
    else{
      y_min = z_min;
      x_min = z_min;
    }
  }

  for(j = 0; j < i; j++){
    x = (x_max - point[j].x) / x_min;
    y = (y_max - point[j].y) / y_min;
    z = (z_max - point[j].z) / z_min;
    fprintf(fq, "%f %f %f %d\n", x, y, z, a);
  }
  fclose(fp);
  fclose(fp);

  return 0;
}/*asc ファイルから必要な点だけ抽出し正規化*/
```

readbmp.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*BITMAP FILE HEADER の宣言*/
typedef struct tagBITMAPFILEHEADER {

  unsigned short bfType;
```

```
  unsigned long  bfSize;
  unsigned short bfReserved1;
  unsigned short bfReserved2;
  unsigned long  bfOffBits;

}BITMAPFILEHEADER;

/*BITMAP INFO HEADERの宣言*/
typedef struct tagBITMAPINFOHEADER{

  unsigned long  biSize;
  signed long    biWidth;
  signed long    biHeight;
  unsigned short biPlanes;
  unsigned short biBitCount;
  unsigned long  biCompression;
  unsigned long  biSizeImage;
  signed long    biXPelsPerMeter;
  signed long    biYPelsPerMeter;
  unsigned long  biClrUsed;
  unsigned long  biClrImportant;

} BITMAPINFOHEADER;

int main(argc, argv)
int argc;
char *argv[];
{
  BITMAPFILEHEADER fileheader;
  BITMAPINFOHEADER infoheader;

  FILE *fp_ref;
  FILE *fp_point;
  FILE *fp_color;
  FILE *fp_final;
  int i=0;
  int k=0;
  int point;
  int x_color_int, y_color_int;
  float x_color, y_color;
  float x_range, y_range, z_range;
  char ch;

  unsigned char *ref_r;
  unsigned char *ref_g;
  unsigned char *ref_b;
  unsigned char *ref;

  ref_r=(unsigned char*)malloc(sizeof(unsigned char)*1024*672);
```

```
ref_g=(unsigned char*)malloc(sizeof(unsigned char)*1024*672);
ref_b=(unsigned char*)malloc(sizeof(unsigned char)*1024*672);
ref=(unsigned char*)malloc(sizeof(unsigned char)*1024*672*3);

if( argc != 5 ){
  fprintf(stderr,"usage: readbmp bmp_file color_point_file seikika_file outfile\n");
  exit(1);
}

/*写真の読み込みの開始*/
if((fp_ref=fopen(argv[1],"rb"))==NULL){
  printf("File open error 1 \n");
  return 0;
}

/*写真のヘッダーの読み込み*/
if(fread(&fileheader.bfType,2,1,fp_ref)!=1)
  printf("read error1");

if(fread(&fileheader.bfSize,4,1,fp_ref)!=1)
  printf("read error2");

if(fread(&fileheader.bfReserved1,2,1,fp_ref)!=1)
  printf("read error3");

if(fread(&fileheader.bfReserved2,2,1,fp_ref)!=1)
  printf("read error4");

if(fread(&fileheader.bfOffBits,4,1,fp_ref)!=1)
  printf("read error5");

if(fread(&infoheader.biSize,4,1,fp_ref)!=1)
  printf("read error6");

if(fread(&infoheader.biWidth,4,1,fp_ref)!=1)
  printf("read error7");

if(fread(&infoheader.biHeight,4,1,fp_ref)!=1)
  printf("read error8");

if(fread(&infoheader.biPlanes,2,1,fp_ref)!=1)
  printf("read error9");

if(fread(&infoheader.biBitCount,2,1,fp_ref)!=1)
  printf("read error10");

if(fread(&infoheader.biCompression,4,1,fp_ref)!=1)
  printf("read error11");
```

```
if(fread(&infoheader.biSizeImage,4,1,fp_ref)!=1)
  printf("read error12");

if(fread(&infoheader.biXPelsPerMeter,4,1,fp_ref)!=1)
  printf("read error13");

if(fread(&infoheader.biYPelsPerMeter,4,1,fp_ref)!=1)
  printf("read error14");

if(fread(&infoheader.biClrUsed,4,1,fp_ref)!=1)
  printf("read error15");

if(fread(&infoheader.biClrImportant,4,1,fp_ref)!=1)
  printf("read error16");

/*写真の中身の読み込み*/
while((fread(&ref[i],sizeof(unsigned char),1,fp_ref))==1)
  i++;

i = 0;

for(k = 0;k < 1024 * 672;k++){
  ref_b[k]=ref[i];
  i++;
  ref_g[k]=ref[i];
  i++;
  ref_r[k]=ref[i];
  i++;
}

if((fp_color = fopen(argv[2], "r")) == NULL){
  printf("FILE ERROR\n");
  return 0;
}/*レンジデータと対応する画像上の点の座標の読み込み*/

if((fp_point = fopen(argv[3], "r")) == NULL){
  printf("FILE ERROR\n");
  return 0;
}/*正規化したレンジデータの読み込み*/

if((fp_final = fopen(argv[4], "w+")) == NULL){
  printf("FILE ERROR\n");
  return 0;
}

/*出力用ファイルへ書き込み*/
while((ch = fgetc(fp_color)) != EOF) {
  fscanf(fp_color, "%f %f", &x_color, &y_color);
  fscanf(fp_point, "%f %f %f", &x_range, &y_range, &z_range);
```

```
    x_color_int = floor(x_color);
    y_color_int = floor(y_color);

    point = (672 - y_color_int) * 1024 + x_color_int;
    /*左上基準から左下基準に*/

    if(x_color_int >= 0 && y_color_int >= 0 &&
       x_color_int < 1024 && y_color_int < 672 ){
      fprintf(fp_final,"%f ", x_range);
      fprintf(fp_final,"%f ", y_range);
      fprintf(fp_final,"%f ", z_range);
      fprintf(fp_final,"%u ", ref_r[point]);
      fprintf(fp_final,"%u ", ref_g[point]);
      fprintf(fp_final,"%u \n", ref_b[point]);
    }
  }

  fclose(fp_ref);
  fclose(fp_point);
  fclose(fp_color);
  fclose(fp_final);

  return 0;
}
```

txt2ply.c

```
#include "math.h"
#include "stdio.h"

#define MAX 50000

/*各点の座標*/
struct point{
  double x;
  double y;
  double z;
  int a;
} point[MAX];

struct point2{
  int a;
  int b;
  int c;
} triangle[MAX];

int main(argc, argv)
```

```
int argc;
char *argv[];
{
  FILE *fp, *fq;
  float x, y, z;
  int i = 0, j, k, num, l = 0, count = 0,
    dummy[1000], dummy2[1000], dummy_num, dummy_num2, a, red ,blue, green;
  char ch;

  if( argc != 3 ){
    fprintf(stderr,"usage: point txt_file ply_file\n");
    exit(1);
  }

  if((fp = fopen(argv[1], "r"))==NULL) {
    printf("Error2");
    exit(1);
  }

  if((fq = fopen(argv[2], "w+"))==NULL) {
    printf("Error2");
    exit(1);
  }

  while((ch = fgetc(fp)) != EOF) {
    fscanf(fp,"%f%f%f%d%d%d", &x, &y, &z, &red, &blue, &green);
    point[i].x = x;
    point[i].y = y;
    point[i].z = z;
    i++;
  }

  for(j = 0; j < i; j++){
    x = point[j].x;
    y = point[j].y;
    z = point[j].z;
  }
  num = j;

  i = 0;

  while(l < num - 10){
    for(a = 0; a < 1000; a++){
      dummy[a] = 0;
      dummy2[a] = 0;
    }
    j = 0;
    do{
      dummy[j] = l;
```

```
  l++;
  j++;
}while(point[l + 1].x > point[l].x);

dummy[j] = l;

l++;
dummy_num = j + 1;
j = 0;

do{
  dummy2[j] = l;
  l++;
  j++;
}while(point[l + 1].x > point[l].x);
dummy2[j] = l;
dummy_num2 = j + 1;

i = 0;
k = 0;

/*1 つ目の点列のほうが点の数が少ないとき*/
if(dummy_num < dummy_num2){
  while(i < dummy_num && k < dummy_num2 - 1){
    triangle[count].a = dummy[i];
    triangle[count].b = dummy2[k];
    triangle[count].c = dummy2[k + 1];
    count++;
    k++;
    while(point[dummy[i]].x > point[dummy2[k]].x){
      triangle[count].a = dummy[i];
      triangle[count].b = dummy2[k];
      triangle[count].c = dummy2[k + 1];
      k++;
      count++;
    }
    if(i < dummy_num - 1){
      triangle[count].a = dummy[i];
      triangle[count].b = dummy[i + 1];
      triangle[count].c = dummy2[k];
      count++;
    }
    i++;
  }
  while(k < dummy_num2 - 1){
    triangle[count].a = dummy[i - 1];
    triangle[count].b = dummy2[k];
    triangle[count].c = dummy2[k + 1];
    count++;
```

```
        k++;
      }
    }

    /*2 つ目の点列のほうが点の数が少ないとき*/
    else{
      while(k < dummy_num2 - 1){
        triangle[count].a = dummy[i];
        triangle[count].b = dummy[i + 1];
        triangle[count].c = dummy2[k];
        count++;
        i++;
        while(point[dummy[i]].x > point[dummy2[k]].x){
          triangle[count].a = dummy[i];
          triangle[count].b = dummy2[k];
          triangle[count].c = dummy2[k + 1];
          k++;
          count++;
        }
        triangle[count].a = dummy2[k];
        triangle[count].b = dummy2[k + 1];
        triangle[count].c = dummy[i];
        k++;
        count++;
      }
      while(i < dummy_num - 1){
        triangle[count].a = dummy[i];
        triangle[count].b = dummy[i + 1];
        triangle[count].c = dummy2[k];
        count++;
        i++;
      }
    }
    l = l + 1 - dummy_num2;
  }
  a = count;
  /*ファイルへ出力*/
  for(count = 0; count < a - 10000; count++)
    fprintf(fq, "%d %d %d \n",
    triangle[count].a, triangle[count].b,
    triangle[count].c);
  fclose(fp);
  fclose(fq);

  return 0;
}
```

texture.c

```
#include <GL/glut.h>
#include <stdlib.h>

#include "stdio.h"

#define MAX 40000

struct point{
  double x;
  double y;
  double z;
  double red;
  double green;
  double blue;
} point[MAX];

struct point2{
  int a;
  int b;
  int c;
} triangles[MAX];

void readData(char *filename)
{
  FILE *fp, *fq;
  int i = 0, j, red, green, blue, a, b, c;
  char ch;
  float x = 0, y = 0, z = 0;

  if((fp = fopen(filename, "r"))==NULL){
    printf("Error");
    exit(1);
  }

  if((fq = fopen("tmp1214.txt", "r"))==NULL){
    printf("Error");
    exit(1);
  }

  while((ch = fgetc(fp)) != EOF){
    fscanf(fp,"%f%f%f%d%d%d", &x, &y, &z, &red, &green, &blue);
    point[i].x = x;
    point[i].y = y;
    point[i].z = z;
    point[i].red = red / 255.0;
    point[i].green = green / 255.0;
    point[i].blue = blue / 255.0;
```

```
    i++;
  }

  i = 0;

  while((ch = fgetc(fq)) != EOF){
    fscanf(fq,"%d%d%d", &a, &b, &c);
    printf("%d %d %d\n", a, b, c);

    triangles[i].a = a;
    triangles[i].b = b;
    triangles[i].c = c;
    i++;
  }
  triangles[i].a = 99999;
  triangles[i].b = 99999;
  triangles[i].c = 99999;

  fclose(fp);
  fclose(fq);
}

void init(void)
{
   glClearColor (1.0, 1.0, 1.0, 0.0);
   glShadeModel (GL_SMOOTH);
}

void triangle(void)
{
  int i = 0, j = 0;
  float l, m, n;

  glPushMatrix();
  glTranslatef(1.0, 0.0, 0.0);
  glRotatef((GLfloat)0, 0.0, 1.0, 0.0);

  while(triangles[j].a != 99999 &&
        triangles[j].b != 99999 &&
        triangles[j].c != 99999){
    /*三角形の頂点の座標と色の入力*/
    glBegin (GL_TRIANGLES);
    glColor3f (point[triangles[j].a].red,
               point[triangles[j].a].green,
               point[triangles[j].a].blue);
    glVertex3f (1 - point[triangles[j].a].x,
                0.5 - point[triangles[j].a].y,
                point[triangles[j].a].z);
    glColor3f (point[triangles[j].b].red,
```

```
                point[triangles[j].b].green,
                point[triangles[j].b].blue);
    glVertex3f (1 - point[triangles[j].b].x,
                0.5 - point[triangles[j].b].y,
                point[triangles[j].b].z);
    glColor3f (point[triangles[j].c].red,
                point[triangles[j].c].green,
                point[triangles[j].c].blue);
    glVertex3f (1 - point[triangles[j].c].x,
                0.5 - point[triangles[j].c].y,
                point[triangles[j].c].z);
    j++;
    glEnd();
  }
  glPopMatrix();
  glutSwapBuffers();
}

void display(void){
   glClear (GL_COLOR_BUFFER_BIT);
   triangle ();
   glFlush ();
}

void reshape (int w, int h){
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   if (w <= h)
     gluOrtho2D (0.0, 1.0, 0.0, 1.0 * (GLfloat) h/(GLfloat) w);

   else
     gluOrtho2D (0.0, 1.0 * (GLfloat) w/(GLfloat) h, 0.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
}

void keyboard(unsigned char key, int x, int y){
  switch (key) {
  case 27:
    exit(0);
    break;
  }
}

int main(int argc, char** argv){
  char filename[30];
  int s;

  if( argc != 2 ){
```

```
    fprintf(stderr,"usage: ./smooth inputfile rangefile\n");
    exit(1);
  }
  strcpy (filename, argv[1]);
  readData(filename);
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (700, 700);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (argv[0]);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc (keyboard);
  glutMainLoop();
  return 0;
}
```