

修士論文

複数視点からのステレオ視による
インクリメンタルモデリングと物体認識

平成12年1月28日提出

指導教官 井上博允 教授

東京大学 大学院 工学系研究科 情報工学専攻

86872 佐川 立昌

目次

1	序論	6
1.1	ステレオ視を用いた距離計測	8
1.2	距離画像からの表面形状モデリング	10
1.3	表面形状のマッチング	12
1.4	本論文の構成	14
2	距離画像生成	15
2.1	カメラモデル	16
2.1.1	レンズの歪みの補正	16
2.1.2	ピンホールカメラパラメータの同定	17
2.1.3	カメラ対の平行等位化	20
2.1.4	ステレオカメラモデル	21
2.2	ステレオ視による視差画像生成	22
2.3	再帰相関演算を用いた対応点探索	23
2.3.1	相関演算	23
2.3.2	再帰相関演算アルゴリズム	25
2.3.3	再帰相関演算アルゴリズムによる対応点探索	25
2.3.4	キャッシュを意識したアルゴリズムの展開	26
2.3.5	MMX 命令を用いた実装による高速化	27
2.4	一貫性評価法による信頼度評価	28
2.4.1	一貫性評価法 (Consistency Checking)	28
2.4.2	LR-check の再帰相関演算への組み込み	29
2.5	ロボット搭載用視差画像生成システム	30
2.5.1	ロボット搭載用システムの開発	30
2.5.2	視差画像生成システムの性能評価	30
2.6	多数の画像を用いたステレオ視	35
3	3次元表面形状モデリング	38
3.1	Marching Cubes Algorithm と符号付き距離法	39
3.1.1	Marching Cubes Algorithm	39
3.1.2	符号付き距離の計算	43
3.1.3	複数の距離画像からの符号付き距離計算	44

3.2	アルゴリズムの高速化	45
3.2.1	octree を用いた階層的な体積表現	45
3.2.2	符号付き距離を計算するボクセルの選択	46
3.2.3	画像面の傾きによる閾値処理	48
3.2.4	インクリメンタルアルゴリズム	49
3.2.5	アルゴリズムの考察	51
3.3	実験	53
3.3.1	人工距離画像の生成	53
3.3.2	仮想環境を用いた実験	53
3.3.3	実画像を用いた実験	55
4	3次元物体認識	63
4.1	スピンイメージ	64
4.1.1	位置姿勢に独立な座標系への変換	64
4.1.2	スピンイメージの生成	65
4.1.3	スピンイメージの比較	67
4.2	スピンイメージを用いた認識アルゴリズム	69
4.2.1	スピンイメージマッチング	69
4.2.2	幾何拘束によるフィルタリングとグループ化	71
4.2.3	位置姿勢パラメータの導出	73
4.3	階層メッシュモデルを用いたマッチング	74
4.3.1	メッシュ解像度の選択	74
4.3.2	複数のメッシュ解像度における認識アルゴリズム	75
4.4	実験	76
5	結論	82
5.1	ステレオ視による距離画像生成	83
5.2	3次元表面形状モデリング	84
5.3	3次元物体認識	84
5.4	3次元観測認識システム	85

謝辞	85
A Marching Cubes Algorithm	88
B Quaternion を用いた変換パラメータの解析的解法	95
B.1 Quaternion による回転の表現	96
B.2 変換パラメータの解析的解法	96
参考文献	97

第 1 章

序論

知能ロボットが未知の環境において行動するには，未知の環境（シーン）についての情報を収集して観測対象をモデリングし，すでに持っている知識を表したモデルと比較することによりシーンを認識することが必要である．ロボットが「知的」に行動するためにはロボット自身により，情報の収集（センシング），モデリング，認識（マッチング）を行なうべきである．本論文では，この3つの要素（センシング，モデリング，マッチング）を1つにまとめて，実環境を観測して観測対象のモデルを生成し，得られたモデルを用いて新たに観測したシーンとマッチングすることによって認識を行なうシステムを提案する．

実環境中の物体の認識の用いる特徴には様々考えられるが，ロボットが認識結果を利用して，行動に反映させるためには3次元についての情報が必要である．[1, 2, 3]では単眼のカメラを用いて観測対象の輪郭を抽出して3次元物体の認識を行なっている．一方，Sumiら[4]は物体の輪郭をステレオ視を用いて距離計測し，得られた3次元の点のモデルを用いてマッチングを行なっている．また，Wheeler[5]はレーザーレンジセンサを用いて距離計測してメッシュモデルを構築し，メッシュモデルと新たに観測したシーンの距離画像のマッチングを行なった．同様に[6, 7, 8]においても距離画像から自由形状の物体に対して認識を行なっている．そこで本論文でも同様に静止した実環境の観測対象に対してセンシング，モデリング，マッチングに用いる特徴として観測対象の形状を扱う．

物体の形状はサンプリングされた表面上の点の位置を計測することによって行なわれる．非接触に表面上の点の位置を計測する方法では，離れた基準点から距離計測して，得られた基準点からの相対位置を利用して位置を計算する手法がある．本論文ではその1つの，視覚を用いた距離計測法によって表面形状の計測を行なう．視覚による距離計測法にはレーザーレンジファインダ，光投影法などのレーザーや可視光を投射してその反射を観測することによって距離を測る方法や，複数のカメラを用いたステレオ視による距離計測方法がある．本論文では以下のような理由からステレオ視による距離計測を採用した．まず，カメラのみを用いることからパッシブなセンシングである．また，カメラは他の様々な光学的特徴が観測できる汎用なセンサであるので，他の特徴の観測にも共用可能である．さらに，ロボットに搭載することを考えるとロボットの移動などから発生する振動に強いセンサであることが必要である．レーザーレンジファインダでは，鏡を振ってスキャンするためにカメラと比べて振動に弱い欠点がある．

物体の形状を表現する方法にはパラメータを用いて表現する方法と表面を表す

メッシュを用いた方法がある．前者には一般化円筒 [9] や超二次曲面 [10]，超楕円 [11] を用いた手法などがある．これらの特徴は多くのパラメータを持った基本要素を少数用いてモデリングすることにより，必要記憶量や計算量を小さくし，またノイズを軽減できることである．これに対して後者のメッシュを用いてモデリングする方法は少ないパラメータを持った基本要素を多数用いたモデリングであるため多くの記憶量，計算量を必要とし，以前は計算機の性能の低さから用いられなかった．しかし，メッシュを用いてモデリングする方法は以下に説明するような利点を持つ．まず，メッシュの大きさを変えることによって任意の複雑さの形状を表現することが可能である．また，センサによって得られたデータから直接モデルを構築することができる．さらに，メッシュによるモデルは3次元コンピュータグラフィックスを用いて容易に視覚的に表示できる．近年の計算機の発展によって大量のデータを扱うことが可能になり，メッシュによる表面形状の表現を用いることが可能になってきた．本論文では以上の理由からメッシュを用いた表面形状のモデリングを行なう．

観測対象を区別するためには，観測対象についての重心，体積などの大域的な情報を用いることが有効であるが，物体が混在し観測対象が部分的に見えない場合には大域的な情報を獲得することは困難である．そのような場合には，観測対象の局所的な情報を用いてマッチングすることが有効である．一方，局所的な情報は観測の誤差に弱い欠点があるので，マッチングに用いる情報に含まれる大域性と局所性を調節することが重要である．メッシュモデルは多くの基本要素(点，辺，法線)から構成されるためモデルを部分的に分割することが容易であり，大域性と局所性を調節することが可能である．したがって，メッシュモデルを用いることはマッチングにおいても利点を持つ．

以上から，本論文では次の3つの処理からなるシステムを提案する．まず，複数のカメラを用いたステレオ視により距離計測する．次に距離計測結果を用いて表面形状をメッシュを用いてモデリングする．最後に得られたメッシュモデル同士をマッチングし物体認識を行なう(図.1.1)．

1.1 ステレオ視を用いた距離計測

これまで超音波センサ，パターン投影ステレオ法，ステレオ画像，レーザーレンジセンサ等を用いたさまざまな距離計測法の研究が行われてきたが，解像度，速度，対振動性といった問題点から，実際にロボットに搭載することの可能な適当な

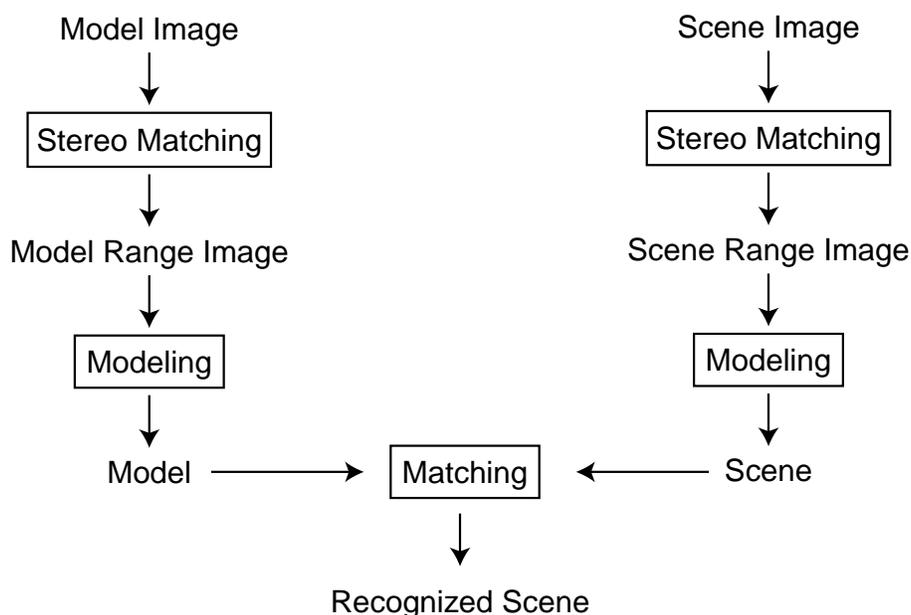


図 1.1: Block diagram of this system

市販の視差画像生成システムは存在しなかった。

一方、画像処理の研究分野ではステレオ視の研究が進み、様々なマッチング手法や精度を向上のための手法の研究が行われ [12]、また専用ハードウェアを用いたものでは優れた性能を示しているものが存在する [13, 14]。

また、近年汎用 CPU がマルチメディア命令と呼ばれる一命令で複数のデータ処理を行う機能を備え、DSP や 画像処理プロセッサよりも高速に画像処理を行うことが可能になってきた。

本論文ではこのような背景をふまえ、画像処理の分野で行われている再帰相関演算手法を用いて視差画像生成アルゴリズムを高速化し、最も普及している Pentium プロセッサの MMX 命令を用いて最適化し、市販の PC 部品のみを組み合わせることにより実時間で視差画像を得ることが可能であることを示す。

これまでステレオ視についての多くの研究がなされているが、本論文では画像のウィンドウ領域の相関演算による対応点探索を行ない、密な視差画像 (図.1.2) を生成する。相関演算には大きい計算量を必要とするが、再帰相関演算手法 [15] を用いてアルゴリズムの効率化を行ない、また、実装においてプロセッサのマルチメディア命令を用いて高速化し、実時間視差画像生成を行なう [16]。



図 1.2: Camera image and dense depthmap

1.2 距離画像からの表面形状モデリング

距離画像を用いて観測対象の表面上の3次元での位置を計算し、画像中で隣り合う点どうしをつなぐことにより、観測対象の表面をメッシュを用いたモデル(距離画像面 range surface)を構築できる。このようにして得られたモデルは、1つの視点からの観測できる部分の形状しかモデリングすることができない。観測対象の全体の形状を観測するためには多くの場合、複数の視点から観測し、それぞれから得られる形状データを統合する処理が必要となる。観測から得られるデータにはノイズが含まれるため、そのノイズに対してロバストな統合を行なうことが重要である。

近年、コンピュータビジョン、コンピュータグラフィックスの分野では実世界の物体の形状をモデリングする研究が盛んに行なわれている。それらの研究の中で複数の距離画像から物体の表面形状をモデリングしているものには次のようなものがある。Turk と Levoy[17] は、複数の距離画像面の境界を接続 (zippering) することによって統一したモデルを生成した。Rutishauser ら [18] はセンサの誤差をモデリングして、冗長に観測された表面部分のメッシュを生成し直すことによりメッシュを統合した。

これに対して、3次元空間中に固定された立方体(ボクセル)を設定し、ボクセルを基準に用いた表現(体積表現 volumetric representation)によってメッシュを生成している研究がある。Conolly[19] と Chien ら [20] は octree を用いてボクセルで表

された空間を2状態で表し、距離画像から計算することによってモデリングした。しかし、これらの研究では物体の表面を生成することは行なっていない。

ボクセルの状態を2値化する手法に対して、Lorenzenらが提案した Marching Cubes Algorithm[21]はボクセル内部にメッシュで表された表面を生成して表面形状をモデリングする。ボクセルの8つの頂点にスカラー値のデータを与えることによってボクセル内部のメッシュの形状を決定するものである。

表面形状のモデリングを行なう研究の中には、Marching Cubes Algorithmと距離画像を組み合わせて表面形状のモデリングを行なっている次のような研究がある。Hoppeら[22]はボクセルの頂点から距離画像面までの最短距離を計算し、物体の外部か内部かによって符号を付加した符号付き距離 (signed distance) を Marching Cubes Algorithm に用いるスカラー値として与えて表面形状のモデリングを行なった。Hoppeらの手法は符号付き距離の計算において、距離画像面の頂点の集合から接平面を推定して最短距離を計算しているために曲率が高い部分では不正確になり、またノイズに弱い欠点を持っていた。

Hiltonら[23]は距離画像面を三角形のメッシュで表し、ボクセルの頂点からメッシュまでの最短距離を符号付き距離とした。Wheelerら[24]は同様に複数の距離画像面について符号付き距離をメッシュまでの最短距離で計算し、得られた符号付き距離の中でもっともらしい値を選ぶことで、ロバスト性を高めた。

CurlessとLevoy[25]はボクセルの頂点から距離画像面までの最短距離を探索する代わりに、カメラからの視線方向に沿って距離画像面までの距離を符号付き距離として用いた。この手法は最短距離の探索を行なう必要がなく、複数の距離画像から得られる符号付き距離の重み付き平均を最終的な符号付き距離とするためにインクリメンタルな符号付き距離の更新が可能である。

ロボットに搭載された視覚の場合、全ての画像をあらかじめ持っているわけではなく、ロボットの行動とともに画像を獲得する。したがって、全ての画像が得られて初めてモデリングできるのではなく、ロボットの行動とともに新しい画像を獲得しモデルをインクリメンタルに更新できることが重要である。よって本論文では、インクリメンタルな符号付き距離の更新が可能な Curless と Levoy の手法を応用してモデリングを行なう。

また、ロボット用視覚には実時間性が要求される。コンピュータグラフィックスの分野においては、密度の高い正確なモデルを構築することが目標となるが、ロボット用視覚の場合、実時間性とのトレードオフが重要である。本論文では次の2

点について変更を加えて高速にモデリングする手法を提案する．

1. octree を用いて階層的に大きさの異なるボクセルを表現し，密度の異なるモデルを扱えるようにする．
2. 距離計測精度を考慮し，符号付き距離の計算アルゴリズムにモデルの細かさ と処理時間のトレードオフを加える．

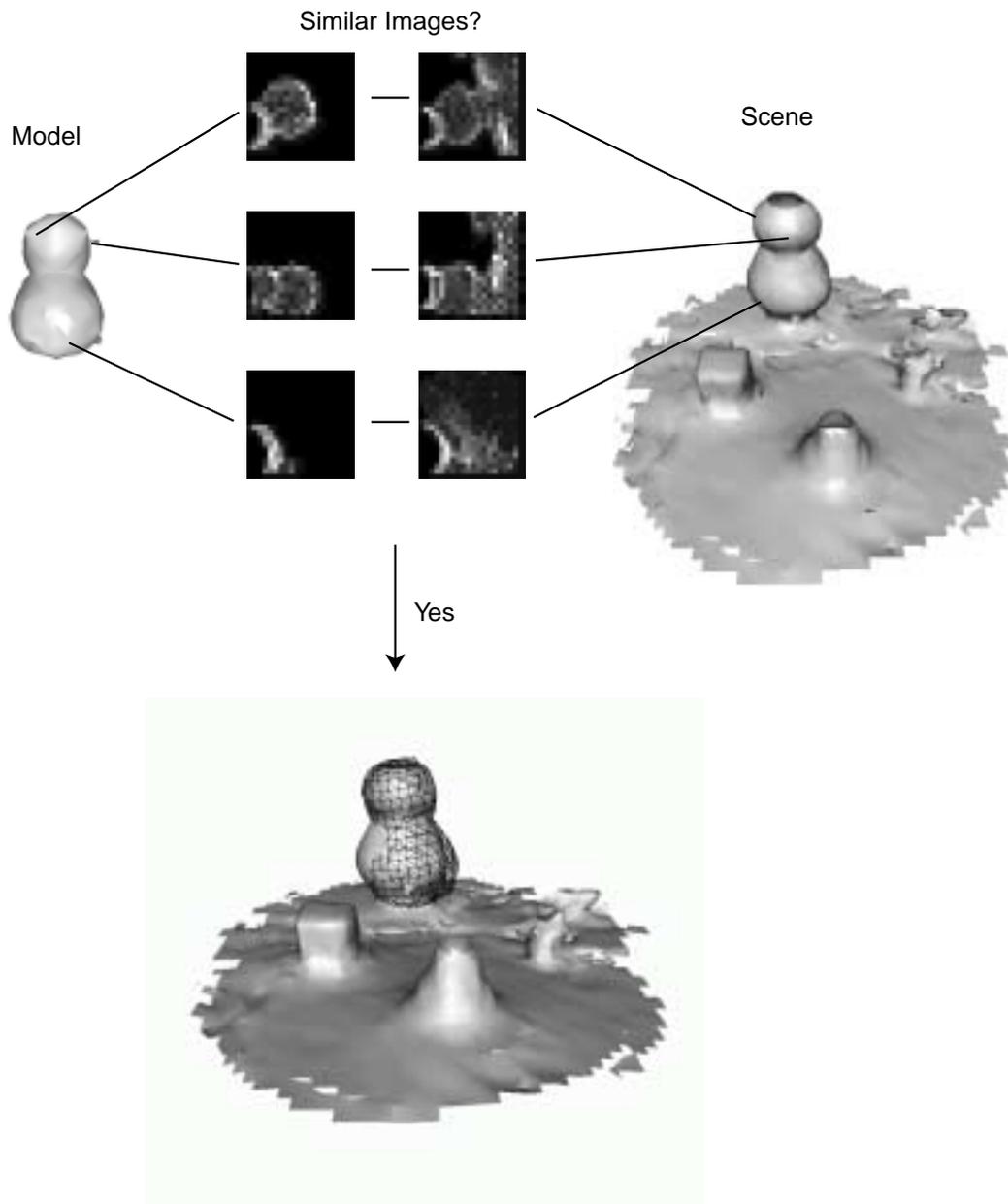
1.3 表面形状のマッチング

メッシュモデルを用いたマッチングはメッシュの頂点のマッチングからなる．2 つのモデル間で多くの頂点が似ているならば，2 つのモデル全体が似ているといえる．モデルのマッチングを点のマッチングに分割することによって問題を小さな問題の集合に分けることができる．

点のマッチングを行なった研究には次のようなものがある．Stein と Medioni[26] は形状につけたインデックスとスプラッシュ(splash) という表現を用いて点のマッチングを行なった．Chua と Jarvis[7] は表面の主曲率を用いて点をマッチングし，モデルとシーンの表面を対応づけした．点の区別するためには点の特徴を記述した表現が必要であるが，その表現に含まれる大域性と局所性を調節することが重要である．

本論文では，Johnson ら [27] によって提案されているスピニメージ (spin-image) によって点の特徴を記述する手法を用いてマッチングを行なう．スピニメージとは，メッシュ上の 1 点の法線方向を基準として他の点を 2 次元パラメータで表し，他の全ての頂点のパラメータを 2 次元配列に加算することによって得られる 2 次元画像である．スピニメージはマッチングにおいて次のような特徴を持つ．スピニメージの座標系は物体を基準にした座標系となり，観測視点位置と独立な表現になる．また，スピニメージの生成に用いるパラメータを変えることにより，スピニメージに含まれる特徴の大域性と局所性を調節することが可能である．また，相関演算を用いて 2 つのスピニメージを直接比較することによりスピニメージのマッチングを行なう．スピニメージを用いた対応点探索により複数の点について対応がとれるとモデルとシーンの間の変換パラメータを計算することができる (図.1.3) ．

本論文では上述した手法によって生成したモデルにスピニメージマッチングを適用する．階層的に密度の異なるモデルを持つことを利用し，次のような特徴を



☒ 1.3: Surface matching concept

持ったマッチングアルゴリズムを提案する．

1. 認識するモデルに合わせてマッチングに用いるメッシュの細かさを選択する．
2. coarse to fine 戦略を用いて粗いメッシュのモデルから細かいメッシュのモデルへ階層的にシーンとモデルのマッチングを行なう．

1.4 本論文の構成

本論文は5つの章からなる．本節では第2章以降の論文構成について説明する．第2章では，まず第2.1節でカメラモデルについて考察しカメラによる3次元から2次元への射影をモデリングする．また，それを用いたカメラのキャリブレーションを説明する．次に第2.2節から第2.5節ではステレオマッチングのアルゴリズムについて説明し，汎用CPUを用いた実装について述べる．第2.6節では多数のカメラを用いたステレオ視の能力の改良法について説明する．

第3章では，まず第3.1節で Marching Cubes Algorithm と符号付き距離法について説明する．次に第3.2節でアルゴリズムのインクリメンタル化と階層的なモデリングについて述べる．

第4章では，第4.1節でスピンイメージの生成とマッチング法について説明する．第4.3節では階層メッシュモデルへスピンイメージマッチングを適用し，メッシュの細かさの選択と coarse to fine 戦略によるマッチングについて述べる．

第5章では，まとめとしての考察および結論を述べる．

第 2 章

距離画像生成

本章では、まずステレオ視の結果を利用して距離計測するためのカメラモデルについて説明する。つぎに、再帰相関演算手法 [15] を用いたステレオ視のアルゴリズムと実装においてプロセッサのマルチメディア命令を用いた高速化 [16] について述べる。また、2 台以上のカメラを用いてステレオ視の結果の改良方法について説明する。

2.1 カメラモデル

まず始めに、カメラの幾何モデルについて説明する。ステレオ視によって距離計測するためには、用いるカメラのパラメータを正確に測定することが重要である。カメラパラメータは、焦点距離、レンズの歪みなどの内部パラメータと、位置姿勢の外部パラメータからなる。本論文ではまず、レンズの歪みを補正してカメラを光学的にピンホールカメラとみなせるようにしてから、Tsai の手法 [28] の基づいて、焦点距離と世界座標に対するカメラの位置姿勢を求める。

2.1.1 レンズの歪みの補正

広い視野を得るための方法の一つとして広角レンズを用いる方法がある。しかし、広角レンズを用いた場合、樽状に歪んだ画像となる。歪んだ画像のままでは、カメラによる3次元から2次元への射影を扱いにくいので、まずレンズの歪みを同定し、射影が行列で表せるピンホールカメラモデルに補正する。本論文では、[29] で提案される次の手法を用いて広角レンズの歪みを補正する。

レンズの歪みを以下のようにモデル化する。 C_x, C_y をカメラの中心とし、 x_d, y_d を補正前の画像座標、 x, y を補正後の画像座標とする。 α, k_1, k_2 が求めるレンズ歪み係数である。

$$\begin{aligned} x &= (x_d - C_x)(1 + k_1 r_d^2 + k_2 r_d^4) + C_x \\ y &= (y_d - C_y)\alpha(1 + k_1 r_d^2 + k_2 r_d^4) + C_y \\ r_d^2 &= (x_d - C_x)^2 + ((y_d - C_y)\alpha)^2 \end{aligned}$$

レンズの歪みを評価する関数は以下である。格子状の模様を持つシートをカメラで撮像し (図.2.1 (Left))、次に画像中で格子を構成する各画素について、ハフ変換を行いハフ空間上に投票を行う。補正が正しければ、格子の各直線に対応するハフ空間上の点の値は鋭いピークとなる。したがって、このピークの鋭さを評価すれば、レンズの歪みを評価することとなる。

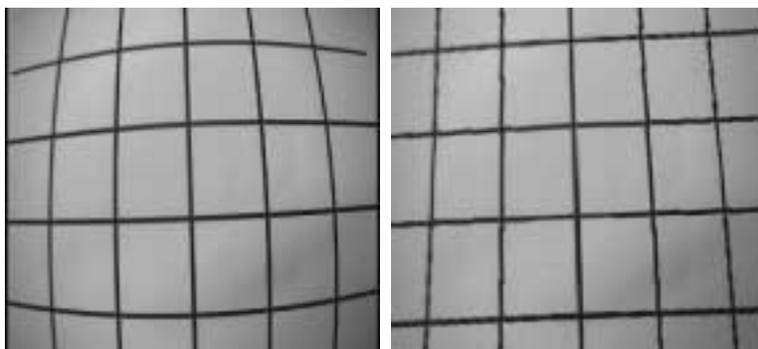


図 2.1: Left)Uncalibrated Image,Right) Calibrated Image

α, k_1, k_2 を適当な範囲で変化させ、ピークの値が最大になるものが、求める歪み係数である。補正後の画像を図.2.1 (Right) に示す。

2.1.2 ピンホールカメラパラメータの同定

カメラパラメータの同定は空間中の既知の点と、その画像中で対応する点から計算できる [28]。対応点は多ければ多いほど、より精度の良いカメラパラメータの同定が可能になるが、大量の対応点の獲得は困難である。本稿では磁場を用いて非接触で位置、姿勢が計測できるポヒマスセンサ¹ をマーカに取り付け、これを画像中でトラッキングすることで大量の対応点を獲得する手法を説明する。

ポヒマスセンサは一辺 3[cm] のレシーバとトランスミッタから構成され、レシーバの位置、姿勢を 120[Hz] で測定できる。またこの時の位置精度は 0.08[cm]、0.15 [degree] である。ただし、レシーバとトランスミッタの距離は 2[m] 以下である必要がある。

大量の対応点の獲得により精度のよいカメラパラメータの同定が可能になることを図.2.2 に示す。対応点を増やすことで、三次元情報を計算した際の、誤差の平均を減少している。カメラと対象物の距離は約 1[m] から 2[m] であった

上述の手法によって求められるパラメータは次のようになる (図.2.3)。

- 焦点距離 (F)
- 投影面上における 1 画素あたりの幅 (d_x, d_y)

¹<http://www.polhemus.com/>

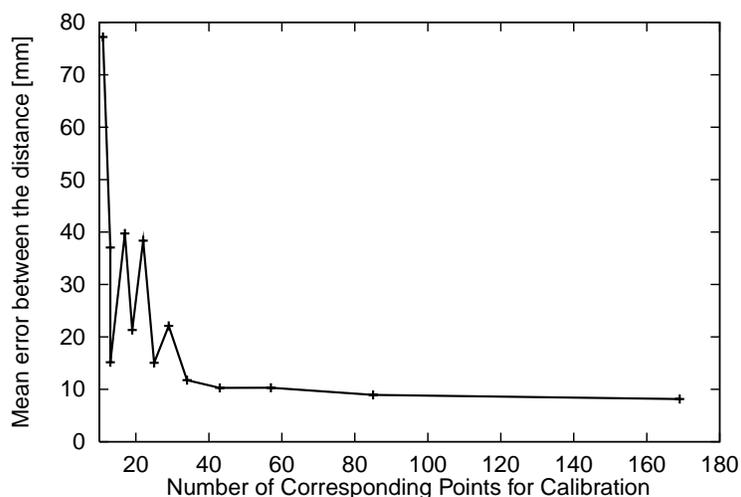


図 2.2: Accuracy of Camera Parameters against Number of Corresponding Points

- 画像の中心点 (c_x, c_y)
- 世界座標に対するカメラ原点の位置 (t_x, t_y, t_z) , 姿勢 (r_x, r_y, r_z)

これらのパラメータを用いるとカメラによる射影は次のように表される .

$$P\mathbf{p}_c = \begin{pmatrix} F & 0 & -c_x & 0 \\ 0 & F & -c_y & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} Fx - c_x z \\ Fy - c_y z \\ 1 \\ z \end{pmatrix} = \mathbf{p}'_c \quad (2.1)$$

点 $\mathbf{p}_c(x, y, z, 1)^T$ はカメラ座標系 C_C における 3 次元中の点 $\mathbf{p}_c(x, y, z)$ の同次座標 , 行列 P はカメラによる射影を表す 4×4 行列である . ここで , 得られた \mathbf{p}'_c の第 4 成分で各成分を割る .

$$\frac{1}{z}\mathbf{p}'_c = \begin{pmatrix} \frac{Fx - c_x z}{z} \\ \frac{Fy - c_y z}{z} \\ \frac{1}{z} \\ 1 \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ \frac{1}{z} \\ 1 \end{pmatrix} = \mathbf{p}_r \quad (2.2)$$

得られた \mathbf{p}_r の第 1 , 2 成分は , \mathbf{p}_c を射影したときの座標を表す . これから , 画像座標系における点 $\mathbf{p}_i = (i, j)$ は次のように求められる .

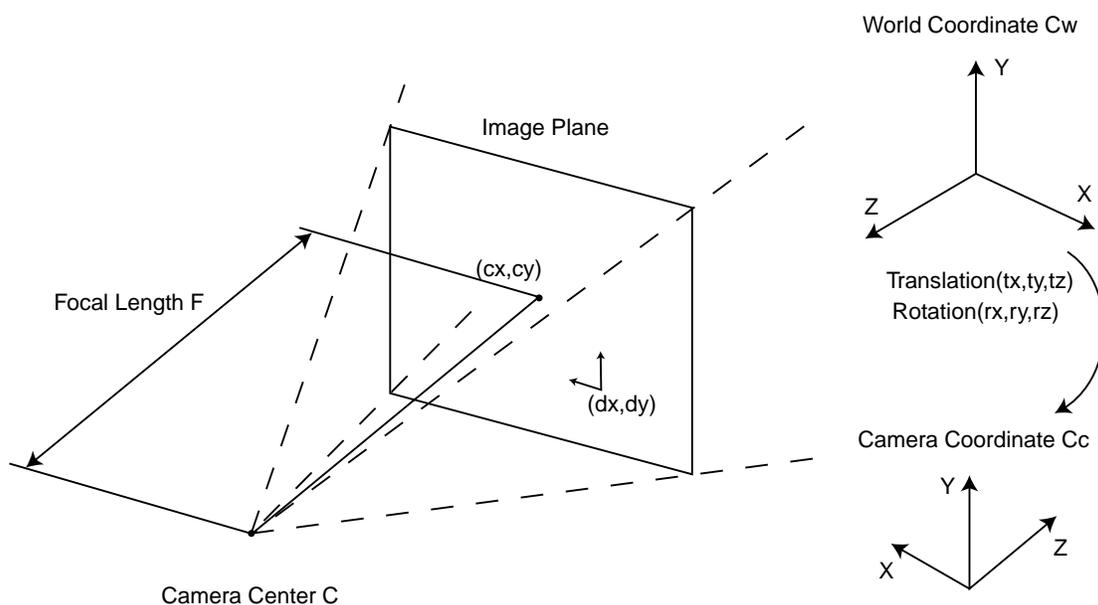


図 2.3: Pin-hole camera model parameters

$$i = \frac{x'}{d_x} + \frac{W_i}{2}, \quad j = \frac{y'}{d_y} + \frac{H_i}{2} \quad (2.3)$$

ここで, W_i, H_i は画像の幅, 高さの画素数である. 以下では, 式 (2.1), 式 (2.2) による p_c から p_r への変換を $p_r = f(p_c)$ で表し, 式 (2.3) による変換を $p_i = g(p_r)$ で表すことにする.

次に, 世界座標系に対するカメラ座標系の回転, 平行移動は次のように表される. まず, 世界座標系の X, Y, Z 軸の周りの回転角 r_x, r_y, r_z から世界座標系に対するカメラ座標系に変換する回転行列 R は次のように表される.

$$R = R_Z(r_z)R_X(r_x)R_Y(r_y) \quad (2.4)$$

ここで, R_X, R_Y, R_Z は X, Y, Z 軸の周りの回転行列である². R_X, R_Y, R_Z の積の順

²それぞれ次のようになる.

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_Y(\theta) = \begin{pmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_Z(\theta) = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

序は r_x, r_y, r_z の求められ方によって異なる．この場合はポヒマスセンサの出力に依存しているものである．また，平行移動は t_x, t_y, t_z を用いて次の行列で表される．

$$T = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.5)$$

R, T を用いて世界座標系で表された点 p_w からカメラ座標系で表された点 p_c の変換行列 M は次のように表される．

$$p_c = Mp_w = RTp_w \quad (2.6)$$

以上の式 (2.1) から式 (2.6) により，世界座標系で表された点 p_w から画像座標系で表された点 p_i までの変換が表された．

2.1.3 カメラ対の平行等位化

ステレオ視の対応点探索において，エピポーラ線が画像の画素の並びに沿っていると効率的に探索を行なうことができる．エピポーラ線がカメラの画素の並びに沿っていない場合にカメラ対を平行等位化 (rectification) する方法 [30] を次に説明する．

2つのカメラ C_1, C_2 に対して，それぞれのカメラ座標原点を O_1, O_2 とし，回転行列を R_1, R_2 とする．

$$\mathbf{x} = \frac{\mathbf{b}}{|\mathbf{b}|}, \quad \mathbf{b} = \mathbf{O}_2 - \mathbf{O}_1 \quad (2.7)$$

$$\hat{\mathbf{z}} = \frac{\mathbf{z}_1 + \mathbf{z}_2}{|\mathbf{z}_1 + \mathbf{z}_2|}, \quad \mathbf{z}_1 = R_1^{-1}\mathbf{z}_0, \quad \mathbf{z}_2 = R_2^{-1}\mathbf{z}_0 \quad (2.8)$$

$$\mathbf{z} = \frac{\hat{\mathbf{z}} - a\mathbf{x}}{|\hat{\mathbf{z}} - a\mathbf{x}|}, \quad a = \mathbf{x} \cdot \hat{\mathbf{z}} \quad (2.9)$$

$$\mathbf{y} = \frac{\mathbf{z} \times \mathbf{x}}{|\mathbf{z} \times \mathbf{x}|} \quad (2.10)$$

ここで， \mathbf{b} はカメラ間のベースラインである．また， $\mathbf{z}_0 = (0, 0, 1, 1)^T$ であり，これから求められる $\mathbf{z}_1, \mathbf{z}_2$ は2つのカメラ座標系における Z 軸に平行なベクトルであ

る．ここで求められた x, y, z を用いて平行等位化されたカメラ C_R の回転行列 R_R は次のように表される．

$$R_R = \begin{pmatrix} x_1 & x_2 & x_3 & 0 \\ y_1 & y_2 & y_3 & 0 \\ z_1 & z_2 & z_3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.11)$$

ここで, $\mathbf{x} = (x_1, x_2, x_3, 1)^T, \mathbf{y} = (y_1, y_2, y_3, 1)^T, \mathbf{z} = (z_1, z_2, z_3, 1)^T$ である．

R_R を用いてカメラ C_1 の画像上の点 \mathbf{p}_{i_1} から補正されたカメラ C_{R1} 上の点 $\mathbf{p}_{i_{R1}}$ への変換と, カメラ C_2 から C_{R2} への変換は次のように表される．

$$\mathbf{p}_{i_{R1}} = g((P_1 R_R R_1^{-1} P_1^{-1}) g^{-1}(\mathbf{p}_{i_1})) \quad (2.12)$$

$$\mathbf{p}_{i_{R2}} = g((P_1 R_R R_2^{-1} P_2^{-1}) g^{-1}(\mathbf{p}_{i_2})) \quad (2.13)$$

ここで, $g(\mathbf{p}_r) = \mathbf{p}_i$ は式 (2.3) による変換である．このようにして平行等位化された2つのカメラ C_{R1}, C_{R2} の対応点は画像上で同じ j の値を持つようになる．

2.1.4 ステレオカメラモデル

前節までに得られたパラメータを用いてステレオ視における距離計測の幾何モデルを図.2.4 に示す [14] . 2 台のカメラ C_1, C_2 から点 p を観測したときの視差 d は次のように表される．

$$d = \frac{B \cdot F}{Z} \quad (2.14)$$

ここで, B はカメラ間の基線長, F はカメラの焦点距離である．

次に画像の量子化が与える距離計測の分解能への影響について考察する．視差の分解能を Δd とすると距離の分解能 ΔZ は次のように表される．

$$\Delta Z = \frac{BF}{d} - \frac{BF}{d + \Delta d} = BF \frac{\Delta d}{d^2 + d \Delta d} \quad (2.15)$$

Δd は画像上での対応点探索における分解能であるため, 画像のどの部分においても不変な定数とみなせる．したがって, 距離分解能 ΔZ は視差 d のみに依存し, d が小さくなるほど, すなわち視点から遠くなるほど大きくなる．

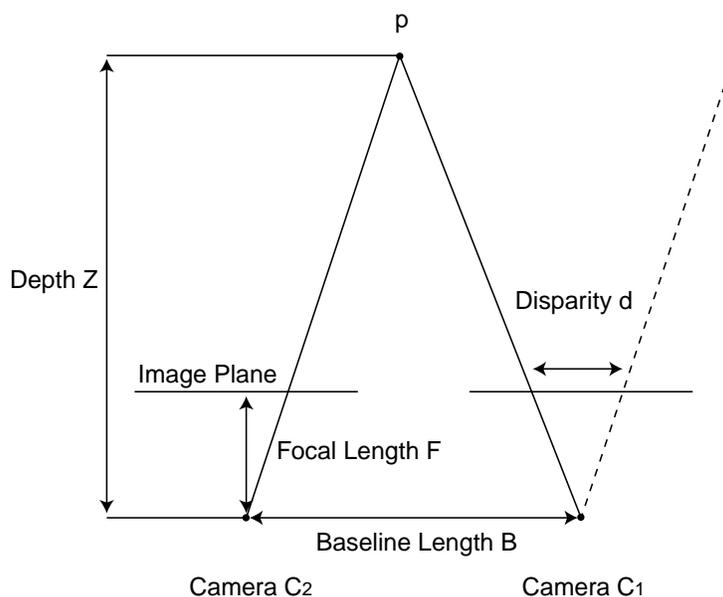


図 2.4: Stereo camera pair

2.2 ステレオ視による視差画像生成

ステレオ視は多くの場合は既知の位置関係にある二つ以上のカメラから同時に得られた画像中の各点の対応点を探索する問題である．これまで主にエッジ等の特徴をマッチングする手法と，局所領域をマッチングする手法の二種類が研究されている．局所領域のマッチングによる方法では相関演算ハードウェアによる高速化といったアプローチも存在する．いずれの手法も左右の画像で対応する領域は基本的に視差分しか離れていない，という仮定の元に探索範囲を拘束できる．またステレオ視では原理的にオクリュージョン（隠れ領域）³ 問題が発生し，この領域を発見可能なことが実用上からは重要である．

一般的に局所領域のマッチングにもとづく視差画像の生成手法は以下になる．

1. 画像のノイズ除去，正規化等の前処理を行う
2. 各小領域に対して可能な視差の範囲で対応する各小領域ごとに相関演算を行う
3. それらの中から適切な視差を選択する
4. サブピクセルレベルで視差を推測する

本論文では (2) を再帰相関演算手法 [14, 15] を用いて，(3) を LR-check 法 [31, 32]

³片目からは見えているのに反対側の目からは隠れて見えない領域

を用いて行う。実時間で視差画像の生成が可能となるように (3) では相関値分布の再配列により, (2) では以下の3段階の高速化を行った。

1. 再帰相関演算を用いたアルゴリズムの効率化
2. キャッシュを意識したアルゴリズムの展開
3. MMX 命令を用いた実装による高速化

1,2 は C 言語で, 3 は MMX のアセンブラによって記述する。

2.3 再帰相関演算を用いた対応点探索

2.3.1 相関演算

ステレオ視による距離計測のためには, 片方の画像の各小領域に対して, もう一方の画像の対応する小領域を求める必要がある。ここではエピポーラ線は完全に水平であり, 二つの画像に対して垂直方向の視差は発生しないとする⁴。各小領域の対応を計算するためには局所領域の相関値を求める相関演算を用う。

相関演算の評価式は, 以下の C_0 式で表す。

$$C_0(x, y, d) = \frac{\sum_{i,j} \{I_1(x+i, y+j) - \overline{I_1(x, y)}\} \times \{I_2(x+i+d, y+j) - \overline{I_2(x, y)}\}}{\sqrt{\sum_{i,j} \{I_1(x+i, y+j) - \overline{I_1(x, y)}\}^2} \times \sqrt{\sum_{i,j} \{I_2(x+i+d, y+j) - \overline{I_2(x, y)}\}^2}} \quad (2.16)$$

$$\overline{I(x, y)} = \frac{1}{W^2} \times \sqrt{\sum_{i,j} I(x+i, y+j)^2}$$

ただし, 左右画素 (x, y) での輝度値を $I_1(x, y), I_2(x, y)$, その値域を $0 \leq x, y < N$, 相関演算のウィンドウサイズを $W \times W$ とし, その値域を $0 \leq i, j < W$, 視差 d の値域を $0 \leq d < D$ とする。

ここで, 画像の各小領域の輝度値の平均が等しいと仮定すると, 以下の C_1 式を, 各小領域の輝度値の平均と分散が等しいとすると以下の C_2 式 (SSD: Sum of Squared Difference), C_3 式 (SAD: Sum of Absolute Difference) を用いることができる。

$$C_1(x, y, d) = \frac{\sum_{i,j} I_1(x+i, y+j) \times I_2(x+i+d, y+j)}{\sqrt{\sum_{i,j} I_1(x+i, y+j)^2} \times \sqrt{\sum_{i,j} I_2(x+i+d, y+j)^2}} \quad (2.17)$$

$$C_2(x, y, d) = \sum_{i,j} \{I_1(x+i, y+j) - I_2(x+i+d, y+j)\}^2$$

⁴すなわち対応する小領域の y 座標は同一であるとする

$$N(x,y,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$N(x+1,y,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$Q(x+W,y,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$Q(x,y,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$N(x+1,y,d) = N(x,y,d) + Q(x+W,y,d) - Q(x,y,d)$$

$$Q(x,y,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$Q(x,y+1,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$P(x,y+W,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$P(x,y,d) = \begin{array}{c} x \\ \text{WxW} \end{array} \times \begin{array}{c} d \\ \text{WxW} \end{array}$$

$$Q(x,y+1,d) = Q(x,y,d) + P(x,y+W,d) - P(x,y,d)$$

☒ 2.5: Recursive correlation calculation

$$C_3(x, y, d) = \sum_{i,j} |I_1(x+i, y+j) - I_2(x+i+d, y+j)| \quad (2.18)$$

以下, 本論文では C_3 式の SAD に基づき高速化の手法を述べる. ただし, この手法は, C_1, C_2 にも適用可能である.

2.3.2 再帰相関演算アルゴリズム

再帰相関演算と呼ぶアルゴリズム [15, 14] を導入することで, ステレオ視における相関演算の計算量を $O(N^2W^2D)$ から, $O(N^2D)$ とすることが可能であることが知られている. 以下にアルゴリズムを説明する.

$$P(x, y, d) = |I_1(x, y) - I_2(x+d, y)| \quad (2.19)$$

とすると, 式 (2.18) は

$$N(x, y, d) = \sum_{i=0}^{W-1} \sum_{j=0}^{W-1} P(x+i, y+j, d) \quad (2.20)$$

と書ける. ここで,

$$Q(x, y, d) = \sum_{j=0}^{W-1} P(x, y+j, d) \quad (2.21)$$

とすると,

$$N(x+1, y, d) = N(x, y, d) + Q(x+W, y, d) - Q(x, y, d) \quad (2.22)$$

と再帰的に計算できる (図.2.5). さらに $Q(x, y, d)$ 自身も再帰的に計算できる.

2.3.3 再帰相関演算アルゴリズムによる対応点探索

式 (2.18) による視差の計算は, 以下のように書くことができる.

$$O(x, y) = \min_d \{N(x, y, d)\} \quad (2.23)$$

$$N(x, y, d) = \sum_{i,j} |I_1(x+i, y+j) - I_2(x+d+i, y+j)|$$

ここでは $W \times W$ 回の乗算が冗長である. すなわち, $N(x, y, d)$ は再帰相関演算アルゴリズムを用いることで, 以下のように再帰的に計算できる.

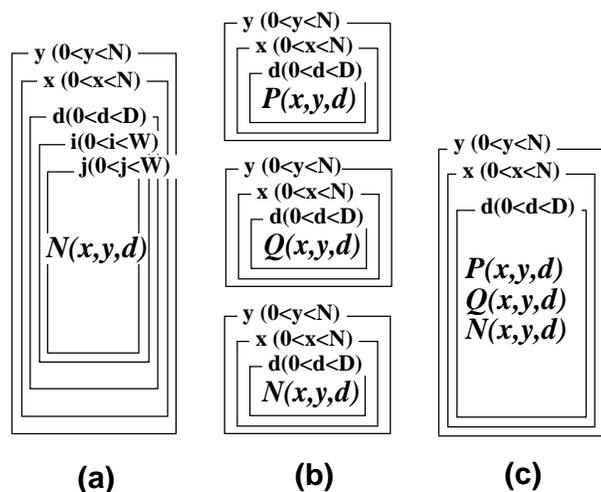


図 2.6: Optimizing implementation with (a) normal correlation, (b) simple recursive correlation, (c) cache optimized recursive correlation. (Algorithms (b,c) don't work when $x=0$ or $y=0$)

$$\begin{aligned}
 P(x, y, d) &= |I_1(x, y) - I_2(x + d, y)| \\
 Q(x, 0, d) &= \sum_j P(x, j, d) \\
 Q(x, y + 1, d) &= Q(x, y, d) + P(x, y + W, d) - P(x, y, d) \\
 N(0, y, d) &= \sum_i Q(i, y, d) \\
 N(x + 1, y, d) &= N(x, y, d) + Q(x + W, y, d) - Q(x, y, d)
 \end{aligned} \tag{2.24}$$

2.3.4 キャッシュを意識したアルゴリズムの展開

図.2.6 は (a) 式 (2.23) を単純に実現したもの, および (b) 再帰相関演算を単純に実現した際のループの構成法を示している. しかし全画面の対応点を計算するためには, 再帰相関演算の式を x, y, d の 3 重のループとして計算することになるが, 単純に式の通りに実装するとキャッシュを利用できず, 計算速度は CPU の演算速度ではなくメインメモリの読み出し / 書き込み速度に依存することになってしまう⁵.

そこでキャッシュの利用効率を最大にするために演算途中に使用する P, Q を計

⁵メインメモリからのデータの読み出し, 書き込みは 10 クロック以上かかるが, 2 次キャッシュからは 1 または 2 クロックで行える

算する際に N も同時に計算することとする (図.2.6(c)) . これにより単純にインプリメントすると P, Q, N は $N \times N \times D$ の配列サイズをそれぞれ持つことになっていたが, この場合には P は $N \times W \times D$, Q は $N \times 2 \times D$, N は $N \times D$ の配列サイズで良いことになる .

従って, 本論文で述べたアルゴリズムに必要なメモリ領域は, $N=128, W=16, D=32$, 画像の各ピクセルのデータ量を 8bit とすると, P は 8bit, Q, N は 16bit となることから, 約 2.6MB となるが, この方法により約 85KB まで低減することができ, PentiumMMX, PentiumII 等の 2 次キャッシュに入りきる⁶ .

2.3.5 MMX 命令を用いた実装による高速化

ここまでは C 言語によるステレオ対応点探索問題の高速化について述べてきた . 本節では最も計算時間を要している再帰相関演算の高速化を図るために, Pentium の MMX 命令を利用する .

MMX 命令

MMX 命令は Intel の PentiumMMX プロセッサ以降に搭載されたマルチメディア命令で, 一クロックで 64bit 長の MMX レジスタ上 (あるいはメモリ上) におかれたデータ配列の和, 積, 積和, 論理演算等を行う, いわゆる SIMD (Single Instruction Multi Data) な命令セットで, 積以外は一クロックで実行される . ただし割り算命令をもっていないことと, 積は $16\text{bit} \times 16\text{bit}$ までしかできないといった制約がある .

MMX 命令による実装

式 (2.24) の $P(x, y, d)$ は MMX 命令を用いて図.2.7 のように実現される⁷ . ここでは 8bit 整数の画像配列をレジスタに移した後, 16bit 整数に拡張し, 4 つずつまとめてお互いに差をとり, 2 つの結果を or することにより条件分岐を行わずに差の絶対値を計算し, P に代入している . このコードは C 言語で書いたものに比べて約 2 倍早く実行される . 同様に Q, N も計算する .

⁶これ以外に画像領域として $128 \times 128 \times 2 \times 8\text{bit} = 32\text{KB}$ のメモリが必要である

⁷gcc(gas) の記述のためにオペランドのソースとデスティネーションは通常の x86 の方式と逆になっていることに注意されたい . %%mm0,1,2 は 8 個存在する MMX レジスタを指す

```

unsigned char  I1[N][N], I2[N][N];
unsigned short P[N][W][D];
asm volatile("pxor %%mm4,%%mm4");
for (d=0; d<D; d+=4) {
  asm volatile("movd %0,%%mm0"::"m"(I1[x][y]));
  asm volatile("movd %0,%%mm1"::"m"(I2[x+d][y]));
  asm volatile("punpcklbw %%mm4,%%mm0");
  asm volatile("punpcklbw %%mm4,%%mm1");
  asm volatile("movq %%mm1,%%mm2");
  asm volatile("psubusb %%mm0,%%mm1");
  asm volatile("psubusb %%mm2,%%mm0");
  asm volatile("por %%mm0,%%mm1");
  asm volatile("movq %%mm1,%0"::"m"
                (P[x][y%W][d]));
} ...

```

図 2.7: MMX implementation of Equation (2.24)

さらに N の配列から最小値 0 を求めるコードも MMX 比較命令を用いることで高速に計算できる。

2.4 一貫性評価法による信頼度評価

2.4.1 一貫性評価法 (Consistency Checking)

視差画像生成システムを実世界で行動するロボットへの適用する場合、特徴量が少なかったりオクルージョンが存在する環境でも信頼のおける視差画像を生成する必要がある。そのためには、相関演算の結果から適切な視差を検出する機構を視差画像生成システムに組み込む必要がある。

この問題を解決するために様々な研究が行われているが、本論文では対応点探索における相関演算の信頼度評価に”Left-to-Right Right-to-Left Consistency Checking(LR-check)”を用いる [31, 32]。

これは片側の画像の点に対応する反対側の画像上の点からの対応点が、最初の点に一致する場合のみ、その対応が信頼できるとするものである (図.2.8)。この方法により、対応点がとれない場所がはっきり区別出来るようになる。LR-check アルゴリズムは以下の通りである。

1. 右画像中の局所領域を参照領域 (領域 1) とし、左画像を探索し最も相関の高い局所領域を領域 1 に対応する領域 (領域 2) とする。

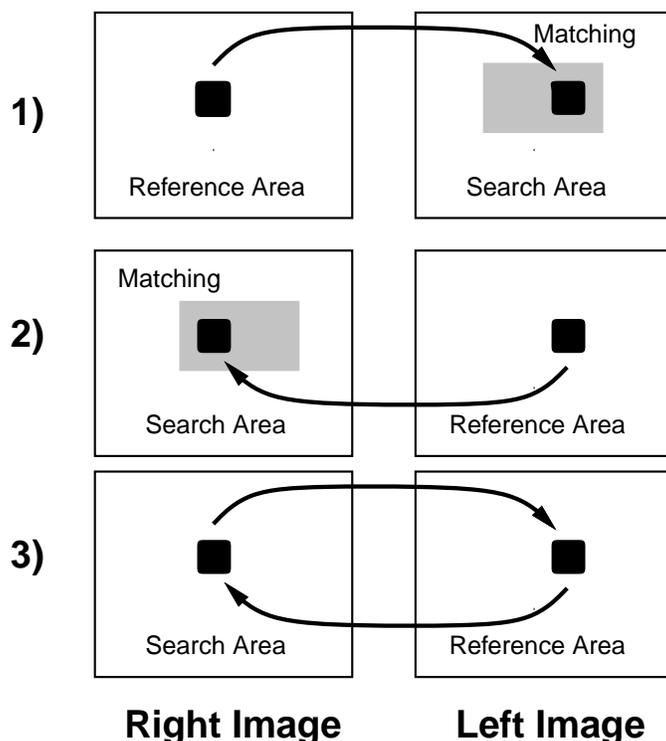


図 2.8: Consistency checking between Left-Right images

2. 領域 2 を参照領域とし，右画像を探索し最も相関の高い局所領域を領域 2 に対応する領域 (領域 3) とする。
3. 領域 1 と領域 3 が同じ局所領域であれば，領域 1 と領域 2 は左右画像中の対応する領域とする。

2.4.2 LR-check の再帰相関演算への組み込み

LR-check 法はキャッシュを意識した高速アルゴリズムのループ中に組み込むことにより，メモリ使用量，計算時間を増やすことなく視差画像を高速に計算できるようになる。

まず左右の画像の信頼度の計算を行う際には，対応点探索のための相関度の計算を右から左，左から右の双方で行う必要はなく，相関演算は一度行い，得られた相関値分布の再配列することで，両側の対応点探索を行えばよい。

従って LR-check は 図.2.6(d) の y のループ内の最後に x 方向の一ライン分まとめて行うことにより，計算時間は 30%程度増加する程度で行える。

2.5 ロボット搭載用視差画像生成システム

2.5.1 ロボット搭載用システムの開発

近年ロボットに PC と視覚処理ボードを搭載し画像処理を行いながら行動する
知能ロボットの研究が盛んに行われている (ex. [33]) .

また、視差画像生成システムを実際にロボットに搭載する際には、ステレオカメラからの画像の取り込みが重要な問題となってくる。ここではロボットに搭載することを目的として研究室で開発された、同期して動作している 2 つのカメラの映像信号を電子的にマージし、インターレースされたステレオ画像を取り込み装置に与える方法を採用した [34] .

画像取り込み装置は Bt848 という画像取り込み IC を用いたキャプチャーカードを用いた。このハードウェアはリアルタイムで画像をメモリに DMA 転送する機能を持ち、Linux 用のデバイスドライバが公開されていることから、容易に実時間画像処理システムを構築することができる、という特徴を持っている。また、カメラは VCC-540 を用いた。

2.5.2 視差画像生成システムの性能評価

本手法の有効性を示すために実験を行った。全ての実験を比較のために PentiumMMX-233MHz(外部バス 66MHz) と PentiumII-500MHz(外部バス 100MHz) を用いて linux 2.2.9 上で行った。C 言語は全て gcc-2.7.2.3 を用いて、また MMX は gcc 上でインラインアセンブラを用いて実現した。実験は全て同一のバイナリを用いて行った。

実時間性の評価

本手法の実時間性を評価した。これまでに述べてきたように、a) SAD による対応点探索による視差計算 (式 (2.23)) を単純に実装した場合、b) 再帰相関演算 (式 (2.24)) を単純に実装した場合、c) キャッシュを意識した再帰相関演算を実装した場合、d) MMX を用いた実装、のそれぞれのバージョンのソフトウェアについて、信頼度評価を行わない場合、一貫性評価法による信頼度評価を行った場合、信頼度評価を行いサブピクセル法により視差を推定した場合の計算時間を計測したものを表.2.1 に示す。

表 2.1: Realtime Performance of Disparity Image Generation

Function	PentiumII 500MHz		
	1)	2)	3)
a)	1837.2	4081.7	3675.8
b)	37.1	55.1	58.3
c)	33.4	42.9	44.9
d)	22.4	29.7	32.8

(msec) ($N=128, W=13, D=32$)

Row : a) Simple Correlation, b) Recursive Correlation, c) Cache Optimal Correlation, d) MMX Implementation.

Column : 1) Without Consistency Checking, 2) With Consistency Checking, 3) With Consistency Checking and Sub Pixel Interpolation.

信頼度評価を行いサブピクセル法により視差を推定することで信頼度の高い視差画像を生成する計算が、再帰相関演算を MMX により実装することで実時間 (33msec 以内) で行えることが確認できる。

距離分解能の評価

距離分解能を評価するために、空間上の既知の約 500 点と、その画像上の位置を計測し、Tsai の手法 [28] の基づきカメラを補正した。その結果、焦点距離 (focal length) は 5.898 [mm] であることが分かった。カメラ (VCC-540) の CCD のサイズより、使用したカメラの画角は 53.81 [degree] であることが分かった。

カメラから 50[cm] から、200[cm] の距離まで、物体を連続的に移動させ、その間の距離を視差画像に基づき測定した。視差画像に基づき測定した距離と、真の距離を比較することで評価を行う。

結果を図.2.9 に示す。軸はカメラから物体までの距離 (単位: [cm]) を示している。実線が視差画像に基づき測定した値であり、破線は視差画像の生成の際にサブピクセルによる視差の推定を行わなかった場合である。また、表.2.2 に真の値と測定した値との誤差の平均と標準偏差を、サブピクセル処理を行う場合、行わない場合のそれぞれについて示した。

これらの結果から、カメラから物体までの距離が 150[cm] 以内では、距離計測の

表 2.2: Evaluation of Distance Estimation : Average error and standard deviation.

Distance between the camera and the object	Estimated from Disparity Image	
	Average error	Standard dev.
50 - 100	1.724	1.929
100 - 150	2.949	4.688
150 - 200	10.001	47.216
	Estimated without Sub Pixel	
50 - 100	2.841	4.458
100 - 150	4.932	11.657
150 - 200	15.952	83.674

誤差は数 [cm] であり, 150[cm] 以上の場合は数十 [cm] の誤差が生じる. 数 [cm] の誤差は物体認識を行うために十分であり, 数十 [cm] の誤差は障害物回避などの目的に適用できると考えている.

四脚歩行ロボットの距離情報に基づく行動実験

視差画像生成システムを四脚歩行ロボット JROB1[35] に搭載し距離情報に基いた行動実験を行った. ロボットは距離情報に基づき障害物を発見, 回避し移動する.

実験は屋外環境で行った. 実験風景を図.2.10 に示す. ロボットの前方に人が立っている状況で, カメラで得られた画像, 視差画像 (サブピクセル無し), 視差画像 (サブピクセル有り) を図.2.11 に示す. 白い部分が近い領域であり, 障害物となる人間が認識できる.

また, 視差画像から三次元情報を再構成し, 入力画像をテクスチャマップした画像を図.2.12 に示す. サブピクセル処理を行わないと, 例えば地面の形状など, 三次元情報の再構成が不十分であることが確認できる.

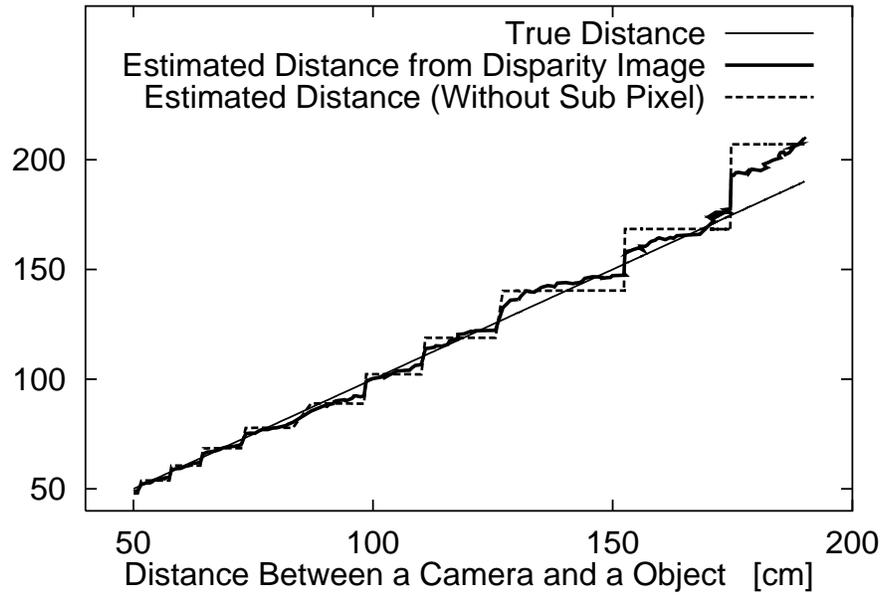


図 2.9: Evaluation of Distance Estimation : True distance and estimated distance from disparity image with subpixel and without subpixel

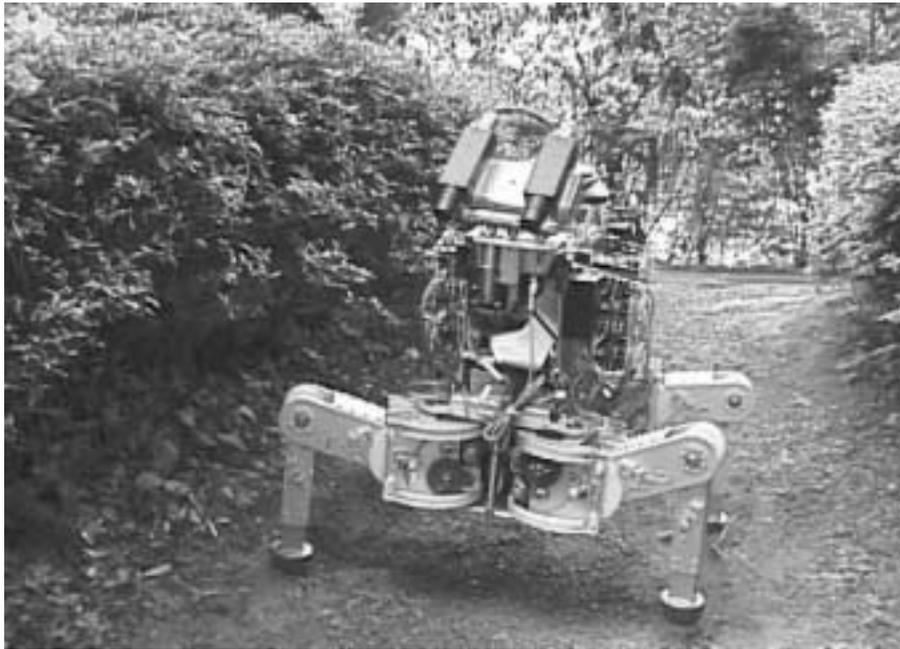


図 2.10: Quadruped Legged Robot JROB1 in Outdoor Environment



図 2.11: Left Top: Input image, Right Top: Disparity Image (brighter is closer), Left Bottom: Disparity Image with Consistency Checking, Right Bottom: Disparity Image with Consistency Checking and Sub Pixel Interpolation

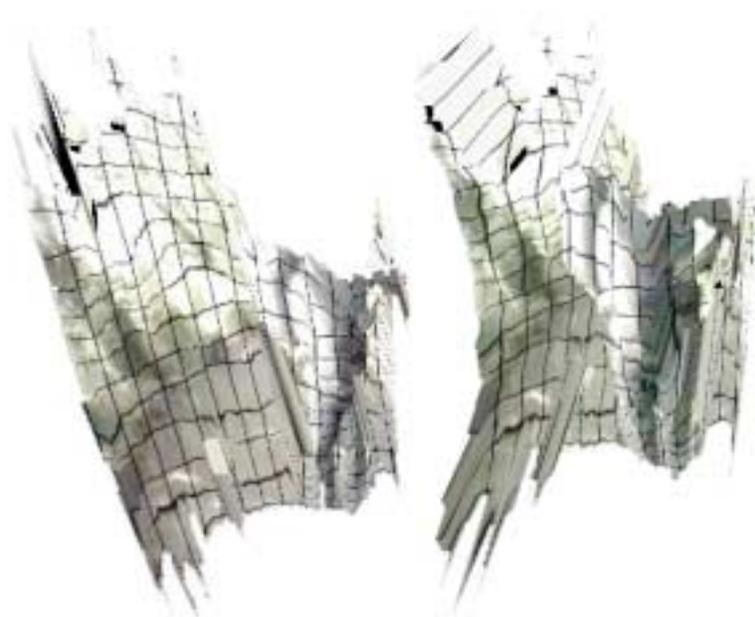


図 2.12: 3D Reconstruction from Disparity Image with Texture Mapping

2.6 多数の画像を用いたステレオ視

距離計測結果を改善する手法として、2枚以上の画像を用いてステレオ視を行なう方法が考えられる。2枚上の画像を用いたウィンドウ領域の相関演算によるステレオ視の研究ではマルチベースラインステレオ法 [14, 36] が挙げられる。マルチベースラインステレオ法を以下に説明する。

前節までにおいて、画像上の点 (x, y) において各視差 $d (= 0, \dots, D)$ に対して評価関数 $C(x, y, d)$ を計算し、その中で最も良い値を与える視差 d を選ぶことにより対応点探索を行っていた。本節では、式 (2.14) から得られる次式を用いて、視差 d に対して評価関数を計算する代わりに、距離 Z に対して評価関数を計算する。

$$Z = \frac{B \cdot F}{d}$$

基線長に依存しない距離 Z を評価関数のパラメータとして用いることにより、基線長によらず正しい距離における評価関数 $C(x, y, Z)$ は良い評価値を与えるはずである。複数の基線長に対応した複数の評価関数を足し合わせることで計測の精度と信頼性の向上が期待できる。すなわち、評価関数 $C_i(x, y, Z) (i = 1, \dots, N)$ を足し合わせた新しい評価関数 $SC(x, y, Z)$ は次のようになる。

$$SC(x, y, Z) = \sum_i C_i(x, y, Z) \quad (2.25)$$

評価関数 $C(x, y, Z)$ は、画像のテクスチャの問題や、観測対象のオクリュージョンの問題によって必ずしも正しい距離 Z において最良の値を持つとは限らないが、異なる基線長のステレオ対から計算された複数の評価関数を足し合わせることで曖昧さを除去し、計測精度を向上させることができる。(図.2.13)

1台のカメラを動かし、位置の変化とともに得られる時系列画像を用いてマルチベースラインステレオを適用した結果を以下に示す [37]。図.2.14のように、平面の背景の前に円筒を置き、カメラを動かしながら画像をフレームレートで取り込む。本稿では、オフライン処理で距離計測を行なっている。また、誤対応領域を少なくするために観測対象にテクスチャを貼っている。使用した画像数と、ウィンドウサイズを変えて生成した距離画像の違いを図.2.15に示す。距離画像1枚につき、ベースライン距離最大約3cm、距離分解能5mmで距離画像生成を行なった。ウィンドウサイズが 3×3 pixel の場合には、使用する画像数を増やすことにより、誤対応領域を減らすことができている。ウィンドウサイズ 11×11 pixel の場合と比較すると、エッジ部分の距離計測が正確に計測できていることがわかる。

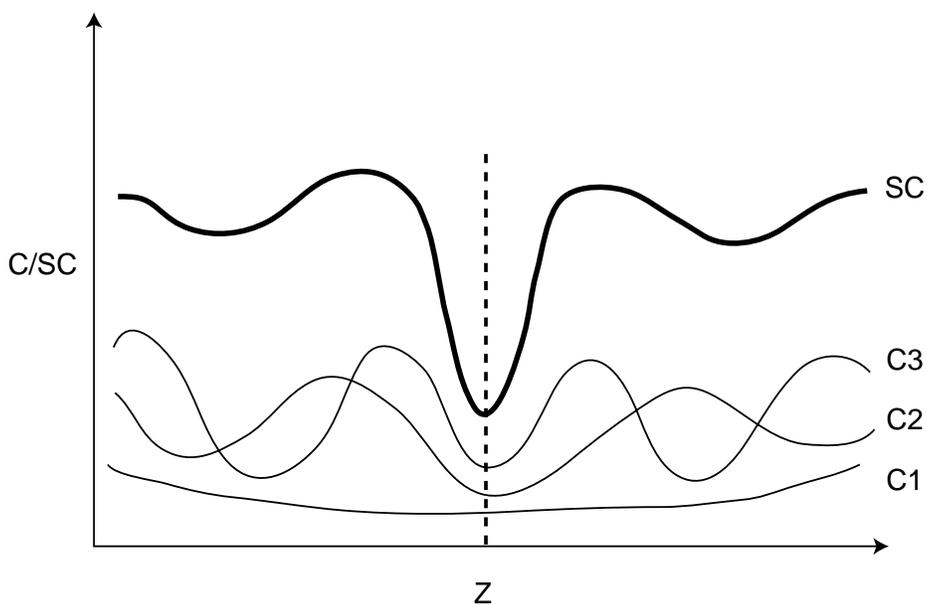


図 2.13: C and SC functions

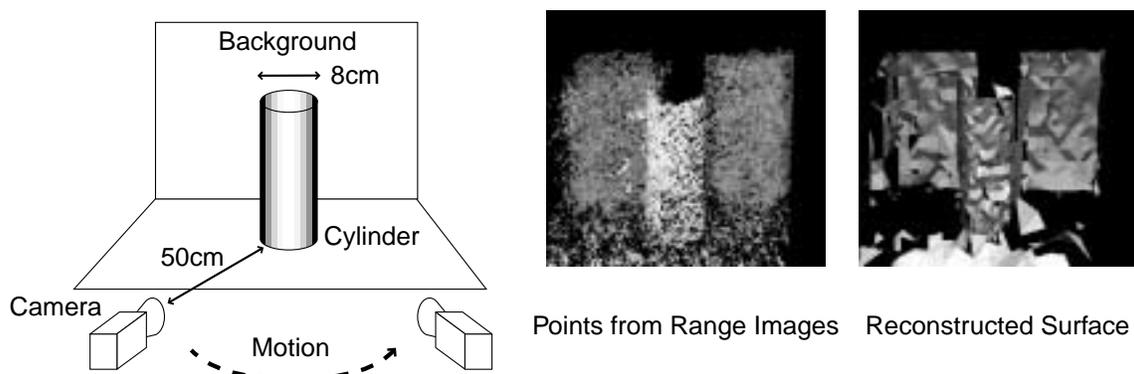
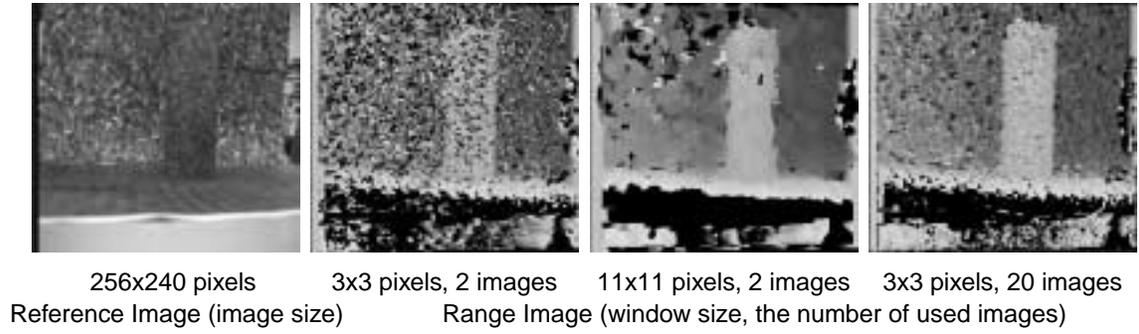
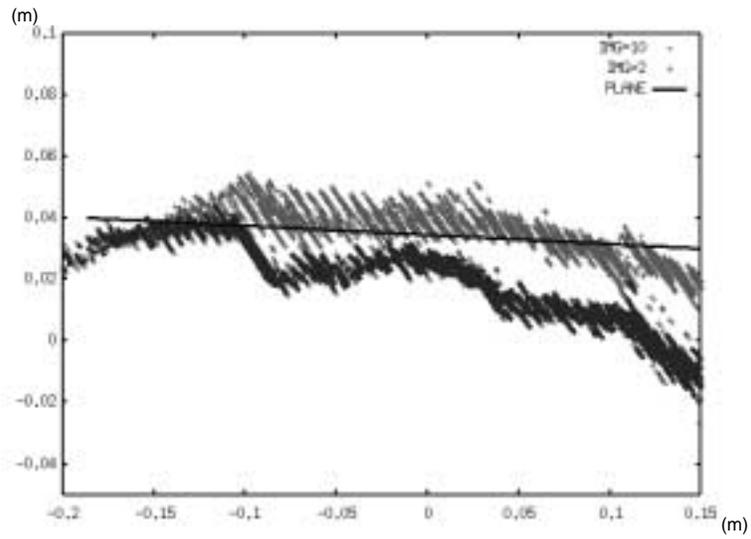


図 2.14: Reconstruction of the object surface from multiple range images

また、図.2.16 はカメラに対して約 45 度傾いた平面について、使用する画像の数を変えて同様に距離計測を行なった結果である。直線で表される平面に対して、2 枚の画像から距離計測を行なった場合には、画像の量子化の影響から結果が階段状になっている。一方、10 枚の画像を用いて距離計測を行なった場合には、複数のベースライン距離を組み合わせるために量子化の影響が抑えられている。



☒ 2.15: Compare range images



☒ 2.16: Measure distance to skew plane

第 3 章

3次元表面形状モデリング

本章では第2章で述べた距離計測法を利用して観測対象の形状をメッシュを用いてモデリングする。ステレオ視などの距離計測法では多くの場合オクリュージョン領域が存在し、観測対象の一部分しか観測できない。したがって、観測対象全体の形状をモデリングするためには複数の視点から距離計測を行ない、それらの結果を統合して1つのモデルを構築する手法が必要となる。

近年、コンピュータビジョン、コンピュータグラフィックスの分野では実世界の物体の形状をモデリングする研究が盛んに行なわれている。それらの研究ではレーザーレンジファインダ、光投影法などの距離計測装置を用いて複数の視点から距離を計測し、メッシュモデルを再構成する手法が提案されている。本論文ではそれらの研究の中で、Marching Cubes Algorithm [21] と符号付き距離法 [25, 23, 24] を組み合わせた手法を応用し、ロボット用視覚として用いるためにアルゴリズムに変更を加え、インクリメンタル性を持ったアルゴリズムを提案する。

3.1 Marching Cubes Algorithm と符号付き距離法

本節で説明する Marching Cubes Algorithm と符号付き距離法を用いた距離画像からメッシュによる表面形状モデリングの処理の流れは図.3.1 のようになる。以下では、Marching Cubes Algorithm によるメッシュ生成と距離画像を用いてメッシュ生成に必要なデータの獲得法を説明する。

3.1.1 Marching Cubes Algorithm

距離画像から得られるメッシュによる表面形状モデルは、視点からの距離によってメッシュの密度が異なるモデルとなる。複数の視点からのモデルを重ねあわせるために、視点位置から依存しないメッシュモデルで表現する。まず3次元を格子状に区切り、メッシュの頂点を格子の辺上に限定する(図.3.2)。これにより、メッシュの密度は視点位置に依存しない体積表現 (volumetric representation) となる。Lorenson と Cline が提案した Marching Cubes Algorithm [21] は、このような体積表現のメッシュモデルを生成する手法である。以下に Marching Cubes Algorithm について説明する。

まず、この手法の入力は隣接する格子点をつないで得られる立方体(ボクセル)の集合である。そして、各ボクセルの頂点にはスカラー値が与えられている。ボクセル V の8つの頂点 $v_j (j = 1, \dots, 8)$ に与えられているスカラー値を $Z(v_j)$ で書くこ

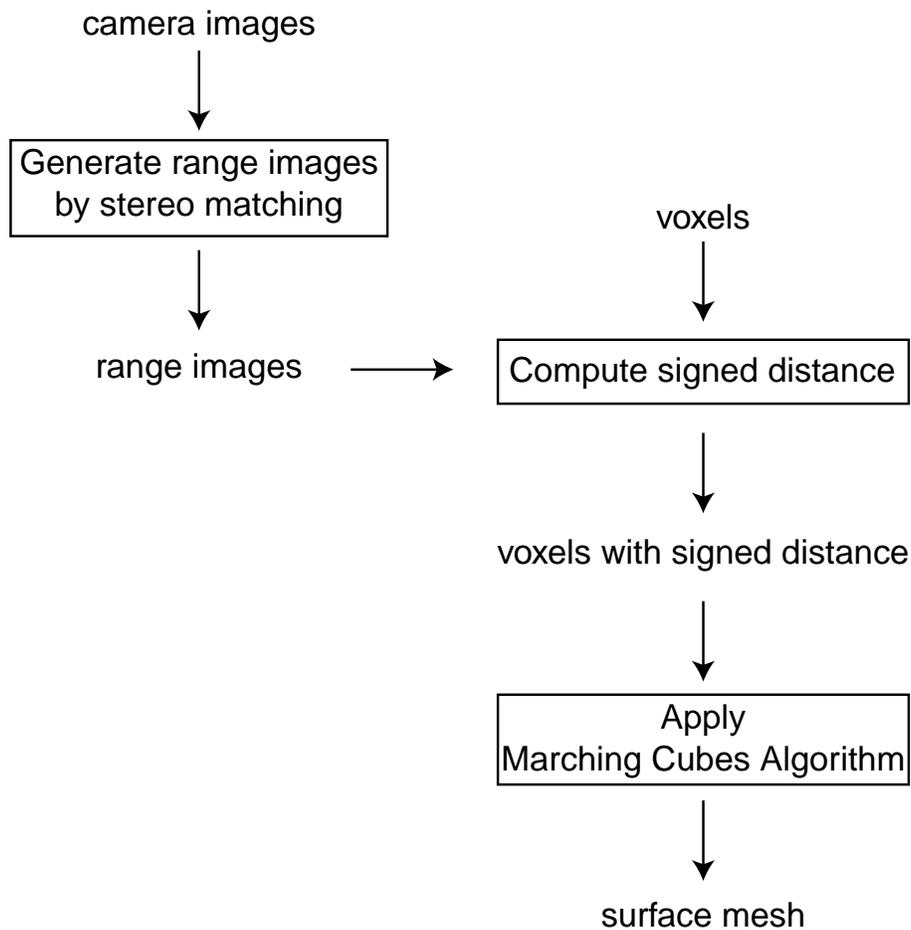


図 3.1: Modeling block diagram

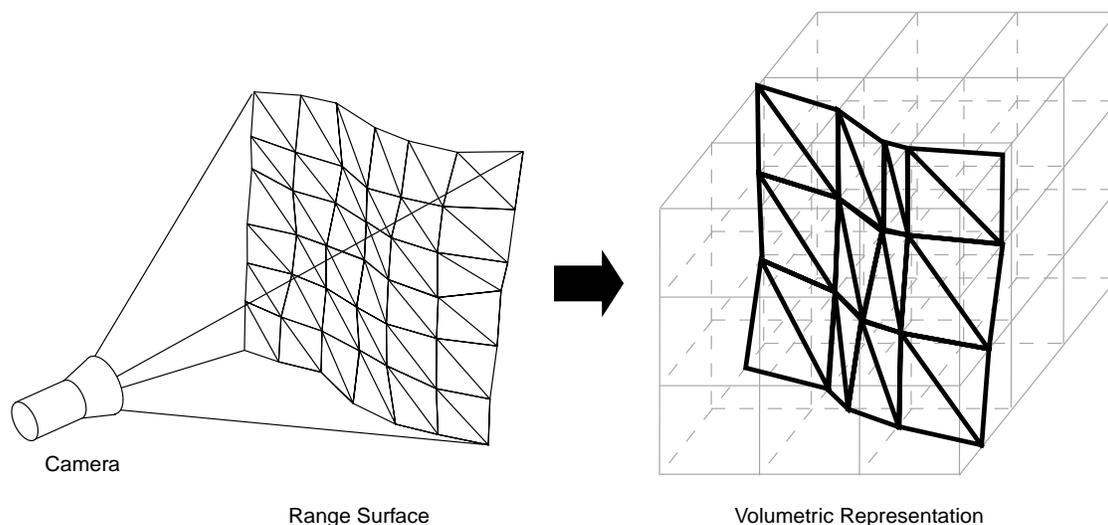


図 3.2: Projective representation to volumetric representation

とにする．すなわち，入力となるボクセルの集合 S_V は次のように表される．

$$S_V = \{V_i | i = 1 \dots N\} \quad (3.1)$$

ここで，頂点に与えられているスカラー値 $Z(v_j)$ は幾何的に次のような意味を持つ．

$$\begin{aligned} Z(v_j) \geq 0 & \text{ 頂点 } v_j \text{ は物体の外側，あるいはその表面上にある} \\ Z(v_j) < 0 & \text{ 頂点 } v_j \text{ は物体の内側にある} \end{aligned} \quad (3.2)$$

これによって，各頂点は物体の外側（あるいは表面上）か内側という2つの状態に分けられる．隣合う格子点の状態が異なればその間には物体の表面が横切っていることになる．その表面をメッシュで近似して表すと図.3.3 のようになる．表面の横切り方は1つの立方体の内部においては $2^8 = 256$ 通りの表面の横切り方があることがわかる．図.3.3 では256通りのうちで対象性を除いた14通りを表している．

頂点 v_{j_1} と頂点 v_{j_2} を結ぶ返上にメッシュの頂点がある場合，その頂点 p の位置は次のようにして計算する．

$$\mathbf{p} = \mathbf{v}_{j_1} + \mu(\mathbf{v}_{j_2} - \mathbf{v}_{j_1}), \quad \mu = \frac{-Z(\mathbf{v}_{j_1})}{Z(\mathbf{v}_{j_2}) - Z(\mathbf{v}_{j_1})} \quad (3.3)$$

Marching Cubes Algorithm のソースコードを付録 A に添付する．

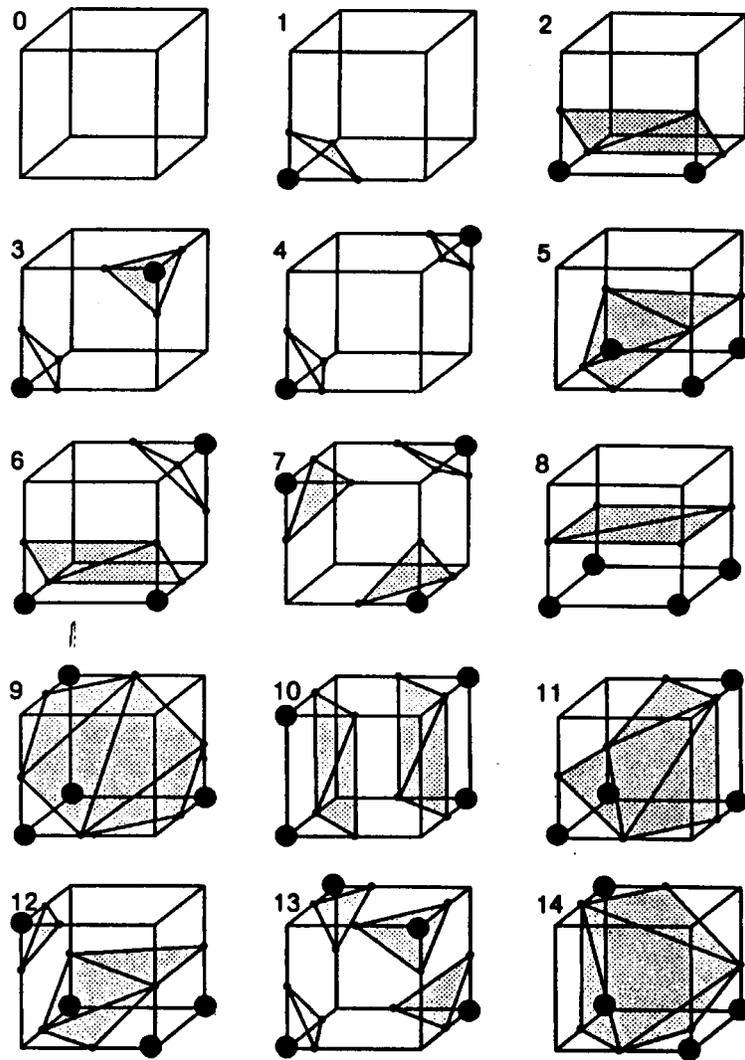


図 3.3: All patterns of intersection

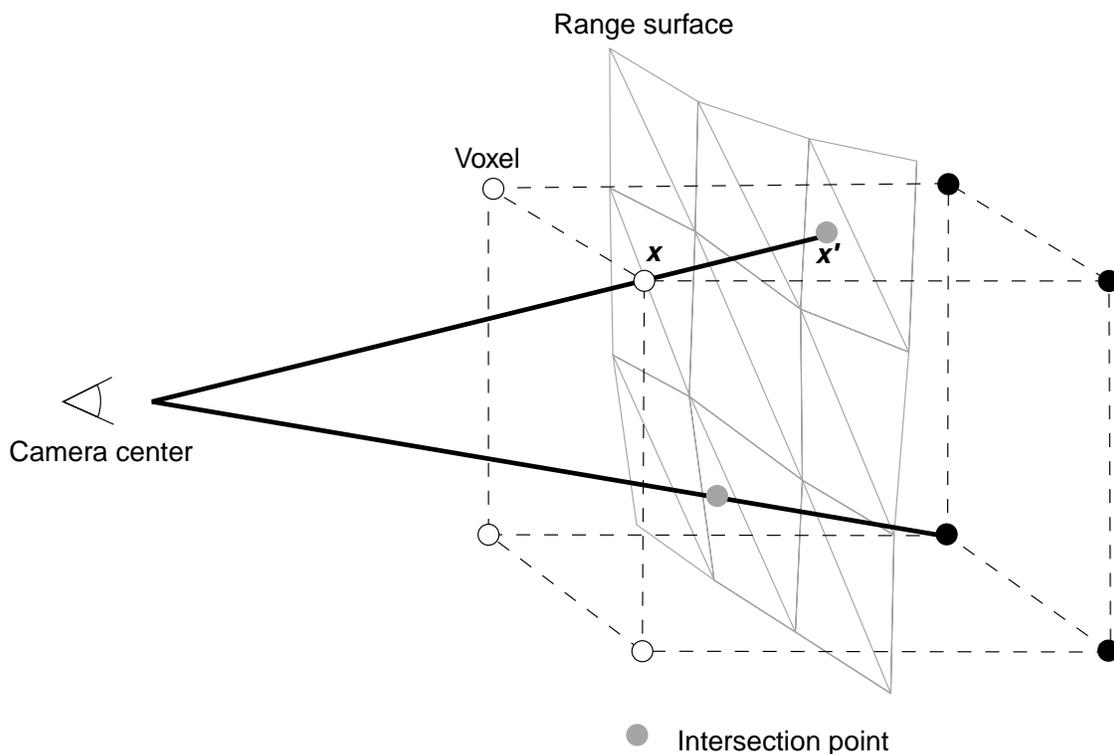


図 3.4: Computation of signed distance

3.1.2 符号付き距離の計算

さて, Marching Cubes Algorithm を用いて体積表現のメッシュを生成するには, 各ボクセルの頂点にスカラー値 $Z(v)$ を与える必要がある. [25, 23, 24] では, 距離画像を用いて $Z(v)$ を決定し, 観測した物体の形状を計算機の仮想空間内に再構成している. 本論文では, これらの研究の中で Curless らの手法 [25] を応用してメッシュモデルの生成を行なう.

[25] では次のように $Z(v)$ を計算する. 図.3.4 において, カメラの視点位置からボクセルの頂点 x に直線を伸ばし, 距離画像から得られる表面 (range surface) との交点を x' とする. このとき, $Z(x)$ は次の式で表される.

$$Z(x) = (x - x') \cdot l \quad (3.4)$$

ここで, l はカメラの光軸に平行な単位ベクトルである. すなわち, $Z(x)$ は, x が x' よりも手前にあれば正となり, そうでなければ負となる. またその絶対値は

頂点 \boldsymbol{x} から表面までの距離¹を表しているので, $Z(\boldsymbol{x})$ は「符号付き距離 (signed distance)」と呼ばれる.

実際の計算では交点 \boldsymbol{x}' を求めることはなく以下のように計算する. まず, 式 (2.1) から式 (2.6) をもちいて頂点 \boldsymbol{x} を変換し, カメラ座標系において (x_c, y_c, z_c) で表され, また画像座標系において (x_i, y_i) に射影されたとする. 点 (x, y) における距離画像の値 $D(x_i, y_i)$ を次のように線形補間する.

$$D(x_i, y_i) = (1-a)(1-b) \cdot D(i, j) + (1-a)b \cdot D(i, j+1) + a(1-b) \cdot D(i+1, j) + ab \cdot D(i+1, j+1) \quad (3.5)$$

$$i = \lfloor x_i \rfloor, \quad j = \lfloor y_i \rfloor, \quad a = x_i - i, \quad b = y_i - j$$

ここで $\lfloor x_i \rfloor$ は x_i より小さい最大の整数を表す.

同様に式 (2.1) から式 (2.6) をもちいて頂点 \boldsymbol{x}' を変換して, カメラ座標系において (x'_c, y'_c, z'_c) が得られたとすると, z'_c は次のようになる.

$$z'_c = D(x_i, y_i)$$

したがって, $Z(\boldsymbol{x})$ は次式で表される.

$$Z(\boldsymbol{x}) = z'_c - z_c = D(x_i, y_i) - z_c \quad (3.6)$$

3.1.3 複数の距離画像からの符号付き距離計算

3.1.1 節, 3.1.2 節において, 距離画像を用いて体積表現のメッシュモデルを生成する方法を説明した. 本節では複数の距離画像の情報を統合したメッシュモデルを生成するために, 複数の距離画像を用いて符号付き距離 $Z(\boldsymbol{v})$ を計算する方法を説明する.

M 枚の距離画像 $D_i (i = 1, \dots, M)$ を用いて, ボクセルの頂点 \boldsymbol{v} についての符号付き距離 $Z_i(\boldsymbol{v}) (i = 1, \dots, M)$ が計算できる. これらの符号付き距離の重み付き平均をとることにより, M 枚の距離画像の距離画像の統合して得られる符号付き距離 $V(\boldsymbol{v})$ は次のように表される. [25]

¹正確にはカメラの光軸に平行な成分の長さである.

$$V(\mathbf{v}) = \sum_i w_i(\mathbf{v}) Z_i(\mathbf{v}) \quad (3.7)$$

$$w_i(\mathbf{v}) = \begin{cases} 1 & |Z_i(\mathbf{v})| < T_w \\ \frac{T_w}{Z_i(\mathbf{v})} & otherwise \end{cases} \quad (3.8)$$

ここで $w_i(\mathbf{v})$ は重み係数, T_w は適当な閾値である. 本論文では $T_w = W_V$ (W_V はボクセル V の幅) を用いている.

距離画像が時系列に沿って得られる場合のように距離画像が漸次増えていく場合, $V(\mathbf{v})$ を次のようにインクリメンタルに更新することが可能である.

$$\begin{aligned} V_M(\mathbf{v}) &= \frac{W_{M-1}(\mathbf{v})V_{M-1}(\mathbf{v}) + w_M(\mathbf{v})Z_M(\mathbf{v})}{W_{M-1}(\mathbf{v}) + w_M(\mathbf{v})} \\ W_M(\mathbf{v}) &= W_{M-1}(\mathbf{v}) + w_M(\mathbf{v}) \end{aligned} \quad (3.9)$$

以上から, 複数の距離画像を用いて符号付き距離を計算し, それを Marching Cubes Algorithm に適用することによって体積表現を用いたメッシュモデルを生成することが可能になった.

3.2 アルゴリズムの高速化

Marching Cubes Algorithm を用いてモデルを生成するためには式 (3.1) におけるボクセルの集合 S_V に含まれる N 個のボクセルについて符号付き距離を計算する必要がある. ボクセルが3次元空間中に $A \times A \times A$ 個並んでいる場合, $N = A^3$ であり, 符号付き距離計算の計算量は $O(A^3)$ となる. これは, ロボット用視覚として用いることを考えた場合, 時系列にしたがって次々と与えられる距離画像を処理するには計算のコストが高い. したがって, 本論文では符号付き距離を計算するボクセルを限定することによって計算のコストを低くする.

3.2.1 octree を用いた階層的な体積表現

本論文では得られた距離画像の内容に応じて体積表現の解像度を調節してインクリメンタル性を保った処理を行なうため, 体積表現に単一の大きさではなく階層的に異なる大きさのボクセルを用いる. 本論文では, octree [19, 20, 38] を用いて階層的に大きさの異なるボクセルを表現する.

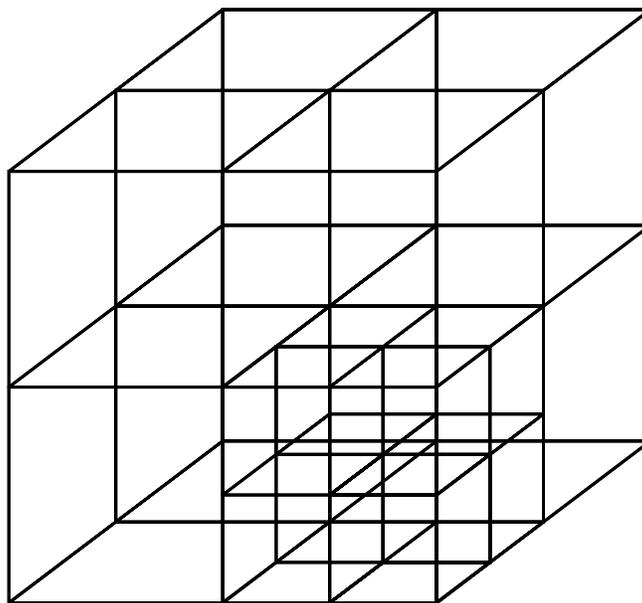


図 3.5: Hierarchical voxels using octree

図.3.5 に示すように octree を用いた表現によって階層的に大きさを変えたボクセルについて符号付き距離を計算しモデルを生成すると，図.3.6 のような表現力の異なるモデルが得られる．ボクセルの解像度を上げることにより表現力の高いモデルが得られるが，距離画像のノイズの影響を受けやすくなるため距離画像の性能に応じてボクセルの大きさを調節することが必要である．

3.2.2 符号付き距離を計算するボクセルの選択

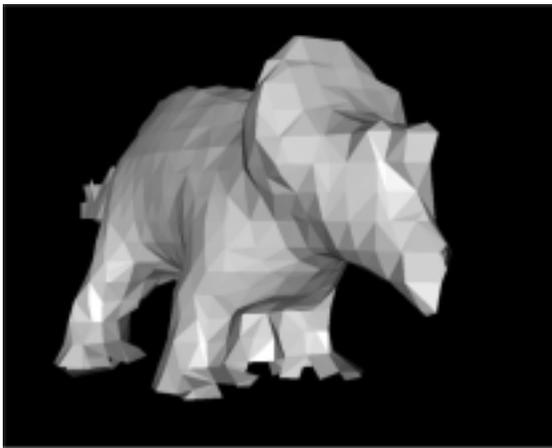
符号付き距離が計算されたボクセルの中で実際に，メッシュの生成に用いられるボクセルは，正負両方の符号付き距離を持つボクセルのみである．すなわち，観測対象の表面付近のボクセルのみ符号付き距離を計算すれば良い．本論文では，以下に説明する方法により符号付き距離を計算するボクセルを限定し，計算量の軽減をはかる．

まず，距離画像中の点の3次元空間での位置を計算することにより，観測対象表面が存在するボクセルを決定する．すなわち，符号付き距離を計算するボクセルの集合 S_V は次のようになる．

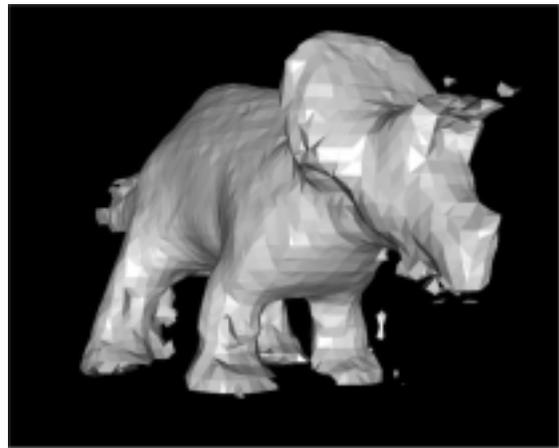
$$S_V = \{V_i | n > T_n, S_p = \{\mathbf{p}_j | \mathbf{p}_j \in V_i, j = 1 \dots n\}\} \quad (3.10)$$



A



B



C

図 3.6: Models by reconstructed using different resolution of voxel

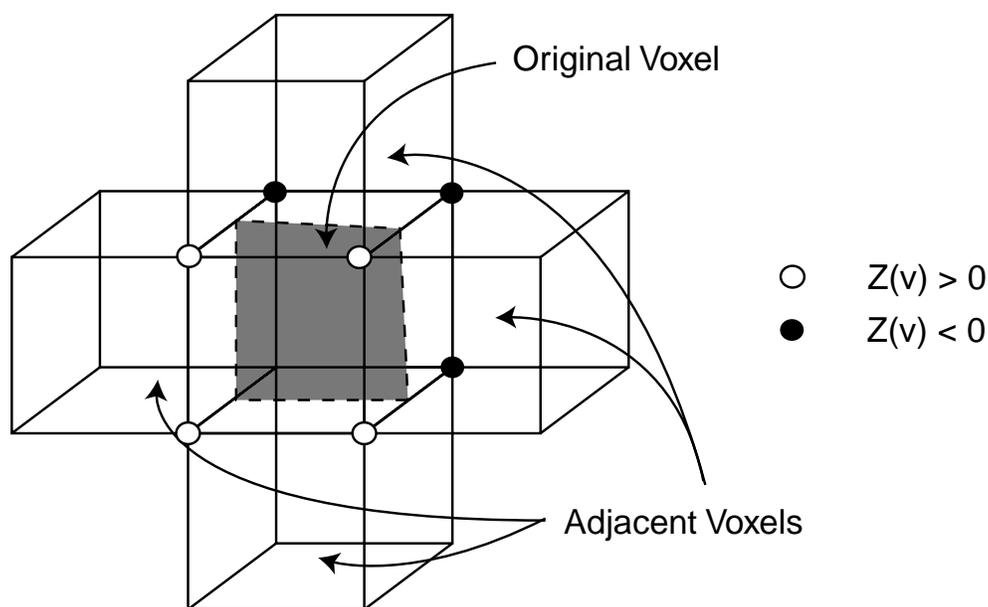


図 3.7: Add adjacent voxels to S_V

ここで p_j は距離画像から計算された点であり，ボクセル V_i 中にある点の数 n が閾値 T_n よりも多いボクセルを S_V に加えることを表す．

連続したメッシュモデルを生成するためには，距離画像を用いて得られたボクセルの集合 S_V のみでは不十分である． $V(v)$ を用いてメッシュを生成したときに，メッシュの頂点が存在する辺を共有する隣接したボクセルを S_V に加え符号付き距離を計算する．図.3.7では，図に示した隣接する4つのボクセルを S_V に加える．式で表すと次のようになる．

$$S_V \leftarrow S_V \cup \{V' \mid Z(\mathbf{v}_i)Z(\mathbf{v}_j) < 0, \mathbf{v}_i \in l_k, \mathbf{v}_j \in l_k, l_k \in V, l_k \in V'\} \quad (3.11)$$

ここで， V は元になるボクセル， V' は V に隣接したボクセル， l_k は V, V' 両方に属する辺， $\mathbf{v}_i, \mathbf{v}_j$ は l_k に属するボクセルの頂点である．

3.2.3 画像面の傾きによる閾値処理

あるボクセルにおいて，距離画像から計算されるメッシュモデルが観測視点に対して非常に傾いている場合，すなわち，ボクセルの8個の頂点について計算される $D(\mathbf{v}_j) (j = 1, \dots, 8)$ が大きく異なる場合，観測対象の表面は連続ではないと考えら

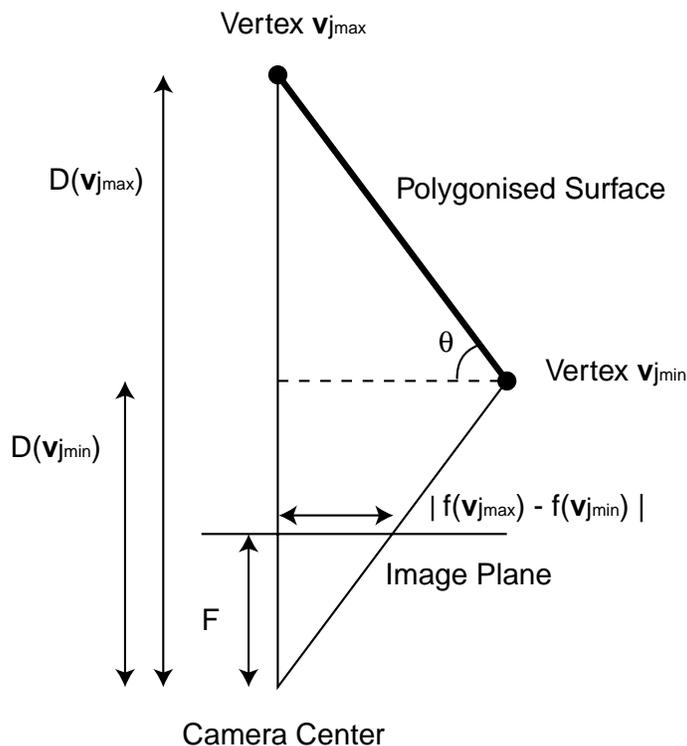


図 3.8: Thresholding by angle of polygonized surface

れる。

したがって、次の条件を満たすボクセルについてのみ $V(v)$ を更新する。

$$\frac{D(\mathbf{v}_{j_{min}})}{F} \cdot \frac{|f(\mathbf{v}_{j_{max}}) - f(\mathbf{v}_{j_{min}})|}{D(\mathbf{v}_{j_{max}}) - D(\mathbf{v}_{j_{min}})} > T_{\theta} \quad (3.12)$$

$$j_{max} = \arg \max_j D(\mathbf{v}_j), \quad j_{min} = \arg \min_j D(\mathbf{v}_j)$$

ここで、 F はカメラの焦点距離、 T_{θ} は閾値である。また、 $f(v)$ はカメラによる射影変換である (図.3.8)。この閾値処理によって図.3.9 に示すように、カメラに対して傾いた面について計算しないようにするものである。同様の閾値処理は [23, 5] で用いられている。本論文では $T_{\theta} = \tan(80^{\circ})$ を用いている。

3.2.4 インクリメンタルアルゴリズム

上述した内容をまとめて提案するアルゴリズムを以下に説明する。まず、 S_V のボクセルを選択し、ボクセルの符号付き距離を更新する手続き $UpdateOctTree$ と、符号付き距離を更新する手続き $UpdateSignedDistance$ は次のようになる。

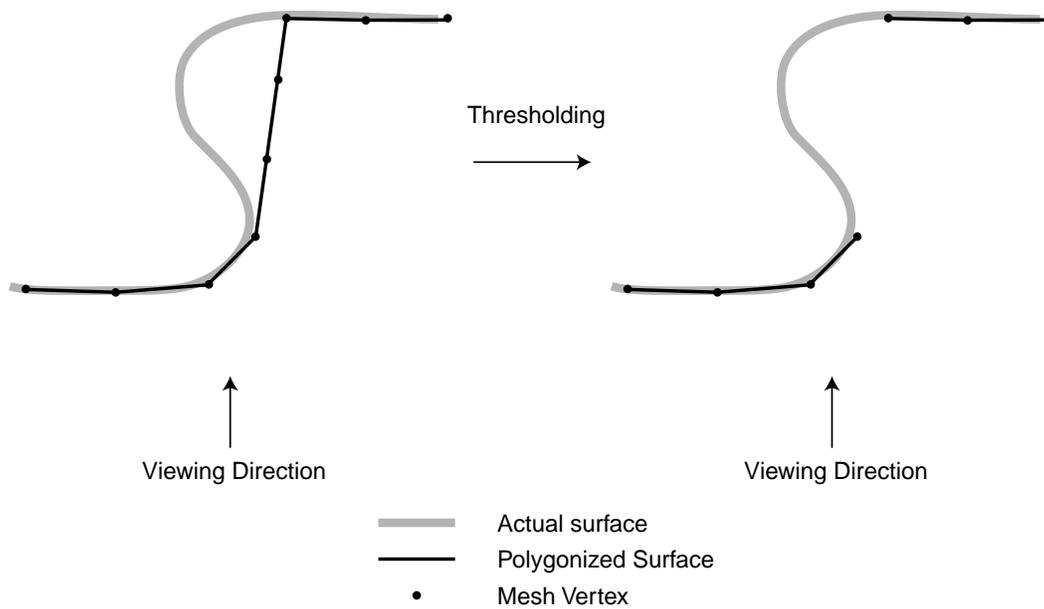


図 3.9: Avoid incorrectly instantiation of surface

Algorithm *UpdateOctTree*

Input: range image D

Input: OctTree O

1. **for** each $p_j \in D$
2. **do for** each $V_i \in O$
3. **do if** $p_j \in V_i$ $n_{V_i} \leftarrow n_{V_i} + 1$
4. **for** each $V_i \in S_V$
5. **do if** $n_{V_i} > T_n$ *UpdateSignedDistance*(D, V_i)

Algorithm *UpdateSignedDistance*

Input: range image D

Input: voxel V

1. **for** each $v_i \in V, i = 1, \dots, 8$
2. **do** compute $D(v_i)$
3. $j_{max} \leftarrow \arg \max_j D(v_j), \quad j_{min} \leftarrow \arg \min_j D(v_j)$
4. **if** $\frac{D(v_{j_{min}})}{F} \cdot \frac{|f(v_{j_{max}}) - f(v_{j_{min}})|}{D(v_{j_{max}}) - D(v_{j_{min}})} > T_\theta$
5. **then** ↩

6. **for** each $\mathbf{v}_i \in V, i = 1, \dots, 8$
7. **do** compute $Z(\mathbf{v}_i), w(\mathbf{v}_i)$
8. $W(\mathbf{v}_i) \leftarrow W(\mathbf{v}_i) + w(\mathbf{v}_i)$
9. $V(\mathbf{v}_i) \leftarrow \frac{W(\mathbf{v}_i)V(\mathbf{v}_i) + w(\mathbf{v}_i)Z(\mathbf{v}_i)}{W(\mathbf{v}_i)}$
10. **for** each adjacent voxel V' which surface intersects
11. **do** $UpdateSignedDistance(D, V')$

10, 11 行目において, 式 (3.11) によって加えられる隣接したボクセルについて再帰的に $UpdateSignedDistance$ を呼びだし, 符号付き距離を計算する. ここで注意点としてこの疑似コードには, ボクセルの各頂点 v の符号付き距離を重複して計算しないようにするコードが含まれていない. 実際のプログラムでは1回の更新において各頂点の符号付き距離は1回のみ更新されるようにする.

複数の距離画像を用いてモデルを生成する手続きは次のようになる. この中で用いられている手続き $MarchingCubes$ は octree O に属するボクセルに $MarchingCubes$ Algorithm を適用してメッシュモデル T を返す手続きである. (付録 A 参照)

Algorithm *ReconstructSurface*

Input: range image $D_i, i = 1, \dots, M$

Output: mesh model T

1. **for** each $D_i, i = 1, \dots, M$
2. **do** $UpdateOctTree(D_i, O)$
3. $T \leftarrow MarchingCubes(O)$
4.  T

3.2.5 アルゴリズムの考察

式 (3.10) を用いて符号付き距離を計算するボクセルを限定することにより計算量の軽減をはかった. この方法を用いた場合, 計算するボクセルは観測対象表面付近のボクセルのみとなる. $A \times A \times A$ 個のボクセルの中で, 観測対象の表面が存在するボクセルの数は $O(A^2)$ であると期待できる [24]. したがって, 符号付き距離の計算量は $O(A^3)$ から $O(A^2)$ に軽減されたことになる.

この手法によって実際に観測対象の表面が存在するボクセルにメッシュを生成するためには, 式 (3.10) による限定によって, それらのボクセルを取り除かないようにしなければならない. 距離計測が正確な場合には, ボクセルの大きさを小さくし

ても計算すべきボクセルを取り除いてしまうことはない。しかし実際の距離計測には誤差が存在するために、ボクセルサイズを小さくし過ぎた場合、実際に表面が存在するボクセルと距離画像から得られる点が存在するボクセルが異なってしまい、式(3.10)によって計算すべきボクセルが取り除かれてしまう。したがって距離計測の精度を見積り、有効なボクセルサイズを決定する必要がある。

ステレオ視による距離計測では、式(2.15)から距離 Z の点においては、次のように ΔZ の誤差が存在する可能性がある。

$$\Delta Z = \frac{\Delta d}{d + \Delta d} \cdot Z \quad (3.13)$$

したがって、カメラから距離 Z において、モデル生成に用いるボクセルの大きさは ΔZ よりも大きくとるべきである。ここで、距離 Z における幅 W のボクセルの画像上での幅 w は次のように表される。

$$w = \frac{F}{d_x \cdot Z} \cdot W \quad (3.14)$$

したがって、このボクセルの式(3.10)における点の数 n は w^2 に比例するとみなせる。式(3.10)に用いている閾値を $T_n = \alpha^2 w^2$ とすると、 $W > \Delta Z$ なるボクセルを S_V に加えるためには、式(3.13)、式(3.14)から T_n が次式を満たせばよい。

$$T_n > \left(\alpha \cdot \frac{F}{d_x} \cdot \frac{\Delta d}{d + \Delta d} \right)^2 \quad (3.15)$$

カメラのパラメータ $F = 10.075(\text{mm})$, $d_x = 0.0441(\text{mm})$ の実測値を用いると、視差の分解能 $\Delta d = 1$ として、視差 $d > 9$ において計測する場合には、

$$T_n > 521.9 \times \alpha^2$$

となる。本論文では $\alpha = 0.5 \sim 1.0$ として T_n を設定している。 $\alpha = 0.5$ ならば $T_n = 131$, $\alpha = 1.0$ ならば $T_n = 522$ である。

この閾値 T_n の見方を変えると、センサのノイズに対するパラメータとも考えられる。例えば $T_n = 100$ の場合、画像上で約 10×10 画素以上の大きさに射影されるボクセルに対してのみ符号付き距離を計算することになる。距離計測ミスによって外れ値が距離画像に存在したとしても、その面積が T_n 以下ならば除外することができる。

上述の2つの理由からは T_n を大きくとった方が良いが、逆に、 T_n を大きく設定するとモデルの解像度が粗くなるため、構築するモデルの解像度と、誤差に対するトレードオフによって T_n を決定する。

3.3 実験

距離画像からモデルを生成するためにはカメラ位置を測定する装置が必要である。また、実画像を用いてステレオ視を行なった場合にはテクスチャの問題から対応点が探索できない問題や、ジャンプエッジ部分²で対応点探索が不正確になる問題が存在する。そこで、まず仮想環境を用いて人工的に距離画像を生成し、ノイズのない環境でアルゴリズムが有効であることを確かめ、その後実画像を用いた実験を行なう。

3.3.1 人工距離画像の生成

本論文では、コンピュータグラフィックスで一般に用いられている3次元グラフィックス環境 OpenGL を用いて仮想3次元空間を構築して仮想的なカメラ画像を生成する。そこで OpenGL の機能を利用して、生成されたカメラ画像に対する距離画像を生成する。

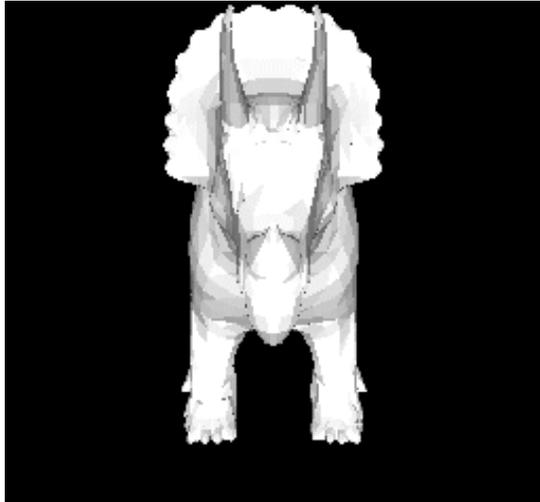
人工距離画像の生成では2.1.4節で述べたステレオ視の距離計測分解能を考慮し、仮想的なカメラパラメータと基線長を与えて人工距離画像の距離データを量子化する(図.3.10)。

3.3.2 仮想環境を用いた実験

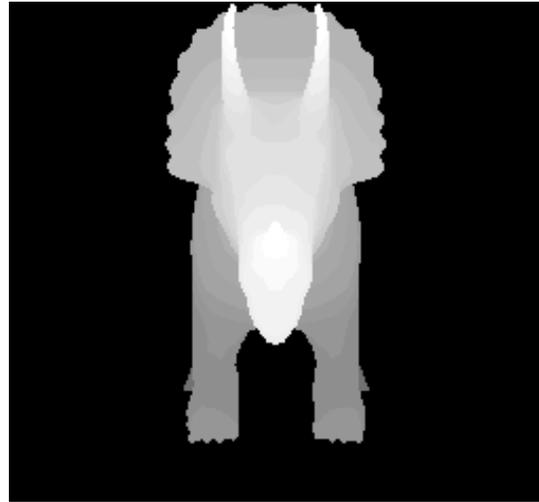
図.3.11に示すテーブルの上いくつかの物体が乗ったものを観測対象のシーンとする。テーブルは仮想環境において直径2mという設定である。このシーンを次の条件において観測し図.3.12に示した167枚の画像を得た。これを用いて距離画像を生成し(図.3.13)、提案した手法を適用してメッシュモデルを構築した。

- 焦点距離 $F = 10.075(\text{mm})$ 。
- 投影面上における1画素あたりの幅 $d_x = 0.0441(\text{mm})$, $d_y = 0.0353(\text{mm})$ 。

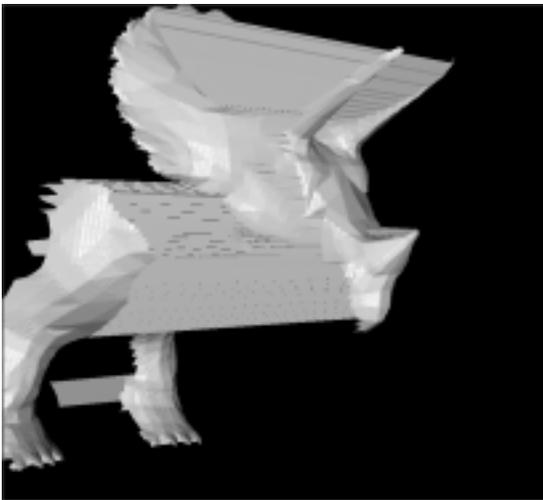
²画像中で距離 (depth) が不連続に大きく変化する部分



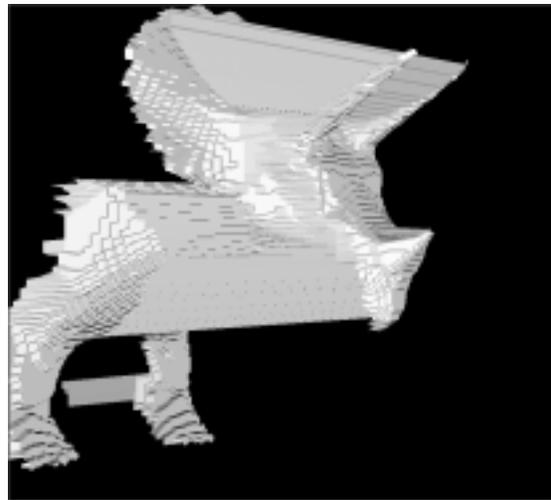
Virtual Model Image



Artificial Range Image



Ideal Range Surface



Quantized Range Surface

図 3.10: Range image generation using virtual model

- 画像の幅 $W_i = 256$ (画素), 高さ $H_i = 240$ (画素) .
- 基線長 $B = 0.12$ (m) . サブピクセル精度の視差計測をしないと仮定する .
- モデリングパラメータ $T_n = 400$.

図.3.14 に距離画像が加えられていく過程でインクリメンタルに更新されるモデルの様子を示す . 図.3.15 にボクセルの大きさ W が異なるモデリング結果を示す . 図.3.15 では , パラメータ離れた場所からのみ観測されている部分ではモデルの生成が行なわれず , ボクセルの大きさが小さくなる毎にモデルの生成されている部分が小さくなっている . PentiumIII 450MHz プロセッサを用いて計算を行ない , 1 枚の画像が加えられた時の符号付き距離の計算時間は平均 191(msec) であった . 通信など他の計算時間も含めて , 167 枚の距離画像を処理するのにかかった計算時間は 82(sec) であった .

本論文で提案した計算量削減の手法を用いない場合 , すなわち , 3 次元空間中に配置されている全てのボクセルについて符号付き距離の計算をした場合のモデリング結果を図.3.16 に示す . 図.3.16 では , $-1 < x < 1, -1 < z < -1, 0.8 < y < 1.8$ 内に存在する幅 $W = 0.078125$ (m) のボクセル全てについて符号付き距離の計算を距離画像毎に行なったものである . 符号付き距離の計算を行なったボクセルの数は距離画像 1 枚あたり平均 67626 個であり , PentiumIII 450MHz プロセッサを用いて平均 3982(msec) であった . ボクセルの限定部分以外の条件は同じである . それに対し本論文で提案したボクセルの限定を行なった場合 , 符号付き距離の計算を行なったボクセルの数は距離画像 1 枚あたり平均 506 個であった . 誤対応のない距離画像の場合において , 本論文で提案した計算量削減の方法は有効であり , 計算時間を 10 分の 1 以下にすることが示された .

3.3.3 実画像を用いた実験

図.3.17 に距離画像から得られる点を 3 次元にプロットしたものと , これに提案した手法を適用して再構成した観測対象をポリゴンで表したものを示す . 図.3.17 は , 距離画像 7 枚を用いて得られた約 2500 ポリゴンの再構成結果である . 1 つのポリゴンは一辺約 5mm の大きさである . この再構成結果を得るまでに , 距離画像が追加されていくにしたがって , 再構成されていく様子を図.3.18 に示す .

距離画像がインクリメンタルに増えるたびにかかる計算時間は PentiumIII 500MHz プロセッサ (Linux 2.2.9) を用いて平均約 200ms であった .

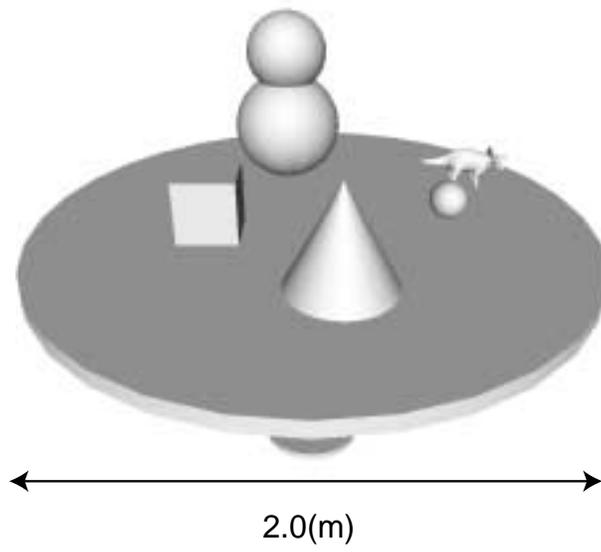


図 3.11: Virtual model of objects on a table

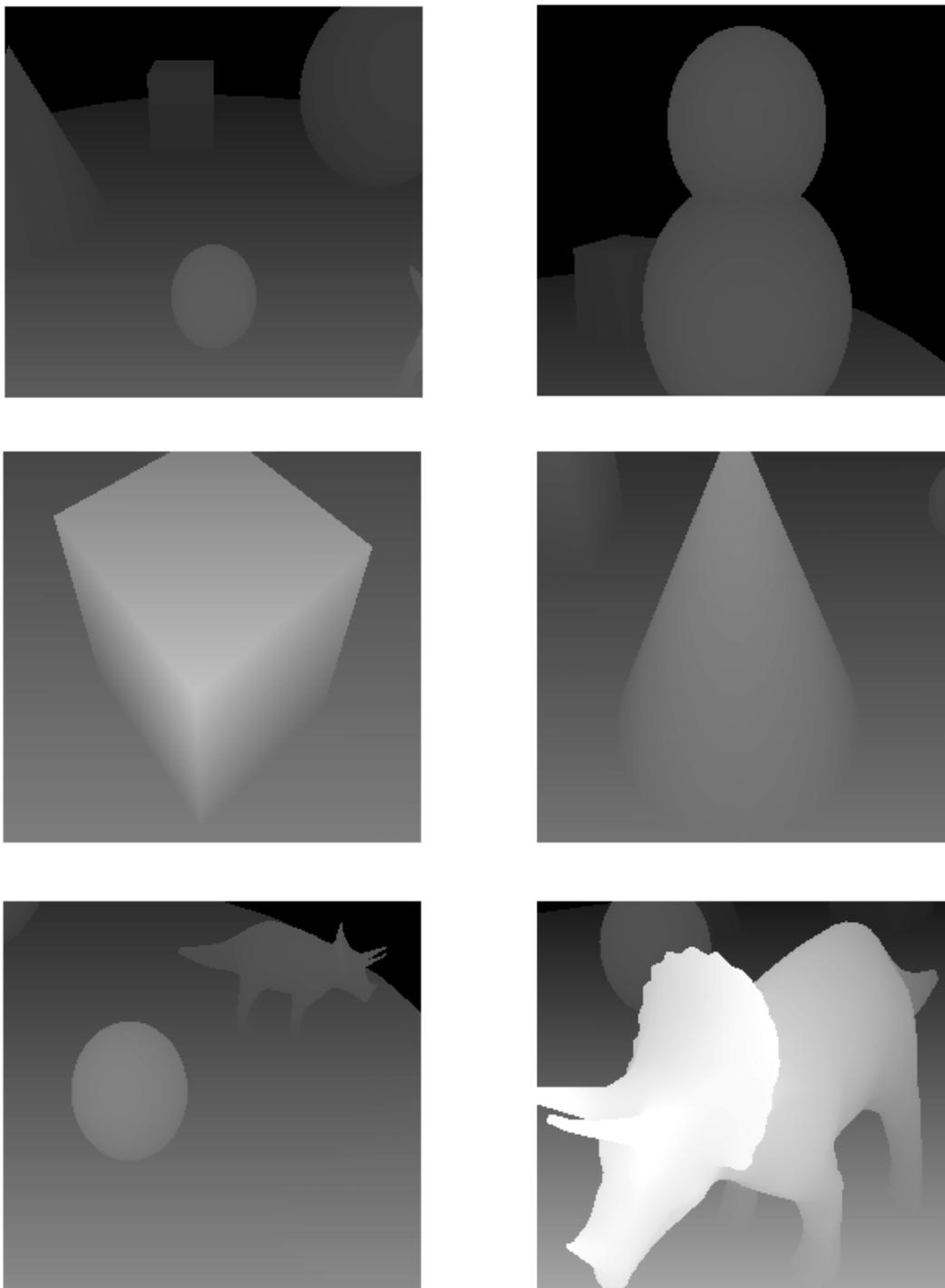
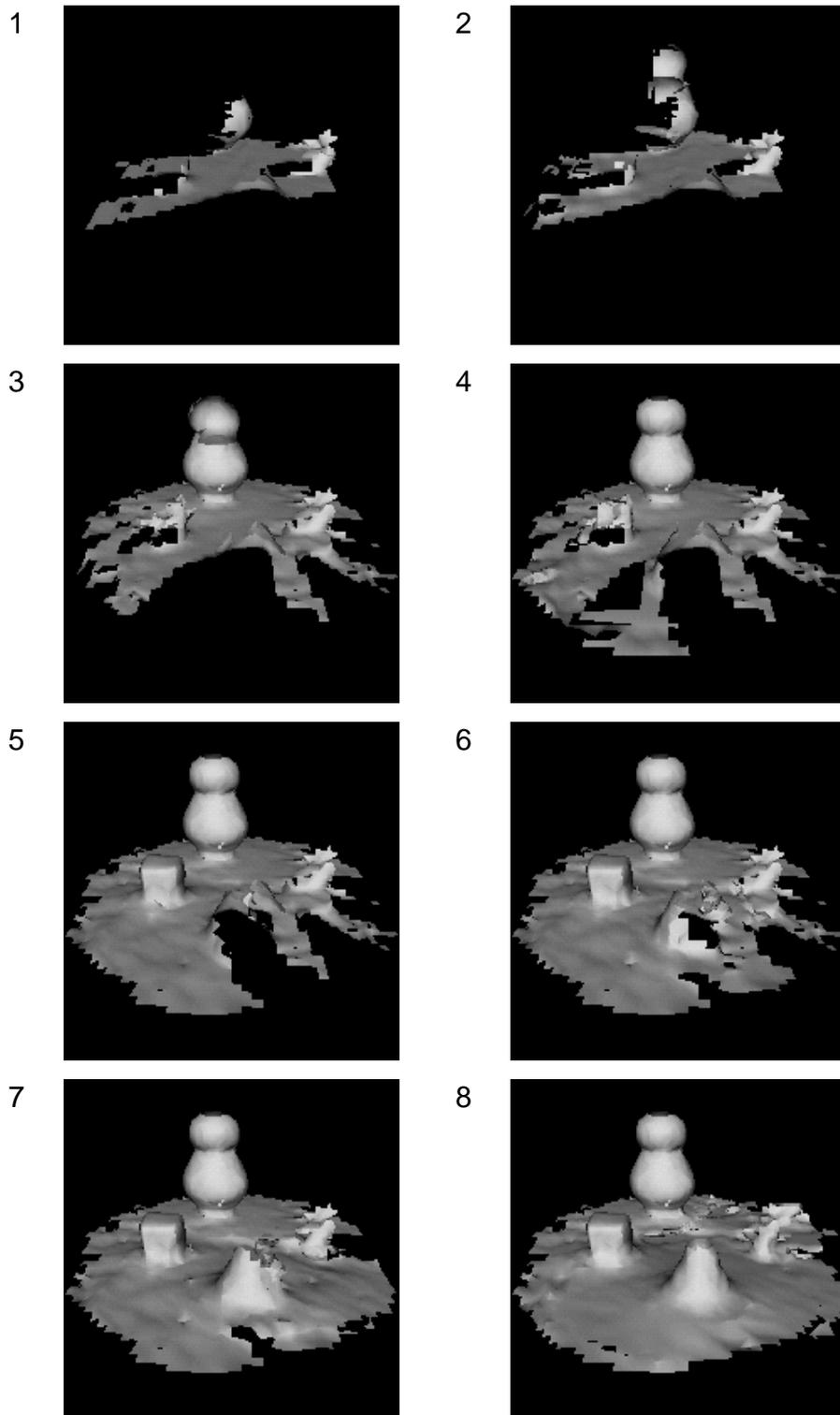
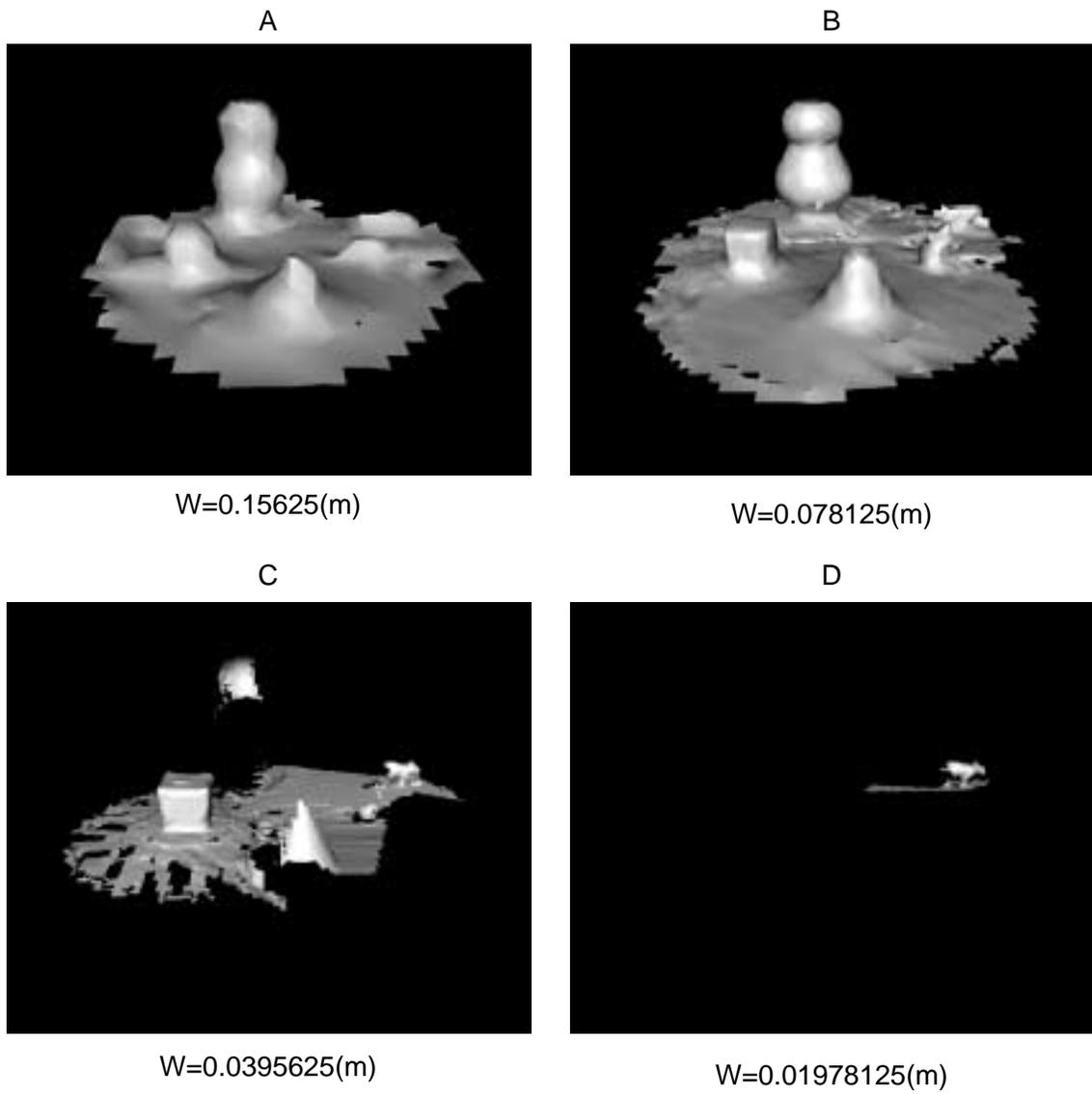


図 3.13: Artificial range images of objects on a table



$W = 0.078125(m)$

図 3.14: Incremental updating of model



W : voxel width

☒ 3.15: Generated mesh models by different voxel size

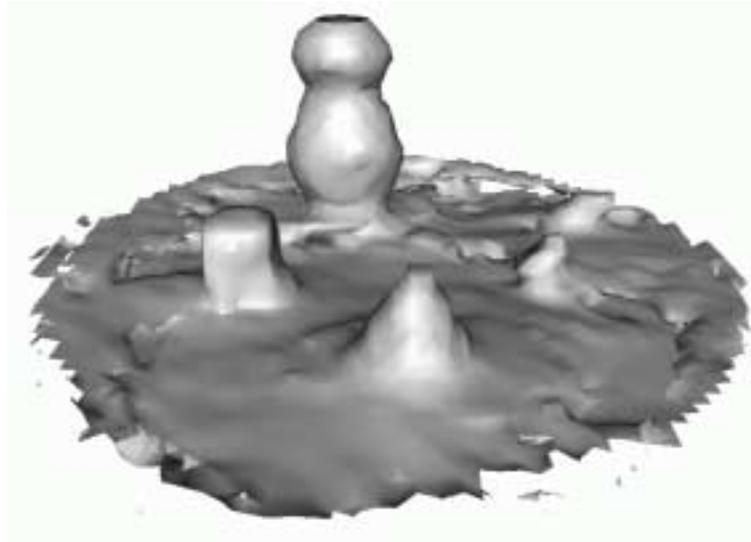


図 3.16: Generated mesh models by computing signed distance of all voxels

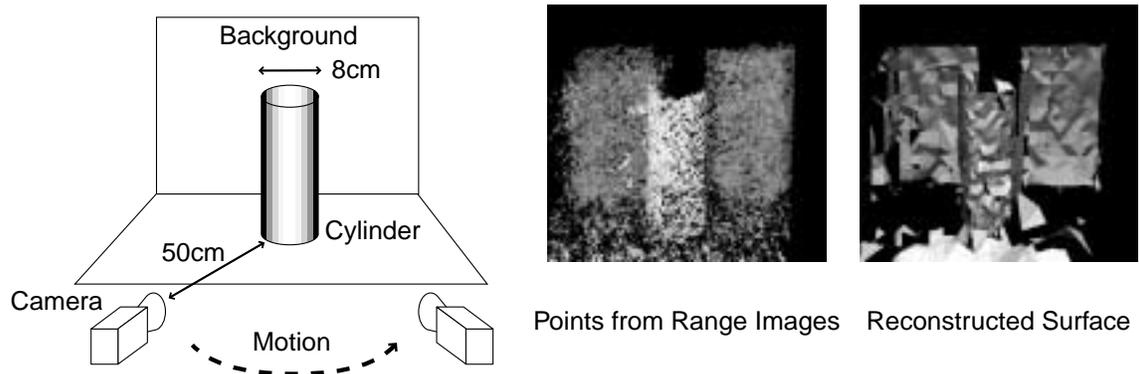


図 3.17: Reconstruction of the object surface from multiple range images

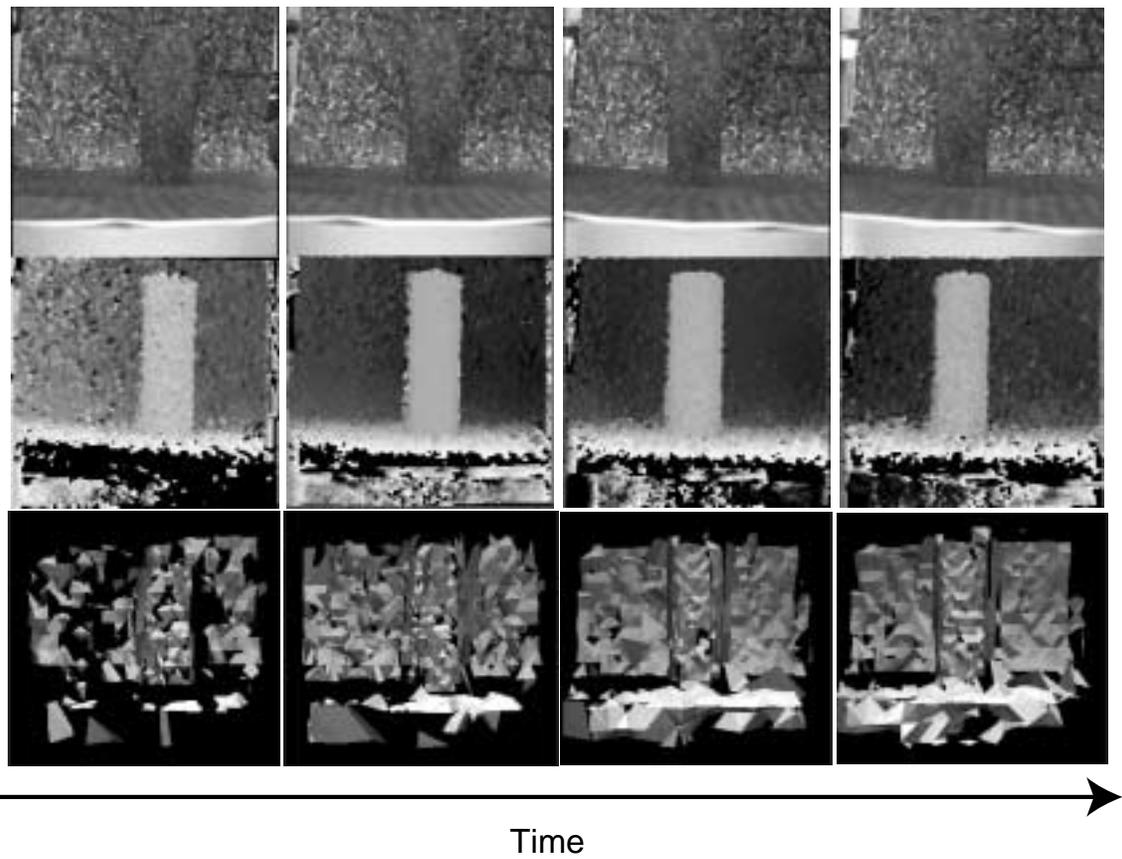


図 3.18: Improvement of reconstructed surface

第 4 章

3次元物体認識

本章では、第3章において構築したモデルを用いて3次元物体の認識を行なう、ここでいう認識とは、新しく得た観測結果とあらかじめ持っている知識を関連づけることである。あらかじめ観測対象のモデルを持っており認識に利用する方法はモデルベース認識と呼ばれる。モデルベース認識の方法の1つとして、観測対象の位置姿勢計測による認識がある [39, 40, 7, 41, 5]。位置姿勢計測のためのステップは次のようになる。まず、認識のためのモデルを構築し記憶する。次に、現在の観測(シーン)をモデルと比較可能なように同じ表現で表し、モデルとシーンで似た部分の間に対応関係を探索する。そして、対応関係を用いてシーンに対するモデルの位置姿勢を計算する。

対応関係を得るための比較の手法は様々存在するが、ロボットに課せられる認識タスクではシーン内に複数の対象が混在していると考えられ、そのような環境下で対応をとることができる手法を用いなければならない。比較のための特徴量は、モデル全体にわたる大域的な特徴量(体積, 表面形状のパラメータなど)と、局所的な特徴量(エッジなど)に分けられる。大域的な特徴量は位置姿勢計測には有効で曖昧性の少ない特徴量であるが、複数の物体が混在するシーンでは特徴量を抽出することが難しい。局所的な特徴量はこのような、大域的な特徴量を計算することが困難なシーンでは有効であるが、局所的な特徴量の抽出はノイズに弱いという欠点がある。

本論文では特徴量として、Johnsonらによって提案されているスピンイメージ(Spin-Image)を用いる手法 [27] を用いて、複数の対象が混在するシーンにおいて物体の認識を行なう。以下では、まずスピンイメージとそれを用いた認識のアルゴリズムについて説明する。次に前章において生成した階層的なモデルに対して適用するアルゴリズムを示す。

4.1 スピンイメージ

4.1.1 位置姿勢に独立な座標系への変換

メッシュモデルによって表されている2つのモデル間でその頂点同士を比較する特徴として、頂点の周囲のメッシュの形状が挙げられる。しかし3次元の形状を比較するためには、位置姿勢の6次元のパラメータについて調べる必要がある。ここでメッシュモデルの各頂点は表面上の点であることから、それぞれ法線を持っていることに注目し、次の式を用いて3次元上の点を2次元に射影することにより、頂

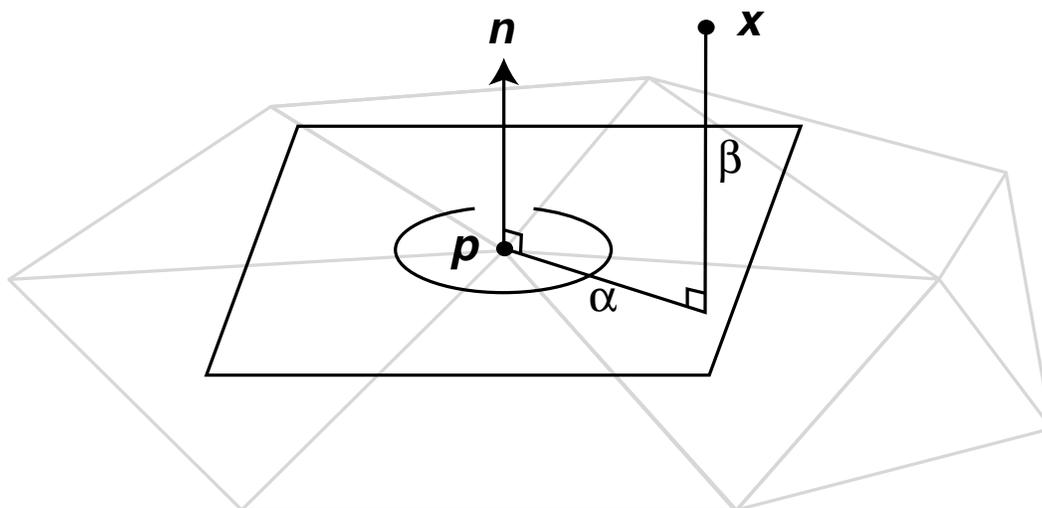


図 4.1: Object-centered coordinate system

点の法線に相対的な座標系 (α, β) に変換する (図.4.1) . 法線相対な座標系を用いると, モデルの形状について位置姿勢に対して独立な表現が得られ, 形状の比較に大きな利点を持つ .

$$S_O : \mathbf{R}^3 \rightarrow \mathbf{R}^2 \quad (4.1)$$

$$S_O(\mathbf{x}) \rightarrow (\alpha, \beta) = (\sqrt{\|\mathbf{x} - \mathbf{p}\|^2 - (\mathbf{n} \cdot (\mathbf{x} - \mathbf{p}))^2}, \mathbf{n} \cdot (\mathbf{x} - \mathbf{p}))$$

ここで, \mathbf{p} は基準となる頂点 O の位置, \mathbf{n} は頂点 O における法線ベクトルである . 式 (4.1) によって定義された射影 S_O をスピスマップ (Spin-Map) と呼ぶ .

スピスマップによってモデルの各頂点を射影すると, 幾何的には法線の周りに面を一周させ頂点を掃引した 2 次元の像が得られる . (図.4.2) これが「Spin-Image」の名前の由来である .

4.1.2 スピイメージの生成

スピイメージを比較に用いるために 2 次元配列として表す . 式 (4.1) によってモデルの各点を射影して得られた (α, β) を図.4.3 に示すように 2 次元で補間し, 量子化した 2 次元配列の要素 $I(i, j)$ に加算することによってスピイメージを生成する . そのアルゴリズムを疑似コードで表すと次のようになる .

Algorithm *CreateSpinImage*

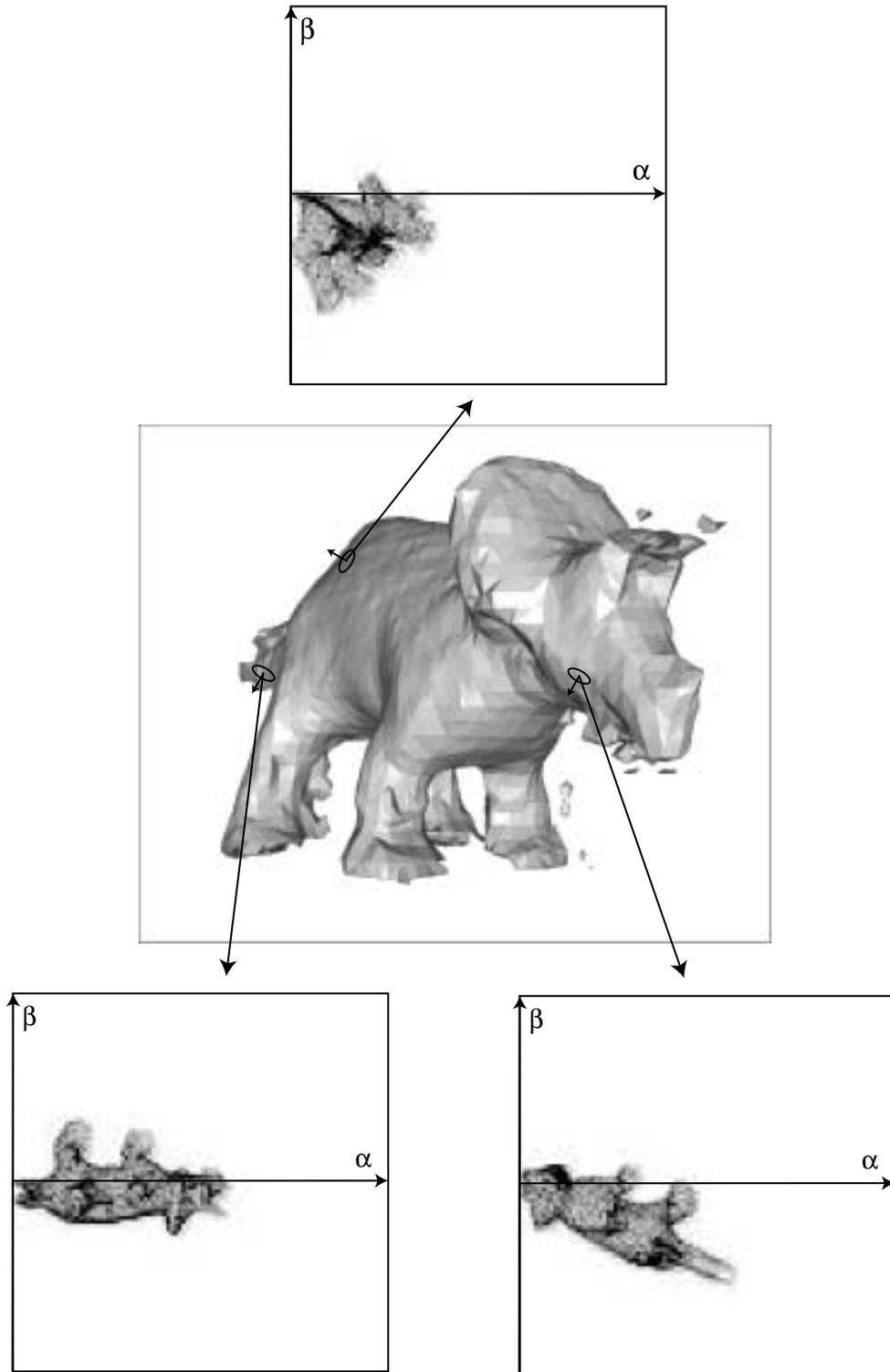


図 4.2: Generated spin-image around oriented-point

Input: oriented-point \mathbf{O}

Input: surface mesh M

Output: spin-image I

1. **for** each point $\mathbf{x} \in M$
2. **do** $(\alpha, \beta) \leftarrow SpinMapCoordinate(\mathbf{O}, \mathbf{x})$ (* 式 (4.1) *)
3. $(i, j) \leftarrow SpinImageBin(\alpha, \beta)$ (* 式 (4.2) *)
4. $(a, b) \leftarrow BilinearWeights(\alpha, \beta)$ (* 式 (4.3) *)
5. $I(i, j) \leftarrow I(i, j) + (1 - a) * (1 - b)$
6. $I(i + 1, j) \leftarrow I(i + 1, j) + (a) * (1 - b)$
7. $I(i, j + 1) \leftarrow I(i, j + 1) + (1 - a) * (b)$
8. $I(i + 1, j + 1) \leftarrow I(i + 1, j + 1) + (a) * (b)$

ビンサイズ b と画像幅 W の2つのパラメータを用いて量子化の式 (4.2) と補間の係数を求める式 (4.3) は次のようになる。さらに、スピニメージ生成のパラメータとしてサポート角 A_s がある。基準となる点 $A(p_A, \mathbf{n}_A)$ に対して、式 (4.4) を満たす点 $B(p_B, \mathbf{n}_B)$ のみをスピニメージに加える閾値として用いる。これらのパラメータを調節することにより、1つのスピニメージが含むモデルの大域的な特徴量と局所的な特徴量を調節し、複数の物体が混在するシーンでの認識を可能にする。

$$i = \left\lfloor \frac{\frac{W}{2} - \beta}{b} \right\rfloor \quad j = \left\lfloor \frac{\alpha}{b} \right\rfloor \quad (4.2)$$

$$a = \frac{W}{2} - \beta - ib \quad b = \alpha - jb \quad (4.3)$$

$$\arccos(\mathbf{n}_A \cdot \mathbf{n}_B) < A_s \quad (4.4)$$

4.1.3 スピニメージの比較

スピニメージは観測対象の形状を位置姿勢と独立に記述するため、同じ物体は異なる位置姿勢であっても、スピニメージ生成のパラメータが等しければ同じスピニメージを持つ。したがって、スピニメージ同士を直接比較することにより物体の点の間の対応が得られる。テンプレートマッチングの方法から、正規化相関

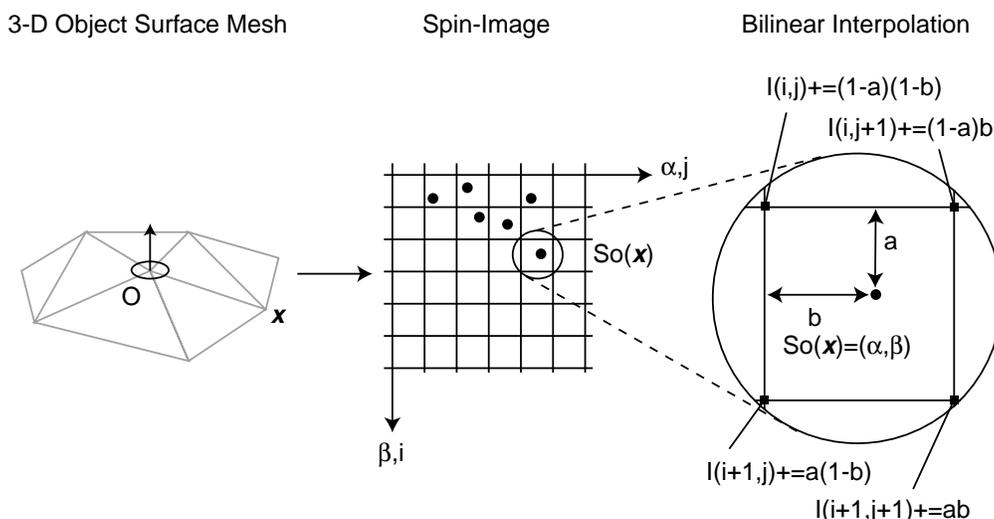


図 4.3: The addition of a point to the 2-D array representation of a spin-image

演算を比較の基準として用いて次の式のように2つのスピニメージ P, Q 間の相関係数 $R(P, Q)$ を計算する .

$$R(P, Q) = \frac{N \sum p_i q_i - \sum p_i \sum q_i}{\sqrt{(N \sum p_i^2 - (\sum p_i)^2)(N \sum q_i^2 - (\sum q_i)^2)}} \quad (4.5)$$

ここで, N はスピニメージのピンの数である . R は -1 (負の相関) から 1 (正の相関) の間の値を取り, R の値が大きければ2つのスピニメージは似ており, R の値が小さければ2つのスピニメージは似てないことになる . このようにして良い対応関係と悪い対応関係の区別が可能になった .

また, スピニメージのどちらかの画像が値を持たない画素では相関係数の計算に加えない . すなわち, 2つの画像が重なっている部分のみにおいて相関係数の計算を行なう . また, 相関係数の計算に用いられる画素の数が多いほど相関係数の信頼性が高いので, 比較の基準に画素の数を考慮に入れる . その信頼性を分散によって測ることができるので, 相関係数 R とその分散を1つの関数にまとめた次の比較基準を用いる .

$$C(P, Q) = (\operatorname{atanh}(R(P, Q)))^2 - \lambda \left(\frac{1}{N-3} \right) \quad (4.6)$$

まず, 統計学的手法により, ハイパボリックアークタンジェント (atanh^1) を用いて変数変換し, 変換した変数の分散が $1/(N-3)$ となる [42]. 重み係数 λ は, 2つのスピニメージが重なっている画素数が λ と比べて非常に大きければ式 (4.6) の第2項は無視でき, λ と比べて非常に小さければ式 (4.6) の値は第2項に支配される.

4.2 スピニメージを用いた認識アルゴリズム

スピニメージを用いた認識アルゴリズムを以下に説明する. 認識の目的は観測対象の位置姿勢を求めることであり, 結果としてモデルからシーン中の対応する部分までの平行移動と, 回転のパラメータが得られる. 認識のステップは次のようになる.

1. スピニメージのマッチングにより対応点探索を行なう.
2. 幾何拘束を用いてマッチングの結果をフィルタリングし, 矛盾のない対応の組み合わせをグループ化する.
3. グループ化した対応の組合せから位置姿勢のパラメータを求める. モデルの全ての頂点に対して対応点探索を行ない位置姿勢のパラメータを検証する. ICP 法 [39, 40] を用いてパラメータの改良を行なう.

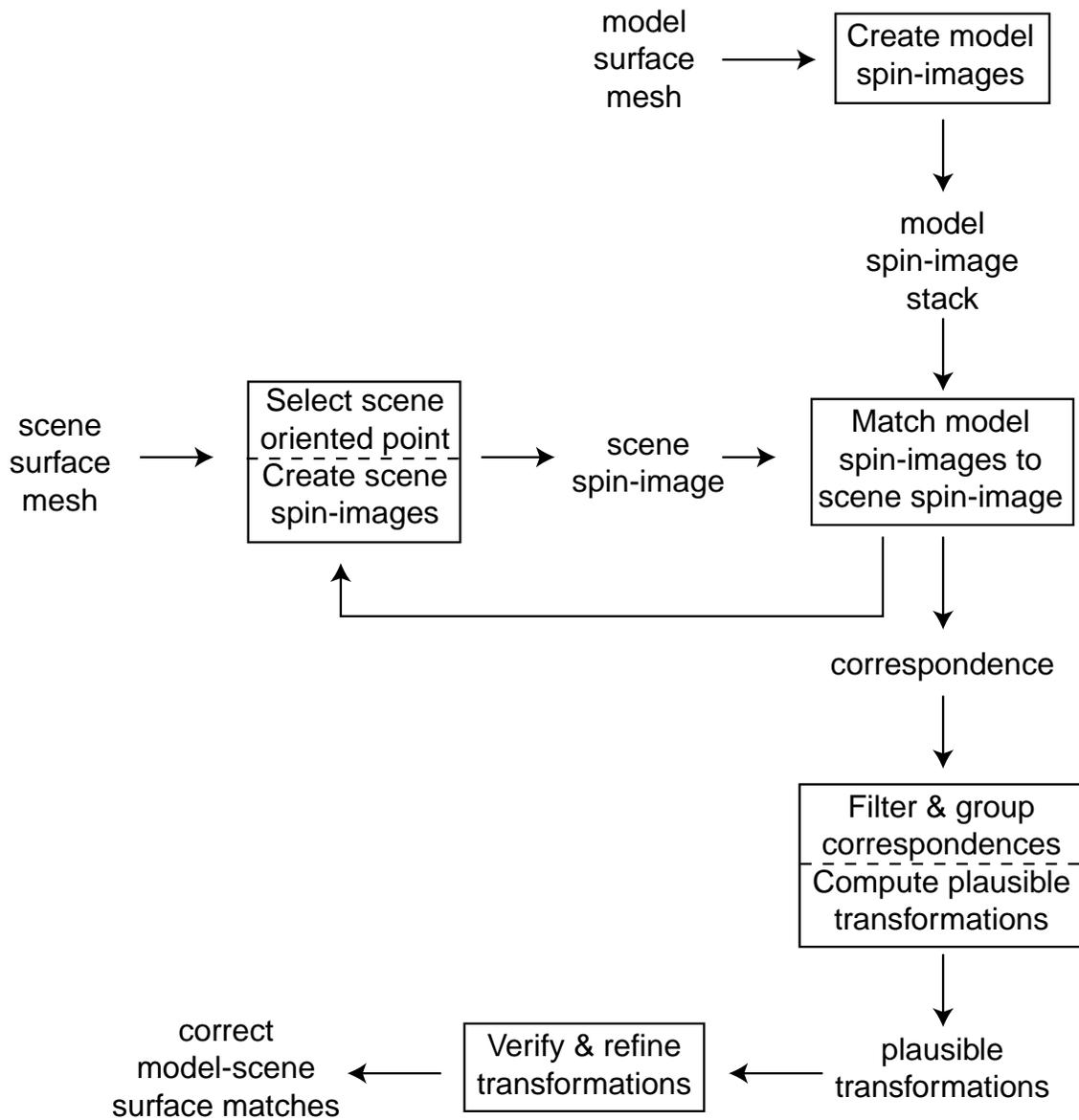
処理の流れをブロック図にすると図.4.4 のようになる.

4.2.1 スピニメージマッチング

あらかじめ知識として持っておくモデルと新しく観測したシーンのスピニメージを比較し, 対応点を得るアルゴリズムを説明する. まず, モデルのスピニメージをオフラインで生成する. 次に得られたシーンの点をランダムに選択してスピニメージを生成し, モデルの全てのスピニメージと比較する. 比較から得られた相関値を用いてヒストグラムを作成し, ヒストグラム中で外れ値となっている対応, すなわち他の対応と比べて非常に相関値が高い対応を選択する. その疑似コードは次のようになる.

Algorithm *SpinImageMatching*

¹ $\operatorname{atanh}(R) = \frac{1}{2} \ln \left(\frac{1+R}{1-R} \right)$



☒ 4.4: Surface matching block diagram

Input: surface mesh S (* Scene *)
Input: surface mesh M (* Model *)
Output: correspondence list L

1. **for** each point $\mathbf{m} \in M$
2. **do** $S_I \leftarrow \text{CreateSpinImage}(\mathbf{m}, M)$
3. $P_S \leftarrow \text{SelectRandomScenePoints}(S)$
4. **for** each point $\mathbf{s} \in P_S$
5. **do** $I_S \leftarrow \text{CreateSpinImage}(\mathbf{s}, S)$
6. **for** each spin-image $I_M \in S_I$
7. **do** $S_C \leftarrow \text{ComputeSimilarityMeasures}(I_S, I_M)$
8. $H \leftarrow \text{CreateSimilarityMeasureHistogram}(S_C)$
9. $O \leftarrow \text{DetectOutliers}(H)$
10. **for** each point $\mathbf{m} \in O$
11. **do** $L \leftarrow \text{CreatePointCorrespondence}(\mathbf{s}, \mathbf{m})$

式(4.6)において比較のパラメータであった λ は比較に用いるモデルのスピンイメージの画素数を N すると、オクリュージョンなどの理由による重なりが小さい場合を考えて、 $\lambda = N/2$ と設定する。ヒストグラムを用いた外れ値の検出は以下のようにする(図.4.5)。まず、相関値の値が大きい方から4分の1の位置にある値 f_u と小さい方から4分の1の位置にある値 f_l の差を f_s とすると、 $f_u + 3f_s$ を外れ値を検出する閾値として用いる。

4.2.2 幾何拘束によるフィルタリングとグループ化

次に、スピンイメージマッチングから計算されたモデルとシーンの対応が幾何的に有効な対応をフィルタリングし、幾何的に整合している対応をグループ化する。2つの対応 $C_1 = [s_1, \mathbf{m}_1]$ 、 $C_2 = [s_2, \mathbf{m}_2]$ が幾何的に整合しているかを測る尺度を式(4.1)を用いて次のように定義する。

$$d_{gc}(C_1, C_2) = \frac{\|S_{m_2}(\mathbf{m}_1) - S_{s_2}(\mathbf{s}_1)\|}{(\|S_{m_2}(\mathbf{m}_1)\| + \|S_{s_2}(\mathbf{s}_1)\|)/2} \quad (4.7)$$

$$D_{gc} = \max(d_{gc}(C_1, C_2), d_{gc}(C_2, C_1)) \quad (4.8)$$

d_{gc} はスピンマップの座標系において正規化された距離を表している。 d_{gc} は対称ではないので $d_{gc}(C_1, C_2)$ と $d_{gc}(C_2, C_1)$ の最大値をとって、幾何整合性を測る距離

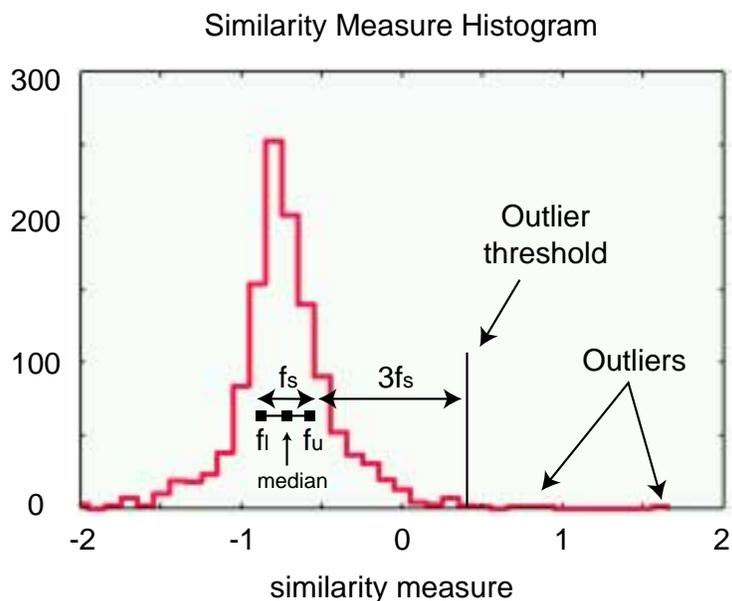


図 4.5: Similarity measure histogram

D_{gc} を定義する．すなわち， D_{gc} が小さければ2つの対応は幾何的に整合性を持つことになる．この D_{gc} を用いたフィルタリングのアルゴリズムは次のようになる．

Algorithm *CorrespondenceFiltering*

Input: correspondence list L

1. **for** each correspondence $C_1 \in L$
2. **do for** other correspondence $C_2 \in L$
3. **do** compute $D_{gc}(C_1, C_2)$
4. $N_{C_1} \leftarrow$ the number of C_2 which $D_{gc}(C_1, C_2) < T_{gc}$
5. **for** each correspondence $C \in L$
6. **do if** $N_C < N_L/4$ **then** remove C from L

ここで N_L は，対応のリスト L の要素数である．また，閾値 T_{gc} は0.25 を用いている．

次に，1つの対応関係からでは位置姿勢のパラメータを求めることはできないので，幾何的に整合性を持つ対応をグループ化を行なう．式 (4.7) を用いて次のようなグループ化のための基準 W_{gc} を定義する．

$$w_{gc}(C_1, C_2) = \frac{d_{gc}(C_1, C_2)}{1 - e^{-(\|S_{m_2}(\mathbf{m}_1)\| + \|S_{s_2}(\mathbf{s}_1)\|)/2\gamma}} \quad (4.9)$$

$$W_{gc} = \max(w_{gc}(C_1, C_2), w_{gc}(C_2, C_1)) \quad (4.10)$$

W_{gc} が小さい時, 2つの対応は整合性を持ち, かつ離れていることを表している. 2つの対応の幾何的な位置が近い場合, ノイズの影響を受けやすく位置姿勢の計算精度が低くなるため [7], 対応間の距離が遠い方が望ましい. 重み γ はスケールに依存しないようにするためのものであり, ここではメッシュの解像度 r の4倍, $\gamma = 4r$ とする. ここでメッシュ解像度とは, 全てのメッシュの辺の長さの中央値と定義する.

対応 C と対応のグループ $\{C_1, \dots, C_n\}$ の間のグループ化の基準は次のようになる.

$$W_{gc}(C, \{C_1, \dots, C_n\}) = \max_i(W_{gc}(C, C_i)) \quad (4.11)$$

これを用いてグループ化のアルゴリズムは次のようになる.

Algorithm *Correspondence Grouping*

Input: correspondence list L

Output: group list G

1. **for** each correspondence $C_i \in L$
2. **do** $G_i \leftarrow \{C_i\}$
3. **for** each correspondence $C_j \in L$
4. **do if** $W_{gc}(C_j, G_i) < T_{gc}$ **then** $G_i \leftarrow C_j$

ここで, 閾値 T_{gc} は 0.25 に設定している.

4.2.3 位置姿勢パラメータの導出

幾何的に整合した対応のグループ化から, モデルとシーンの複数の点について対応が得られたので, 位置姿勢のパラメータを求めることが可能になった. 対応の組 $[s_i, m_i]$ に対して, 次の式で表される残差 E_T を最小化することにより位置姿勢のパラメータからなる変換 T が計算される. 計算の詳細は [43] による. (付録 B 参照)

$$E_T = \sum \|s_i - T(\mathbf{m}_i)\|^2 \quad (4.12)$$

最後にモデル全体とシーン全体を比較することにより, 計算された位置姿勢パラメータの検証と改良を行なう. 検証には Iterative Closest Point Algorithm(ICP 法)

[39, 40, 44] を用いる．ICP 法はモデルとシーンの点の距離を用いた対応点探索とパラメータの改良を繰り返し行なう手法である．したがって局所最小解に陥る可能性があり，モデルとシーンの相対位置が任意の場合には用いることができない．そこで，スピンイメージマッチングから得られた位置姿勢パラメータを初期値として用いて対応点探索を行なう．初期値が良い対応であるならば，モデル全体とシーン全体の対応点探索を行なった結果，対応の組の数が大きく増えることが期待される．したがって，まず大きく対応の組の数が増えた場合には初期値は良いパラメータであり，そうでない場合には悪いパラメータであると判断して検証を行なう．次に，良いパラメータについてICP法を適用してパラメータの改良を行なう．

対応点探索に用いる距離には法線の向きも考慮に入れて，次のような6次元のベクトルによる距離を用いる．

$$d_6 = \sqrt{\|p_1 - p_2\| + \nu \|n_1 - n_2\|} \quad (4.13)$$

p_1, p_2 は点の位置， n_1, n_2 は法線ベクトルである．6次元ベクトルの距離において法線方向は頂点の位置よりも重要であるので，重み係数 $\nu = 2r$ と設定する (r はメッシュ解像度である)． d_6 を用いて $d_6 < 2r$ ならば対応がとれているとする．

対応点探索の効率を上げるために k-D tree[45] を用いる．あらかじめシーンの頂点を用いて k-D tree を構築することによりシーンの各頂点に対して，モデルの頂点の中から d_6 が最小となる頂点を対数オーダの比較回数で見つけることが可能になる．

4.3 階層メッシュモデルを用いたマッチング

本節ではスピンイメージマッチングを第3章で提案した手法を用いて得られたモデルに適用する．得られたモデルの特徴は複数のメッシュ解像度で構築されたモデルを同時に持つことであり，これを利用して認識すべき物体の形状に合わせたメッシュ解像度を選択してマッチングを行なう．

4.3.1 メッシュ解像度の選択

[27] において提案されているスピンイメージマッチングではスピンイメージの生成のためにメッシュの解像度が一定であることが重要である．メッシュの解像度を一定に保つためにメッシュ解像度の調節法 [46, 47] を用いているが，本論文では体

積表現においてメッシュを生成しているためにメッシュの解像度はモデル全体にわたって一定であり，メッシュ解像度をボクセルの幅 W を用いて簡潔にメッシュ解像度 $r = W/2$ とみなす．

メッシュ解像度を自動的に選択することが望ましいが，本論文における実装ではメッシュ解像度の選択を開発者が生成されたメッシュを分析してマッチングに用いる解像度を選択している．図.3.6では，異なった解像度で生成されたモデルを示している．スピンイメージマッチングによって認識を行なう場合，モデル以外の観測対称と区別可能なスピンイメージ生成のためにはある程度頂点数が必要であり，また，頂点数が非常に多い場合にはマッチングにかかるコストが大きくなるので，本論文では解像度 B を用いてスピンイメージマッチングを行なった．

4.3.2 複数のメッシュ解像度における認識アルゴリズム

[42]では複数のモデルについてマッチングを行なう場合，全てのモデルについて同時にスピンイメージマッチングを行ない，それらをまとめてヒストグラムを作成している．それによって複数のモデルにわたった相関値の外れ値を持つ対応が得られ，誤対応が得られる可能性が低くなる．しかし，このような並列に複数のモデルのマッチングを行なうことができるのはメッシュ解像度が等しい場合であり，本論文で用いる異なった解像度で表されたモデルではモデルの解像度を等しくする必要がある．しかし，コンピュータグラフィックスの分野で用いられているメッシュ解像度の調節法 [46, 47] は計算コストが高いため敢えてメッシュ解像度の調節は行わず，本論文では複数の解像度において階層的なマッチングによる認識を行なう．

本論文で提案する階層的なマッチング法では，認識戦略として良く用いられる *coarse to fine* 戦略にしたがって粗い解像度で表されたモデルから，次第に細かい解像度で表されたモデルに順次的にマッチングをとる．粗い解像度でマッチング可能なモデルは，視点位置が離れていてもマッチング可能なモデリングができる観測対象である．逆に細かい解像度で表されたモデルは，シーンの観測対象に接近して細かく観測する必要がある．ロボットを用いて提案する認識手法を適用することを考えると，まず大きな物体からマッチングをとることによってシーンをおおまかに認識し，その後認識から得た知識を利用してシーンの詳細を認識するという行動をとることができる．

また，マッチングのコストを下げるためにできるだけ粗い解像度でマッチングを行ないたい．粗い解像度の場合，特に距離計測誤差の影響などによって生成され

たモデルの形状が異なると法線方向が変化し, スピンイメージマッチングがうまく働かないことが多くなる. 粗い解像度のためにもともと頂点数が少ないので, スピンイメージマッチングを満たす対応が少なくなり, 幾何拘束によるフィルタリングの結果, 全ての対応が除外されてしまう可能性がある. [42] では, まずシーン全体の頂点数の一定の割合 (例えば 10%) をランダムに選択してスピンイメージマッチングを行ない, 幾何拘束によるフィルタリングでは, 幾何的に整合した対応の数 N_C を対応の組全体の数 N_L を用いて $N_C > N_L/4$ となる対応だけを残すように閾値処理している. 本論文では頂点数が少ない場合に適用できるようにこの部分のアルゴリズムを変更し, スピンイメージマッチングとフィルタリングと繰り返し行なう手法を提案する.

まずスピンイメージマッチングを行ない, 対応が適当な数 T_N だけ得られるまでシーンの頂点についてマッチングを行なう. 得られた対応についてフィルタリングし, 定数の閾値 T_C を用いて $N_C > T_C$ となる対応のリストの要素数 L_N を数える. 閾値 T_L を用いて $L_N > T_N$ となるまで再びスピンイメージマッチングとフィルタリングを繰り返す. 全てのシーンの頂点についてマッチングを行なっても $L_N > T_N$ とならない場合, N_C の大きい対応から T_N 個の対応を選択する.

複数のメッシュ解像度による階層的なマッチングアルゴリズムは, スピンイメージマッチングとフィルタリングを繰り返すアルゴリズム *MatchingAndFiltering* を用いて次のようになる.

Algorithm *HierarchicalMatching*

Input: surface mesh S

Input: model list L

1. **for** each model mesh $M \in L$
2. **do for** each verified correspondence list $G_i \in G$
3. **do** remove vertices from S which are used in G_i
4. $L \leftarrow \text{MatchingAndFiltering}(S, M)$
5. $G \leftarrow \text{CorrespondenceGrouping}(L)$
6. $\text{VerifyCorrespondence}(G)$

4.4 実験

第3章と同様に本章においても, 仮想環境を用いた実験を行なう.

図.4.6 に示す複数のモデルを図.3.11 で示したシーンをモデリングした結果 (図.3.15) に対してマッチングを行なった。マッチングのためのパラメータは次のように設定した。

- ビンサイズ $b = W/2$, W はボクセルサイズ。
- スピンイメージサイズ $W_i = 20$ 。
- サポートアングル $A_s = 180^\circ$ 。
- 繰り返しの閾値 $T_N = 100, T_C = 30$ 。

マッチング時に用いたボクセルが大きい方から順に、テーブル、だるま、立方体、円錐、トリケラトプスについてマッチングをとった。階層的にマッチングし、対応がとれたシーンの頂点を取り除いていく過程を図.4.8 に示し、最終的なマッチング結果を図.4.7 に示す。シーンのメッシュを面を用いて表し、モデルのメッシュをエッジのみを用いて表している。

対応がとれたシーンの頂点を取り除くことにより、スピンイメージマッチングにおける相関値 C が改善することを図.4.9 に示す。図.4.9 では、だるまの表面上の点についてシーンそのままの場合 (A) とシーンからテーブルとマッチングした部分の頂点を取り除いた場合 (B) についてそれぞれ相関値を計算したものである。(B) ではスピンイメージに含まれるだるま以外の部分が減少した結果、モデルのスピンイメージとの相関値が大きくなっている。物体が混在するシーンにおいて既にマッチングした結果を用いることにより、スピンイメージに含まれる他の物体の影響を小さくできることが示された。

全てのモデルのマッチングにかかった時間は 214sec であった。マッチングの各ステップに要した時間を表.4.1 に、各ステップで処理した頂点数と対応の数を表.4.2 に示す。*MatchingAndFiltering* の時間はモデルの頂点数とシーンの頂点数に大きく影響されている。同様に、*VerifyCorrespondence* では対応のグループの数が多い場合、その検証に大きな計算時間がかかっている。また、シーンの頂点の除去における対応点探索が全体の計算時間の大きな割合を占めている。したがって、今後の課題として次の2点が挙げられる。第一にフィルタリングとグループ化の結果を調節し、実時間性を持つ、すなわち処理の制限時間が決められているアルゴリズムを開発する。第二に、k-D tree[45] を用いた探索を実装し、シーンの頂点の除去の計算時間の短縮をはかる。

表 4.1: Times in each step of matching (msec)

	Remove vertices	<i>MatchingAnd Filtering</i>	<i>Correspondence Grouping</i>	<i>Verify Correspondence</i>
Table	0	7416	29	3411
Daruma	9509	28893	65	15666
Cube	24033	2293	1	271
Cone	25748	2649	4	1400
Triceratops	24553	34590	126	33948

表 4.2: Number of vertices and correspondences

	Model vertices	Original scene vertices	Remaining scene vertices	Compared scene vertices	Correspondences after filtering	Groups of correspondences
Table	267	727	727	727	100	26
Daruma	735	3457	1932	1932	100	39
Cube	150	3457	1193	1193	7	6
Cone	259	3457	1029	1029	30	17
Triceratops	1233	1438	1130	1130	95	65

また、現在の実装においてはICP法 [39, 40, 44] による位置姿勢の改良が実装されていないため、マッチング結果にずれが存在している。ICP法を実装し位置姿勢のパラメータを改善することも今後の課題である。

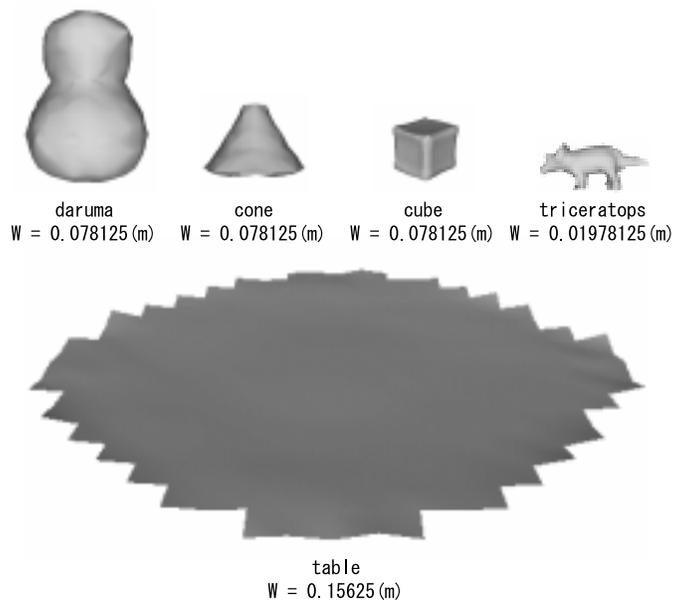


図 4.6: Matching models

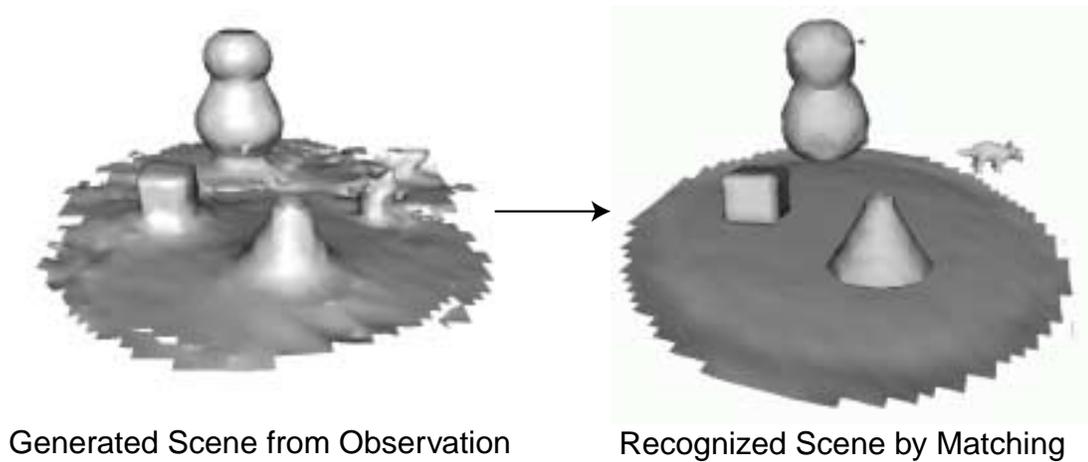


図 4.7: Final matching result

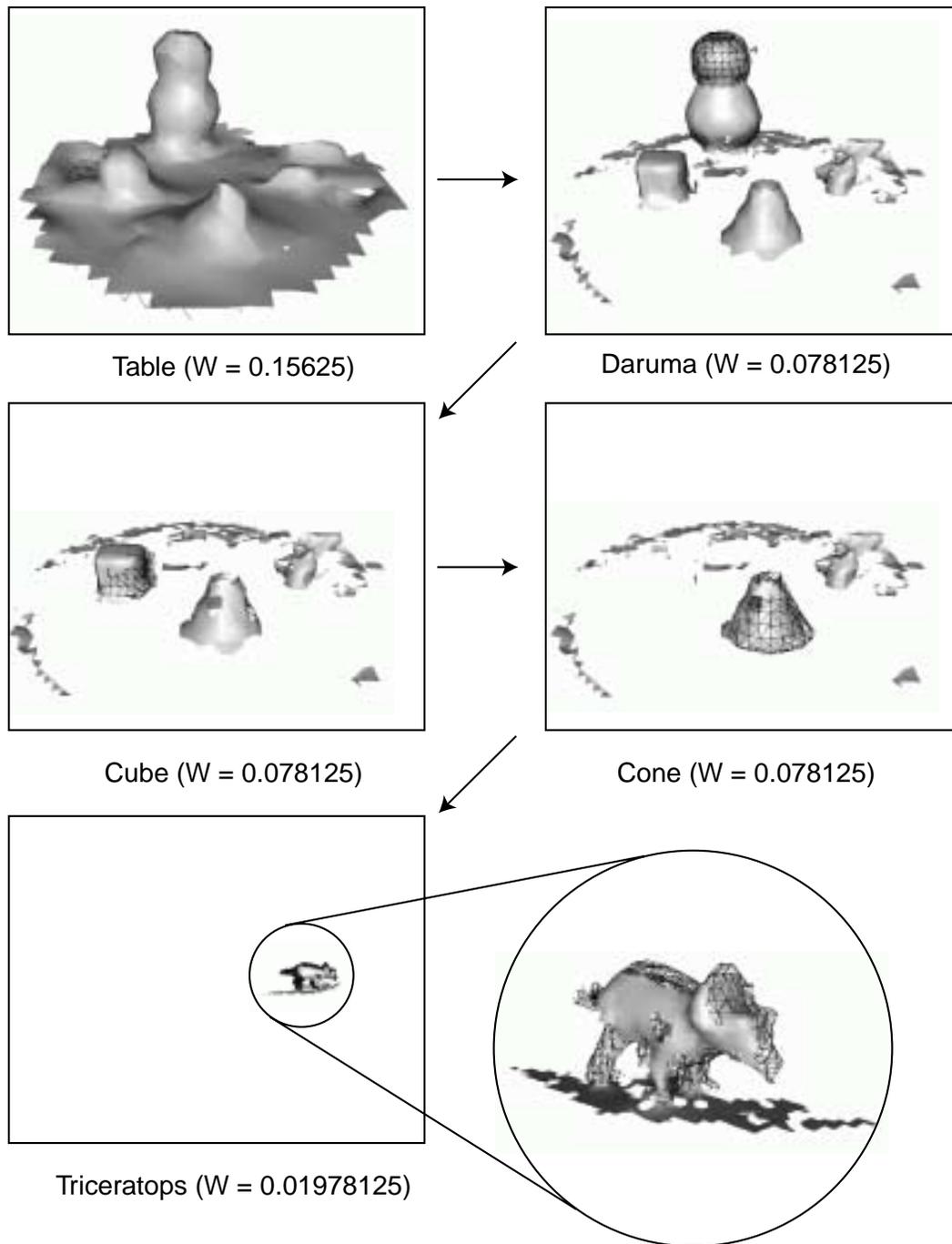
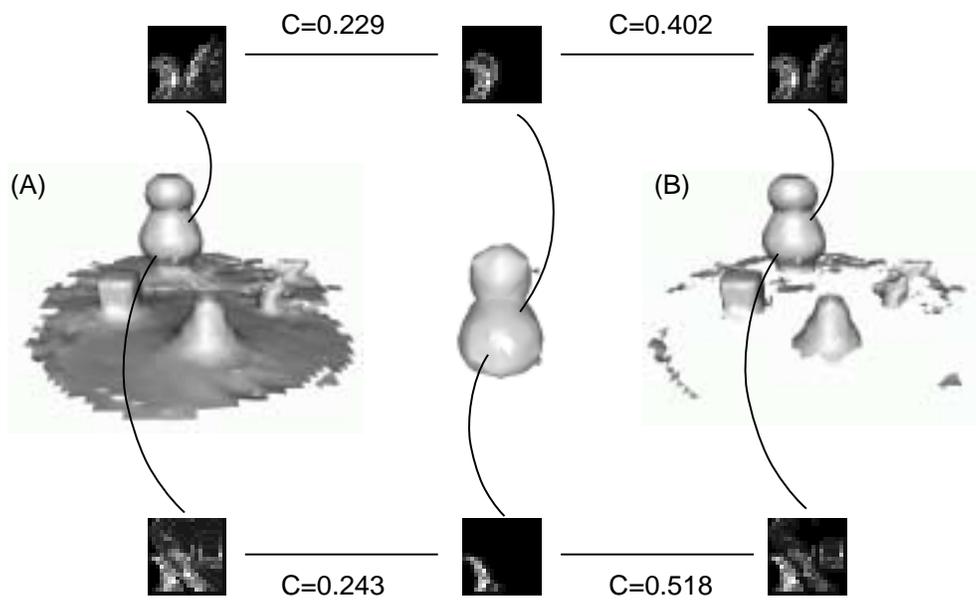


図 4.8: Hierarchical matching process with removing matched vertices



☒ 4.9: Improvement of correlation by removing matched vertices

第 5 章

結論

本論文ではロボットのための物体形状観測認識システムを提案した。そのシステムは、ステレオ視による物体形状の観測、複数視点からの観測結果を用いた物体形状のモデリング、形状モデルのマッチングによる物体認識の3つの段階から構成される。3つの段階それぞれについて結論を述べ、最後にシステム全体の結論を述べる。

5.1 ステレオ視による距離画像生成

まず3次元空間から2次元カメラ画像への変換を知るためにカメラモデルについて考察した。第一に、ピンホールカメラモデルを適用するため、カメラ画像のレンズによる歪みを補正した。第二に、ピンホールカメラモデルにおけるカメラパラメータを求め、それらを用いて3次元空間から2次元カメラ画像への変換を定義した。第三に、ステレオ視の幾何的な意味について述べ、量子化されたカメラ画像を用いたステレオ視の計測精度を考察した。

ステレオ視のアルゴリズムについて画像処理の分野で行われている再帰的関連演算手法を用いて、ステレオ画像の高速な対応点探索を行うアルゴリズムを、キャッシュを有効に利用する形に最適化し、Pentium プロセッサのマルチメディア命令 (MMX 命令) を利用することにより、市販のPC部品のみを組み合わせで実時間で視差画像を得ることが可能であることを示した。本論文で示した視差画像生成システムは一般的かつ強力であり、当研究室で開発している車輪移動上半身型ヒューマノイド H4 [48]、車輪移動型ロボット HyperMouse [49] など、様々なロボットで利用されている。

また MMX 命令では命令セットとデータ長の制約から正規化相関を実現できなかったが、PentiumIII に搭載された次期マルチメディア命令 SSE ではこの問題もクリアされるため、正規化相関も実時間になることを確認している。

実世界を行動するロボットでは視差画像と通常の視覚を併せて得られるステレオ画像システムは非常に有用であることから、今後は視差画像と通常の視覚の統合による物体の認識や3次元デプスマップといった研究が可能になってくる。また再帰的関連演算の手法を2次元に拡張することによりマトリクスになったオプティカルフローが高速に計算できる [50]。

また、複数のカメラを用いて距離計測を行なうことにより、2枚の画像を用いたステレオ視と比べて、マッチングに用いるウインドウのサイズを小さくしても対応点探索が可能である、狭いベースラインからの観測でも、サブピクセル精度の距離

計測が可能である，という利点が得られたことを確認した．

5.2 3次元表面形状モデリング

複数視点から得られた距離画像を用いて3次元メッシュモデルを生成した．複数視点からの距離画像を統合することにより，単一視点からでは観測できない形状の物体をモデリング可能になった．統合において体積表現 (volumetric representation) を用いることにより，観測視点に関わらず平均的なメッシュ密度のモデルを生成することが可能である．

従来提案されている Marching Cubes Algorithm と符号付き距離法を用いて3次元モデリングする手法をロボット用視覚として用いるために以下に述べるアルゴリズムの変更を行ない，高速インクリメンタルモデリングを可能にした．第一に，octree を用いて階層的に大きさの異なるボクセルを表現し，メッシュ密度の異なるモデルを同時にモデリングすることを可能にした．第二に，符号付き距離を計算するボクセルを距離画像面を利用して限定することにより，計算量を減らし高速モデリングが可能になった．第三に，ステレオ視の距離計測精度を考察することによりモデリングに用いるボクセルの大きさを調節し，計算時間とモデル解像度のトレードオフする基準を提案した．

5.3 3次元物体認識

記憶しているモデルと新たに観測したシーンをメッシュモデルで表し，その要素の頂点間で対応点探索によりモデルのマッチングを行なった．頂点を特徴を表す表現としてスピンイメージを用いてマッチングすることにより次のような利点が得られる．第一に，物体が混在する場合にもロバストにマッチング可能である．第二に，部分的なモデルに対してもマッチングが適用できる．第三に，マッチングする対象の形状に仮定をおかない．第四に，ノイズに敏感な特徴抽出を行なう必要がない．

本論文で提案したモデル生成法を用いて得られた階層メッシュモデルに対してスピンイメージマッチングを適用し，モデルに適応したモデル解像度を選択してマッチングを行なった．これによって，モデルの全体的な大きさに関わらずマッチングが適用可能になった．また，粗い解像度のモデルから細かい解像度のモデルに coarse to fine 戦略に基づいて順序的にマッチングを行なった，これによって，観測しているシーンに対してマッチングすべき対象が占める割合が小さい場合にも効

率的なマッチングが可能になった。

5.4 3次元観測認識システム

ステレオ視による距離画像生成，複数視点からの距離画像を用いたモデリング，生成された階層モデルを用いたマッチングをまとめて，本論文では視覚を用いて物体の形状をモデリングし，生成したモデルを用いて物体の認識を行なうシステムを提案した．提案したシステムの特徴は次のようになる．第一に，あらかじめ記憶するモデルと，認識するシーンのモデリング方法が同一である．すなわち，開発者がモデルそのものを与える必要がない．第二に，対象の形状を仮定しない．ステレオ視，モデリング，マッチングのそれぞれにおいて本論文が提案する手法は，観測対象の形状について仮定をおく必要がない．第三に，観測認識する対象に適応的である．すなわち，モデリングにおいては観測対象までの距離とカメラパラメータに従って，モデリングの解像度を調節し，マッチングにおいては観測対象に適したモデルの解像度を用いて認識を行なう．

今後の課題は，まず，このシステムをロボットに実装しロボットの行動と結び付けることである．このシステムを用いて探索と認識を繰り返すことにより，ロボットを取り巻く環境を探索する行動が可能になる．観測結果の不確実な部分を注視し観測すべき部分をプランニングする研究 [51, 52, 53] と同様にモデリング結果を利用することができる．また，本論文においてはマッチングに用いるモデルの獲得はオフラインで行なったが，単体のロボットにおいてオンラインでモデリングから認識まで行なわせることも可能になる．

また，本論文で扱ったのは物体の形状だけであったが，形状を観測対象の持っている特徴の一つに過ぎない．ロバストな認識を行なうためには様々な特徴を抽出し，観測対象に適した特徴を選ぶことが重要である．様々な特徴についての認識を加え，適応的な認識を行なうシステムとする．

本論文では形状についてボトムアップなモデリングとトップダウンな認識を提案した．様々な特徴について，このモデリングと認識のサイクルを確立し，環境に適応的な行動が可能なロボットを目指す．

謝辭

本研究は東京大学機械情報工学科井上博允教授，稲葉雅幸助教授の御指導のもとで行なわれました。

井上教授には，研究に対して鋭い指摘や意見をいただきとても勉強になりました。また，研究以外の社会性など人間的な面でもご指導頂きました。

稲葉助教授には，大学院から研究室にはいった筆者を暖かく迎えていただき，大変感謝しております。

また藤田技官，戸塚技官，千代延秘書，大澤秘書には研究を進めるための様々な事務処理をして頂いたおかげで，順調に研究を進めることができました。

リサーチアソシエイツの加賀美さんには，研究活動の様々な面において大変お世話になり，筆者の拙い疑問，質問に丁寧に答えていただきました。本来なら筆者が取り組むべき実験の下準備など，スムーズな研究が出来る環境を作っていただき，大変感謝しております。

先輩である，PDの金広さん，長島さん，D3の稲邑さん，長坂さん，D2の水内さん，香山さん，星野さん，D1の陰山さん，西脇さん，岡田さんには研究室でお世話になりました。

稲邑さんには，学会発表前の原稿校正や実験装置に関する疑問等でお手数をお掛け致しました。また，稲邑さんの研究に取り組む姿勢を身近にみることができ，是非参考にしたいと考えています。

岡田さんには，日頃から一緒に議論して頂き，方針の決定に大変参考になりました。また，実験では直接手伝って頂いたり，プログラム等を利用させて頂くなど，とても感謝しております。

同学年である，大武さん，垣内君，桜沢さん，宅間君，二井君，山本さん，北川君とは，2年間一緒に研究ができ大変感謝しております。

M1の福島君，杉原君，中君，服部君，中井君，川島君，原君，伊藤君は，それぞれ豊かな発想で研究を進め，筆者が教えられることも多かったです。

研究につまった時，研究室の多くの皆さんに話しかけ，おしゃべりしてよく邪魔をしたのにはとても御迷惑をおかけしました。

また，研究室を離れたところで，筆者を暖かく見守っていただき，すばらしい発想を披露してくれた，先輩達，友人達に感謝します。

最後に筆者を支えてくれた両親に感謝します。

付録 A

Marching Cubes Algorithm

Marching Cubes Algorithm の C ソースコードを以下に示す。このソースコードは Paul Bourke , Cory Gene Bloyd によるものである。¹

```
typedef struct {
    XYZ p[3];
} TRIANGLE;

typedef struct {
    XYZ p[8];
    double val[8];
} GRIDCELL;

/*
    Given a grid cell and an isolevel, calculate the triangular
    facets required to represent the isosurface through the cell.
    Return the number of triangular facets, the array "triangles"
    will be loaded up with the vertices at most 5 triangular facets.
    0 will be returned if the grid cell is either totally above
    or totally below the isolevel.
*/
int Polygonise(grid, isolevel, triangles)
GRIDCELL grid;
double isolevel;
TRIANGLE *triangles;
{
    int i, ntriang;
    int cubeindex;
    XYZ vertlist[12];

    int edgeTable[256]={
0x0 , 0x109, 0x203, 0x30a, 0x406, 0x50f, 0x605, 0x70c,
0x80c, 0x905, 0xa0f, 0xb06, 0xc0a, 0xd03, 0xe09, 0xf00,
0x190, 0x99 , 0x393, 0x29a, 0x596, 0x49f, 0x795, 0x69c,
0x99c, 0x895, 0xb9f, 0xa96, 0xd9a, 0xc93, 0xf99, 0xe90,
0x230, 0xf39, 0x33 , 0x13a, 0x636, 0x73f, 0x435, 0x53c,
0xa3c, 0xb35, 0x83f, 0x936, 0xe3a, 0xf33, 0xc39, 0xd30,
0x3a0, 0x2a9, 0x1a3, 0xaa , 0x7a6, 0x6af, 0x5a5, 0x4ac,
0xbac, 0xaa5, 0x9af, 0x8a6, 0xfaa, 0xea3, 0xda9, 0xca0,
0x460, 0x569, 0x663, 0x76a, 0x66 , 0x16f, 0x265, 0x36c,
0xc6c, 0xd65, 0xe6f, 0xf66, 0x86a, 0x963, 0xa69, 0xb60,
0x5f0, 0x4f9, 0x7f3, 0x6fa, 0x1f6, 0xff , 0x3f5, 0x2fc,
0xdc, 0xcf5, 0xffff, 0xef6, 0x9fa, 0x8f3, 0xbf9, 0xaf0,
0x650, 0x759, 0x453, 0x55a, 0x256, 0x35f, 0x55 , 0x15c,
0xe5c, 0xf55, 0xc5f, 0xd56, 0xa5a, 0xb53, 0x859, 0x950,
0x7c0, 0x6c9, 0x5c3, 0x4ca, 0x3c6, 0x2cf, 0x1c5, 0xcc ,
0xfcc, 0xec5, 0xdcf, 0xcc6, 0xbca, 0xac3, 0x9c9, 0x8c0,
0x8c0, 0x9c9, 0xac3, 0xbca, 0xcc6, 0xdcf, 0xec5, 0xfcc,
0xcc , 0x1c5, 0x2cf, 0x3c6, 0x4ca, 0x5c3, 0x6c9, 0x7c0,
0x950, 0x859, 0xb53, 0xa5a, 0xd56, 0xc5f, 0xf55, 0xe5c,
0x15c, 0x55 , 0x35f, 0x256, 0x55a, 0x453, 0x759, 0x650,
0xaf0, 0xbf9, 0x8f3, 0x9fa, 0xef6, 0xffff, 0xcf5, 0xdfc,
0x2fc, 0x3f5, 0xff , 0x1f6, 0x6fa, 0x7f3, 0x4f9, 0x5f0,
0xb60, 0xa69, 0x963, 0x86a, 0xf66, 0xe6f, 0xd65, 0xc6c,
0x36c, 0x265, 0x16f, 0x66 , 0x76a, 0x663, 0x569, 0x460,
0xca0, 0xda9, 0xea3, 0xfaa, 0x8a6, 0x9af, 0xaa5, 0xbac,
0x4ac, 0x5a5, 0x6af, 0x7a6, 0xaa , 0x1a3, 0x2a9, 0x3a0,
0xd30, 0xc39, 0xf33, 0xe3a, 0x936, 0x83f, 0xb35, 0xa3c,
0x53c, 0x435, 0x73f, 0x636, 0x13a, 0x33 , 0x339, 0x230,
0xe90, 0xf99, 0xc93, 0xd9a, 0xa96, 0xb9f, 0x895, 0x99c,
0x69c, 0x795, 0x49f, 0x596, 0x29a, 0x393, 0x99 , 0x190,
0xf00, 0xe09, 0xd03, 0xc0a, 0xb06, 0xa0f, 0x905, 0x80c,
0x70c, 0x605, 0x50f, 0x406, 0x30a, 0x203, 0x109, 0x0 };
    int triTable[256][16] = {
    { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
    { 0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
    { 0, 1, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
    { 1, 8, 3, 9, 8, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
    { 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
    { 0, 8, 3, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
    { 9, 2, 10, 0, 2, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
```

¹<http://www.swin.edu.au/astronomy/pbourke/modelling/polygonise/>

```

{ 2, 8, 3, 2, 10, 8, 10, 9, 8, -1, -1, -1, -1, -1, -1, -1},
{ 3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 11, 2, 8, 11, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 9, 0, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 11, 2, 1, 9, 11, 9, 8, 11, -1, -1, -1, -1, -1, -1, -1},
{ 3, 10, 1, 11, 10, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 10, 1, 0, 8, 10, 8, 11, 10, -1, -1, -1, -1, -1, -1, -1},
{ 3, 9, 0, 3, 11, 9, 11, 10, 9, -1, -1, -1, -1, -1, -1, -1},
{ 9, 8, 10, 10, 8, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 3, 0, 7, 3, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 1, 9, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 1, 9, 4, 7, 1, 7, 3, 1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 2, 10, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 3, 4, 7, 3, 0, 4, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1},
{ 9, 2, 10, 9, 0, 2, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1},
{ 2, 10, 9, 2, 9, 7, 2, 7, 3, 7, 9, 4, -1, -1, -1, -1},
{ 8, 4, 7, 3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 11, 4, 7, 11, 2, 4, 2, 0, 4, -1, -1, -1, -1, -1, -1, -1},
{ 9, 0, 1, 8, 4, 7, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1},
{ 4, 7, 11, 9, 4, 11, 9, 11, 2, 9, 2, 1, -1, -1, -1, -1},
{ 3, 10, 1, 3, 11, 10, 7, 8, 4, -1, -1, -1, -1, -1, -1, -1},
{ 1, 11, 10, 1, 4, 11, 1, 0, 4, 7, 11, 4, -1, -1, -1, -1},
{ 4, 7, 8, 9, 0, 11, 9, 11, 10, 11, 0, 3, -1, -1, -1, -1},
{ 4, 7, 11, 4, 11, 9, 9, 11, 10, -1, -1, -1, -1, -1, -1, -1},
{ 9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 9, 5, 4, 0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 5, 4, 1, 5, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 8, 5, 4, 8, 3, 5, 3, 1, 5, -1, -1, -1, -1, -1, -1, -1},
{ 1, 2, 10, 9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 3, 0, 8, 1, 2, 10, 4, 9, 5, -1, -1, -1, -1, -1, -1, -1},
{ 5, 2, 10, 5, 4, 2, 4, 0, 2, -1, -1, -1, -1, -1, -1, -1},
{ 2, 10, 5, 3, 2, 5, 3, 5, 4, 3, 4, 8, -1, -1, -1, -1},
{ 9, 5, 4, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 11, 2, 0, 8, 11, 4, 9, 5, -1, -1, -1, -1, -1, -1, -1},
{ 0, 5, 4, 0, 1, 5, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1},
{ 2, 1, 5, 2, 5, 8, 2, 8, 11, 4, 8, 5, -1, -1, -1, -1},
{ 10, 3, 11, 10, 1, 3, 9, 5, 4, -1, -1, -1, -1, -1, -1, -1},
{ 4, 9, 5, 0, 8, 1, 8, 10, 1, 8, 11, 10, -1, -1, -1, -1},
{ 5, 4, 0, 5, 0, 11, 5, 11, 10, 11, 0, 3, -1, -1, -1, -1},
{ 5, 4, 8, 5, 8, 10, 10, 8, 11, -1, -1, -1, -1, -1, -1, -1},
{ 9, 7, 8, 5, 7, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 9, 3, 0, 9, 5, 3, 5, 7, 3, -1, -1, -1, -1, -1, -1, -1},
{ 0, 7, 8, 0, 1, 7, 1, 5, 7, -1, -1, -1, -1, -1, -1, -1},
{ 1, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 9, 7, 8, 9, 5, 7, 10, 1, 2, -1, -1, -1, -1, -1, -1, -1},
{ 10, 1, 2, 9, 5, 0, 5, 3, 0, 5, 7, 3, -1, -1, -1, -1},
{ 8, 0, 2, 8, 2, 5, 8, 5, 7, 10, 5, 2, -1, -1, -1, -1},
{ 2, 10, 5, 2, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1, -1},
{ 7, 9, 5, 7, 8, 9, 3, 11, 2, -1, -1, -1, -1, -1, -1, -1},
{ 9, 5, 7, 9, 7, 2, 9, 2, 0, 2, 7, 11, -1, -1, -1, -1},
{ 2, 3, 11, 0, 1, 8, 1, 7, 8, 1, 5, 7, -1, -1, -1, -1},
{ 11, 2, 1, 11, 1, 7, 7, 1, 5, -1, -1, -1, -1, -1, -1, -1},
{ 9, 5, 8, 8, 5, 7, 10, 1, 3, 10, 3, 11, -1, -1, -1, -1},
{ 5, 7, 0, 5, 0, 9, 7, 11, 0, 1, 0, 10, 11, 10, 0, -1},
{ 11, 10, 0, 11, 0, 3, 10, 5, 0, 8, 0, 7, 5, 7, 0, -1},
{ 11, 10, 5, 7, 11, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 10, 6, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 8, 3, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 9, 0, 1, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 8, 3, 1, 9, 8, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1},
{ 1, 6, 5, 2, 6, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 6, 5, 1, 2, 6, 3, 0, 8, -1, -1, -1, -1, -1, -1, -1},
{ 9, 6, 5, 9, 0, 6, 0, 2, 6, -1, -1, -1, -1, -1, -1, -1},
{ 5, 9, 8, 5, 8, 2, 5, 2, 6, 3, 2, 8, -1, -1, -1, -1},
{ 2, 3, 11, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 11, 0, 8, 11, 2, 0, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1},
{ 0, 1, 9, 2, 3, 11, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1},
{ 5, 10, 6, 1, 9, 2, 9, 11, 2, 9, 8, 11, -1, -1, -1, -1},
{ 6, 3, 11, 6, 5, 3, 5, 1, 3, -1, -1, -1, -1, -1, -1, -1},
{ 0, 8, 11, 0, 11, 5, 0, 5, 1, 5, 11, 6, -1, -1, -1, -1},
{ 3, 11, 6, 0, 3, 6, 0, 6, 5, 0, 5, 9, -1, -1, -1, -1},
{ 6, 5, 9, 6, 9, 11, 11, 9, 8, -1, -1, -1, -1, -1, -1, -1},
{ 5, 10, 6, 4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 3, 0, 4, 7, 3, 6, 5, 10, -1, -1, -1, -1, -1, -1, -1},
{ 1, 9, 0, 5, 10, 6, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1},

```

```

{ 10, 6, 5, 1, 9, 7, 1, 7, 3, 7, 9, 4, -1, -1, -1, -1},
{ 6, 1, 2, 6, 5, 1, 4, 7, 8, -1, -1, -1, -1, -1, -1},
{ 1, 2, 5, 5, 2, 6, 3, 0, 4, 3, 4, 7, -1, -1, -1, -1},
{ 8, 4, 7, 9, 0, 5, 0, 6, 5, 0, 2, 6, -1, -1, -1, -1},
{ 7, 3, 9, 7, 9, 4, 3, 2, 9, 5, 9, 6, 2, 6, 9, -1},
{ 3, 11, 2, 7, 8, 4, 10, 6, 5, -1, -1, -1, -1, -1, -1},
{ 5, 10, 6, 4, 7, 2, 4, 2, 0, 2, 7, 11, -1, -1, -1, -1},
{ 0, 1, 9, 4, 7, 8, 2, 3, 11, 5, 10, 6, -1, -1, -1, -1},
{ 9, 2, 1, 9, 11, 2, 9, 4, 11, 7, 11, 4, 5, 10, 6, -1},
{ 8, 4, 7, 3, 11, 5, 3, 5, 1, 5, 11, 6, -1, -1, -1, -1},
{ 5, 1, 11, 5, 11, 6, 1, 0, 11, 7, 11, 4, 0, 4, 11, -1},
{ 0, 5, 9, 0, 6, 5, 0, 3, 6, 11, 6, 3, 8, 4, 7, -1},
{ 6, 5, 9, 6, 9, 11, 4, 7, 9, 7, 11, 9, -1, -1, -1, -1},
{ 10, 4, 9, 6, 4, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 10, 6, 4, 9, 10, 0, 8, 3, -1, -1, -1, -1, -1, -1},
{ 10, 0, 1, 10, 6, 0, 6, 4, 0, -1, -1, -1, -1, -1, -1},
{ 8, 3, 1, 8, 1, 6, 8, 6, 4, 6, 1, 10, -1, -1, -1, -1},
{ 1, 4, 9, 1, 2, 4, 2, 6, 4, -1, -1, -1, -1, -1, -1},
{ 3, 0, 8, 1, 2, 9, 2, 4, 9, 2, 6, 4, -1, -1, -1, -1},
{ 0, 2, 4, 4, 2, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 8, 3, 2, 8, 2, 4, 2, 6, -1, -1, -1, -1, -1, -1, -1},
{ 10, 4, 9, 10, 6, 4, 11, 2, 3, -1, -1, -1, -1, -1, -1},
{ 0, 8, 2, 2, 8, 11, 4, 9, 10, 4, 10, 6, -1, -1, -1, -1},
{ 3, 11, 2, 0, 1, 6, 0, 6, 4, 6, 1, 10, -1, -1, -1, -1},
{ 6, 4, 1, 6, 1, 10, 4, 8, 1, 2, 1, 11, 8, 11, 1, -1},
{ 9, 6, 4, 9, 3, 6, 9, 1, 3, 11, 6, 3, -1, -1, -1, -1},
{ 8, 11, 1, 8, 1, 0, 11, 6, 1, 9, 1, 4, 6, 4, 1, -1},
{ 3, 11, 6, 3, 6, 0, 0, 6, 4, -1, -1, -1, -1, -1, -1, -1},
{ 6, 4, 8, 11, 6, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 7, 10, 6, 7, 8, 10, 8, 9, 10, -1, -1, -1, -1, -1, -1, -1},
{ 0, 7, 3, 0, 10, 7, 0, 9, 10, 6, 7, 10, -1, -1, -1, -1},
{ 10, 6, 7, 1, 10, 7, 1, 7, 8, 1, 8, 0, -1, -1, -1, -1},
{ 10, 6, 7, 10, 7, 1, 1, 7, 3, -1, -1, -1, -1, -1, -1, -1},
{ 1, 2, 6, 1, 6, 8, 1, 8, 9, 8, 6, 7, -1, -1, -1, -1},
{ 2, 6, 9, 2, 9, 1, 6, 7, 9, 0, 9, 3, 7, 3, 9, -1},
{ 7, 8, 0, 7, 0, 6, 6, 0, 2, -1, -1, -1, -1, -1, -1, -1},
{ 7, 3, 2, 6, 7, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 2, 3, 11, 10, 6, 8, 10, 8, 9, 8, 6, 7, -1, -1, -1, -1},
{ 2, 0, 7, 2, 7, 11, 0, 9, 7, 6, 7, 10, 9, 10, 7, -1},
{ 1, 8, 0, 1, 7, 8, 1, 10, 7, 6, 7, 10, 2, 3, 11, -1},
{ 11, 2, 1, 11, 1, 7, 10, 6, 1, 6, 7, 1, -1, -1, -1, -1},
{ 8, 9, 6, 8, 6, 7, 9, 1, 6, 11, 6, 3, 1, 3, 6, -1},
{ 0, 9, 1, 11, 6, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 7, 8, 0, 7, 0, 6, 3, 11, 0, 11, 6, 0, -1, -1, -1, -1},
{ 7, 11, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 7, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 3, 0, 8, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 1, 9, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 8, 1, 9, 8, 3, 1, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1},
{ 10, 1, 2, 6, 11, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 2, 10, 3, 0, 8, 6, 11, 7, -1, -1, -1, -1, -1, -1, -1},
{ 2, 9, 0, 2, 10, 9, 6, 11, 7, -1, -1, -1, -1, -1, -1, -1},
{ 6, 11, 7, 2, 10, 3, 10, 8, 3, 10, 9, 8, -1, -1, -1, -1},
{ 7, 2, 3, 6, 2, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 7, 0, 8, 7, 6, 0, 6, 2, 0, -1, -1, -1, -1, -1, -1, -1},
{ 2, 7, 6, 2, 3, 7, 0, 1, 9, -1, -1, -1, -1, -1, -1, -1},
{ 1, 6, 2, 1, 8, 6, 1, 9, 8, 8, 7, 6, -1, -1, -1, -1},
{ 10, 7, 6, 10, 1, 7, 1, 3, 7, -1, -1, -1, -1, -1, -1, -1},
{ 10, 7, 6, 1, 7, 10, 1, 8, 7, 1, 0, 8, -1, -1, -1, -1},
{ 0, 3, 7, 0, 7, 10, 0, 10, 9, 6, 10, 7, -1, -1, -1, -1},
{ 7, 6, 10, 7, 10, 8, 8, 10, 9, -1, -1, -1, -1, -1, -1, -1},
{ 6, 8, 4, 11, 8, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 3, 6, 11, 3, 0, 6, 0, 4, 6, -1, -1, -1, -1, -1, -1, -1},
{ 8, 6, 11, 8, 4, 6, 9, 0, 1, -1, -1, -1, -1, -1, -1, -1},
{ 9, 4, 6, 9, 6, 3, 9, 3, 1, 11, 3, 6, -1, -1, -1, -1},
{ 6, 8, 4, 6, 11, 8, 2, 10, 1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 2, 10, 3, 0, 11, 0, 6, 11, 0, 4, 6, -1, -1, -1, -1},
{ 4, 11, 8, 4, 6, 11, 0, 2, 9, 2, 10, 9, -1, -1, -1, -1},
{ 10, 9, 3, 10, 3, 2, 9, 4, 3, 11, 3, 6, 4, 6, 3, -1},
{ 8, 2, 3, 8, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1, -1},
{ 0, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 9, 0, 2, 3, 4, 2, 4, 6, 4, 3, 8, -1, -1, -1, -1},
{ 1, 9, 4, 1, 4, 2, 2, 4, 6, -1, -1, -1, -1, -1, -1, -1},
{ 8, 1, 3, 8, 6, 1, 8, 4, 6, 6, 10, 1, -1, -1, -1, -1},
{ 10, 1, 0, 10, 0, 6, 6, 0, 4, -1, -1, -1, -1, -1, -1, -1},
{ 4, 6, 3, 4, 3, 8, 6, 10, 3, 0, 3, 9, 10, 9, 3, -1},

```

```

{ 10, 9, 4, 6, 10, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 9, 5, 7, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 8, 3, 4, 9, 5, 11, 7, 6, -1, -1, -1, -1, -1, -1},
{ 5, 0, 1, 5, 4, 0, 7, 6, 11, -1, -1, -1, -1, -1, -1},
{ 11, 7, 6, 8, 3, 4, 3, 5, 4, 3, 1, 5, -1, -1, -1, -1},
{ 9, 5, 4, 10, 1, 2, 7, 6, 11, -1, -1, -1, -1, -1, -1},
{ 6, 11, 7, 1, 2, 10, 0, 8, 3, 4, 9, 5, -1, -1, -1, -1},
{ 7, 6, 11, 5, 4, 10, 4, 2, 10, 4, 0, 2, -1, -1, -1, -1},
{ 3, 4, 8, 3, 5, 4, 3, 2, 5, 10, 5, 2, 11, 7, 6, -1},
{ 7, 2, 3, 7, 6, 2, 5, 4, 9, -1, -1, -1, -1, -1, -1},
{ 9, 5, 4, 0, 8, 6, 0, 6, 2, 6, 8, 7, -1, -1, -1, -1},
{ 3, 6, 2, 3, 7, 6, 1, 5, 0, 5, 4, 0, -1, -1, -1, -1},
{ 6, 2, 8, 6, 8, 7, 2, 1, 8, 4, 8, 5, 1, 5, 8, -1},
{ 9, 5, 4, 10, 1, 6, 1, 7, 6, 1, 3, 7, -1, -1, -1, -1},
{ 1, 6, 10, 1, 7, 6, 1, 0, 7, 8, 7, 0, 9, 5, 4, -1},
{ 4, 0, 10, 4, 10, 5, 0, 3, 10, 6, 10, 7, 3, 7, 10, -1},
{ 7, 6, 10, 7, 10, 8, 5, 4, 10, 4, 8, 10, -1, -1, -1, -1},
{ 6, 9, 5, 6, 11, 9, 11, 8, 9, -1, -1, -1, -1, -1, -1},
{ 3, 6, 11, 0, 6, 3, 0, 5, 6, 0, 9, 5, -1, -1, -1, -1},
{ 0, 11, 8, 0, 5, 11, 0, 1, 5, 5, 6, 11, -1, -1, -1, -1},
{ 6, 11, 3, 6, 3, 5, 5, 3, 1, -1, -1, -1, -1, -1, -1},
{ 1, 2, 10, 9, 5, 11, 9, 11, 8, 11, 5, 6, -1, -1, -1, -1},
{ 0, 11, 3, 0, 6, 11, 0, 9, 6, 5, 6, 9, 1, 2, 10, -1},
{ 11, 8, 5, 11, 5, 6, 8, 0, 5, 10, 5, 2, 0, 2, 5, -1},
{ 6, 11, 3, 6, 3, 5, 2, 10, 3, 10, 5, 3, -1, -1, -1, -1},
{ 5, 8, 9, 5, 2, 8, 5, 6, 2, 3, 8, 2, -1, -1, -1, -1},
{ 9, 5, 6, 9, 6, 0, 0, 6, 2, -1, -1, -1, -1, -1, -1},
{ 1, 5, 8, 1, 8, 0, 5, 6, 8, 3, 8, 2, 6, 2, 8, -1},
{ 1, 5, 6, 2, 1, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 3, 6, 1, 6, 10, 3, 8, 6, 5, 6, 9, 8, 9, 6, -1},
{ 10, 1, 0, 10, 0, 6, 9, 5, 0, 5, 6, 0, -1, -1, -1, -1},
{ 0, 3, 8, 5, 6, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 10, 5, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 11, 5, 10, 7, 5, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 11, 5, 10, 11, 7, 5, 8, 3, 0, -1, -1, -1, -1, -1, -1},
{ 5, 11, 7, 5, 10, 11, 1, 9, 0, -1, -1, -1, -1, -1, -1},
{ 10, 7, 5, 10, 11, 7, 9, 8, 1, 8, 3, 1, -1, -1, -1, -1},
{ 11, 1, 2, 11, 7, 1, 7, 5, 1, -1, -1, -1, -1, -1, -1},
{ 0, 8, 3, 1, 2, 7, 1, 7, 5, 7, 2, 11, -1, -1, -1, -1},
{ 9, 7, 5, 9, 2, 7, 9, 0, 2, 2, 11, 7, -1, -1, -1, -1},
{ 7, 5, 2, 7, 2, 11, 5, 9, 2, 3, 2, 8, 9, 8, 2, -1},
{ 2, 5, 10, 2, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1},
{ 8, 2, 0, 8, 5, 2, 8, 7, 5, 10, 2, 5, -1, -1, -1, -1},
{ 9, 0, 1, 5, 10, 3, 5, 3, 7, 3, 10, 2, -1, -1, -1, -1},
{ 9, 8, 2, 9, 2, 1, 8, 7, 2, 10, 2, 5, 7, 5, 2, -1},
{ 1, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 8, 7, 0, 7, 1, 1, 7, 5, -1, -1, -1, -1, -1, -1},
{ 9, 0, 3, 9, 3, 5, 5, 3, 7, -1, -1, -1, -1, -1, -1},
{ 9, 8, 7, 5, 9, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 5, 8, 4, 5, 10, 8, 10, 11, 8, -1, -1, -1, -1, -1, -1},
{ 5, 0, 4, 5, 11, 0, 5, 10, 11, 11, 3, 0, -1, -1, -1, -1},
{ 0, 1, 9, 8, 4, 10, 8, 10, 11, 10, 4, 5, -1, -1, -1, -1},
{ 10, 11, 4, 10, 4, 5, 11, 3, 4, 9, 4, 1, 3, 1, 4, -1},
{ 2, 5, 1, 2, 8, 5, 2, 11, 8, 4, 5, 8, -1, -1, -1, -1},
{ 0, 4, 11, 0, 11, 3, 4, 5, 11, 2, 11, 1, 5, 1, 11, -1},
{ 0, 2, 5, 0, 5, 9, 2, 11, 5, 4, 5, 8, 11, 8, 5, -1},
{ 9, 4, 5, 2, 11, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 2, 5, 10, 3, 5, 2, 3, 4, 5, 3, 8, 4, -1, -1, -1, -1},
{ 5, 10, 2, 5, 2, 4, 4, 2, 0, -1, -1, -1, -1, -1, -1},
{ 3, 10, 2, 3, 5, 10, 3, 8, 5, 4, 5, 8, 0, 1, 9, -1},
{ 5, 10, 2, 5, 2, 4, 1, 9, 2, 9, 4, 2, -1, -1, -1, -1},
{ 8, 4, 5, 8, 5, 3, 3, 5, 1, -1, -1, -1, -1, -1, -1},
{ 0, 4, 5, 1, 0, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 8, 4, 5, 8, 5, 3, 9, 0, 5, 0, 3, 5, -1, -1, -1, -1},
{ 9, 4, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 11, 7, 4, 9, 11, 9, 10, 11, -1, -1, -1, -1, -1, -1},
{ 0, 8, 3, 4, 9, 7, 9, 11, 7, 9, 10, 11, -1, -1, -1, -1},
{ 1, 10, 11, 1, 11, 4, 1, 4, 0, 7, 4, 11, -1, -1, -1, -1},
{ 3, 1, 4, 3, 4, 8, 1, 10, 4, 7, 4, 11, 10, 11, 4, -1},
{ 4, 11, 7, 9, 11, 4, 9, 2, 11, 9, 1, 2, -1, -1, -1, -1},
{ 9, 7, 4, 9, 11, 7, 9, 1, 11, 2, 11, 1, 0, 8, 3, -1},
{ 11, 7, 4, 11, 4, 2, 2, 4, 0, -1, -1, -1, -1, -1, -1},
{ 11, 7, 4, 11, 4, 2, 8, 3, 4, 3, 2, 4, -1, -1, -1, -1},
{ 2, 9, 10, 2, 7, 9, 2, 3, 7, 7, 4, 9, -1, -1, -1, -1},
{ 9, 10, 7, 9, 7, 4, 10, 2, 7, 8, 7, 0, 2, 0, 7, -1},
{ 3, 7, 10, 3, 10, 2, 7, 4, 10, 1, 10, 0, 4, 0, 10, -1},

```

```

{ 1, 10, 2, 8, 7, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 9, 1, 4, 1, 7, 7, 1, 3, -1, -1, -1, -1, -1, -1},
{ 4, 9, 1, 4, 1, 7, 0, 8, 1, 8, 7, 1, -1, -1, -1},
{ 4, 0, 3, 7, 4, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 4, 8, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 9, 10, 8, 10, 11, 8, -1, -1, -1, -1, -1, -1, -1, -1},
{ 3, 0, 9, 3, 9, 11, 11, 9, 10, -1, -1, -1, -1, -1, -1},
{ 0, 1, 10, 0, 10, 8, 8, 10, 11, -1, -1, -1, -1, -1, -1},
{ 3, 1, 10, 11, 3, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 2, 11, 1, 11, 9, 9, 11, 8, -1, -1, -1, -1, -1, -1},
{ 3, 0, 9, 3, 9, 11, 1, 2, 9, 2, 11, 9, -1, -1, -1},
{ 0, 2, 11, 8, 0, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 3, 2, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 2, 3, 8, 2, 8, 10, 10, 8, 9, -1, -1, -1, -1, -1, -1},
{ 9, 10, 2, 0, 9, 2, -1, -1, -1, -1, -1, -1, -1, -1},
{ 2, 3, 8, 2, 8, 10, 0, 1, 8, 1, 10, 8, -1, -1, -1},
{ 1, 10, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 1, 3, 8, 9, 1, 8, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 9, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{ 0, 3, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1},
{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1};

/*
  Determine the index into the edge table which
  tells us which vertices are inside of the surface
*/
cubeindex = 0;
if (grid.val[0] < isolevel) cubeindex |= 1;
if (grid.val[1] < isolevel) cubeindex |= 2;
if (grid.val[2] < isolevel) cubeindex |= 4;
if (grid.val[3] < isolevel) cubeindex |= 8;
if (grid.val[4] < isolevel) cubeindex |= 16;
if (grid.val[5] < isolevel) cubeindex |= 32;
if (grid.val[6] < isolevel) cubeindex |= 64;
if (grid.val[7] < isolevel) cubeindex |= 128;

/* Cube is entirely in/out of the surface */
if (edgeTable[cubeindex] == 0)
  return(0);

/* Find the vertices where the surface intersects the cube */
if (edgeTable[cubeindex] & 1)
  vertlist[0] =
    VertexInterp(isolevel, grid.p[0], grid.p[1], grid.val[0], grid.val[1]);
if (edgeTable[cubeindex] & 2)
  vertlist[1] =
    VertexInterp(isolevel, grid.p[1], grid.p[2], grid.val[1], grid.val[2]);
if (edgeTable[cubeindex] & 4)
  vertlist[2] =
    VertexInterp(isolevel, grid.p[2], grid.p[3], grid.val[2], grid.val[3]);
if (edgeTable[cubeindex] & 8)
  vertlist[3] =
    VertexInterp(isolevel, grid.p[3], grid.p[0], grid.val[3], grid.val[0]);
if (edgeTable[cubeindex] & 16)
  vertlist[4] =
    VertexInterp(isolevel, grid.p[4], grid.p[5], grid.val[4], grid.val[5]);
if (edgeTable[cubeindex] & 32)
  vertlist[5] =
    VertexInterp(isolevel, grid.p[5], grid.p[6], grid.val[5], grid.val[6]);
if (edgeTable[cubeindex] & 64)
  vertlist[6] =
    VertexInterp(isolevel, grid.p[6], grid.p[7], grid.val[6], grid.val[7]);
if (edgeTable[cubeindex] & 128)
  vertlist[7] =
    VertexInterp(isolevel, grid.p[7], grid.p[4], grid.val[7], grid.val[4]);
if (edgeTable[cubeindex] & 256)
  vertlist[8] =
    VertexInterp(isolevel, grid.p[0], grid.p[4], grid.val[0], grid.val[4]);
if (edgeTable[cubeindex] & 512)
  vertlist[9] =
    VertexInterp(isolevel, grid.p[1], grid.p[5], grid.val[1], grid.val[5]);
if (edgeTable[cubeindex] & 1024)
  vertlist[10] =
    VertexInterp(isolevel, grid.p[2], grid.p[6], grid.val[2], grid.val[6]);
if (edgeTable[cubeindex] & 2048)
  vertlist[11] =

```

```

        VertexInterp(isolevel,grid.p[3],grid.p[7],grid.val[3],grid.val[7]);

/* Create the triangle */
ntriang = 0;
for (i=0;triTable[cubeindex][i]!=-1;i+=3) {
    triangles[ntriang].p[0] = vertlist[triTable[cubeindex][i  ]];
    triangles[ntriang].p[1] = vertlist[triTable[cubeindex][i+1]];
    triangles[ntriang].p[2] = vertlist[triTable[cubeindex][i+2]];
    ntriang++;
}

return(ntriang);
}

/*
  Linearly interpolate the position where an isosurface cuts
  an edge between two vertices, each with their own scalar value
*/
XYZ VertexInterp(isolevel,p1,p2,valp1,valp2)
double isolevel;
XYZ p1,p2;
double valp1,valp2;
{
    double mu;
    XYZ p;

    if (ABS(isolevel-valp1) < 0.00001)
        return(p1);
    if (ABS(isolevel-valp2) < 0.00001)
        return(p2);
    if (ABS(valp1-valp2) < 0.00001)
        return(p1);
    mu = (isolevel - valp1) / (valp2 - valp1);
    p.x = p1.x + mu * (p2.x - p1.x);
    p.y = p1.y + mu * (p2.y - p1.y);
    p.z = p1.z + mu * (p2.z - p1.z);

    return(p);
}

```

付録 B

Quaternion を用いた変換パラメータの解析的 解法

4次元ベクトルで表される Quaternion を用いて, シーンとモデルの対応から回転と平行移動のパラメータを求める方法を説明する [43].

B.1 Quaternion による回転の表現

次のように定義される 4次元ベクトル $\mathbf{q} = (u, v, w, s)^T$ を用いて回転を表現する. この 4次元ベクトルは Quaternion と呼ばれ, 3次元ベクトル $\mathbf{u} = (u, v, w)^T$ とスカラー s からなるとみなされる.

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \mathbf{u}_1 \cdot \mathbf{u}_2 + s_1 s_2$$

$$|\mathbf{q}| = (\mathbf{q} \cdot \mathbf{q})^{-\frac{1}{2}}$$

Quaternion の積は次のように定義される.

$$\mathbf{q}_1 * \mathbf{q}_2 = ((s_1 \mathbf{u}_2 + s_2 \mathbf{u}_1 + \mathbf{u}_1 \times \mathbf{u}_2), s_1 s_2 - \mathbf{u}_1 \cdot \mathbf{u}_2)^T \quad (\text{B.1})$$

また, \mathbf{q} と共役な Quaternion $\bar{\mathbf{q}}$ は $(-\mathbf{u}, s)$ で表される. 行列 R をベクトル \mathbf{v} のまわりに角度 θ の回転を表す行列とすると Quaternion \mathbf{q} と行列 R の対応は次のようになる.

$$R\mathbf{x} = \mathbf{q} * \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} * \bar{\mathbf{q}} \quad (\text{B.2})$$

$$\mathbf{u} = \sin \frac{\theta}{2} \mathbf{v}, \quad s = \cos \frac{\theta}{2} \quad (\text{B.3})$$

ここで, $\mathbf{x} = (x, y, z)^T$ の 3次元ベクトルである.

B.2 変換パラメータの解析的解法

モデルの点 \mathbf{x}_i とシーンの点 \mathbf{x}'_i の対応 ($i = 1, \dots, N$) が与えられたとき, その変換のパラメータ, 平行移動ベクトル \mathbf{t} と回転 Quaternion \mathbf{q} は次の式から与えられる.

$$\text{Min} \sum_{i=1}^N |\mathbf{q} * \mathbf{x}_i * \bar{\mathbf{q}} + \mathbf{t} - \mathbf{x}'_i|^2 \quad (\text{B.4})$$

この式は次のように変形される.

$$\text{Min} \sum_{i=1}^N |\mathbf{q} * \mathbf{x}_i - \mathbf{x}'_i * \bar{\mathbf{q}} + \mathbf{t} * \mathbf{q}|^2 \quad (\text{B.5})$$

式 (B.2) から $\mathbf{q} * \mathbf{x}_i - \mathbf{x}'_i * \bar{\mathbf{q}}$ は行列 A_i を用いて次のように表される .

$$\mathbf{q} * \mathbf{x}_i - \mathbf{x}'_i * \bar{\mathbf{q}} = \begin{pmatrix} 0 & z + z' & -(y + y') & x - x' \\ -(z + z') & 0 & x + x' & y - y' \\ y + y' & -(x + x') & 0 & z - z' \\ -(x - x') & -(y - y') & -(z - z') & 0 \end{pmatrix} \mathbf{q} = A_i \mathbf{q} \quad (\text{B.6})$$

Quaternion $\mathbf{t}' = \mathbf{t} * \mathbf{q}$ を用いて 8 次元ベクトル $V = (\mathbf{q}, \mathbf{t}')^T$ を定義するとこの問題は次のように表される .

$$\text{Min} V^T B V \quad (\text{B.7})$$

最小化の問題は $|\mathbf{q}| = 1$ と $\mathbf{q} \cdot \mathbf{t}' = 0$ によっている . ここで , 対称行列 B は次のように表される .

$$B = \begin{pmatrix} A & C \\ C^T & N \times I \end{pmatrix} \quad (\text{B.8})$$

ここで , I は単位行列であり ,

$$A = \sum_{i=1}^N A_i^T A_i \quad (\text{B.9})$$

$$C = \sum_{i=1}^N A_i \quad (\text{B.10})$$

である . 以上から解は次のように与えられる .

$$\mathbf{t}' = C \mathbf{q}_{\min} / N \quad (\text{B.11})$$

$$\mathbf{t} = \mathbf{t}' * \bar{\mathbf{q}} \quad (\text{B.12})$$

\mathbf{q}_{\min} は行列 $A - C^T C / N$ の最小の固有値 λ_{\min} を持つ固有ベクトルである .

参考文献

-
- [1] D.J.Kriegman and J.Ponce. on recognizing and positioning curved 3-d objects from image contours. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 12, No. 12, pp. 1127–1137, 1990.
 - [2] R.Basri and S.Ullman. The alignment of objects with smooth surfaces. *CVGIP: Image Understanding*, Vol. 57, No. 3, pp. 331–345, 1993.
 - [3] M.Zerroug and R.Nevatia. Pose estimation of multi-part curved object. In *Proc. Int'l Symposium on Computer Vision*, pp. 431–436, 1995.
 - [4] Y.Sumi, Y.Kawai, T.Yoshimi, and F.Tomita. Recognition of 3d free-form objects using segment-based stereo vision. In *Proc. Int'l Conf. on Computer Vision*, pp. 668–674. IEEE, 1998.
 - [5] Mark D. Wheeler. *Automatic Modeling and Localization for Object Recognition*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1996.
 - [6] F.Arman and J.K.Aggarwal. Model-based object recognition in dense-range images — a review. *ACM Computing Surveys*, Vol. 25, No. 1, pp. 5–43, 1993.
 - [7] C.Chua and R. Jarvis. 3-d free-form surface registration and object recognition. *Int'l Jour. Computer Vision*, Vol. 17, No. 1, pp. 77–99, 1996.
 - [8] C.Dorai and A.K.Jain. Recognition of 3d free-form objects. In *Proc. 13th Int'l Conf. on Pattern Recognition*, Vol. I, pp. 697–701, 1996.
 - [9] R.A.Brooks. Model-based three-dimensional interpretations of two-dimensional images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 5, No. 2, pp. 140–150, 1983.
 - [10] N.Raja and A.Jain. Recognizing geons from superquadrics fitted to range data. *Image and Vision Computing*, Vol. 10, No. 3, pp. 179–190, 1992.
 - [11] P.Whaite and F.P.Ferrie. From uncertainty to visual exploration. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 13, No. 10, pp. 1038–1049, 1991.

-
- [12] D. Scharstein and R. Szeliski. Stereo Matching with Nonlinear Diffusion. *International Journal of Computer Vision*, Vol. 28, No. 2, pp. 155–174, June/July. 1998.
- [13] K. Konolige. Small Vision Systems: Hardware and Implementation. In Y. Shirai and S. Hirose, editors, *Robotics Research: The Eighth International Symposium*, pp. 203–212. Springer, 1997.
- [14] 金出武雄, 蚊野浩, 木村茂, 川村英二, 吉田収志, 織田和夫. ビデオレートステレオマシンの開発. *日本ロボット学会誌*, Vol. 15, No. 2, pp. 261–267, Mar. 1997.
- [15] O. Faugeras, B. Hots, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real Time Correlation-Based Stereo: Algorithm, Implementations and Applications. Technical Report N°2013, INRIA, 1993.
- [16] 加賀美聡, 岡田慧, 稲葉雅幸, 井上博允. ロボット搭載用実時間視差画像生成システム. 第4回ロボティクスシンポジウム予稿集, pp. 177–182, 1999.
- [17] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH'94*, pp. 311–318. ACM, 1994.
- [18] M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitrary shaped objects. In *Proc. Computer Vision and Pattern Recognition*, pp. 573–580. IEEE, June 1994.
- [19] C.I. Conolly. Cumulative generation of octree models from range data. In *Proc. Intl. Conf. Robotics*, pp. 25–32, March 1984.
- [20] C.H. Chien, Y.B. Sim, and J.K. Aggarwal. Generation of volumetric/surface octree from range data. In *Proc. Computer Vision and Pattern Recognition*, pp. 254–260, June 1988.
- [21] W. Lorensen and H. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *Proc. SIGGRAPH'87*, pp. 163–170. ACM, 1987.

- [22] H. Hoppe, T. DeRose, T. Duchamp, J.A. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proc. SIGGRAPH'92*, pp. 71–78. ACM, 1992.
- [23] A.Hilton, A.J.Stoddart, J.Illingworth, and T.Windeatt. Reliable surface reconstruction from multiple range images. In *Proceedings of the European Conference on Computer Vision*, pp. 117–126, Springer-Verlag, 1996.
- [24] M.D. Wheeler, Y. Sato, and K. Ikeuchi. Consensus surfaces for modeling 3d objects from multiple range images. In *Proc. International Conference on Computer Vision*, January 1998.
- [25] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH'96*, pp. 303–312. ACM, 1996.
- [26] F.Stein and G.Medioni. Structural indexing. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 14, No. 2, pp. 125–145, 1992.
- [27] A.E. Johnson and M.Hebert. Efficient multiple model recognition in cluttered 3-d scenes. In *Proc. Computer Vision and Pattern Recognition*, pp. 671–677, 1998.
- [28] R. Y. Tsai. An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp. 364–374, 1986.
- [29] 岡田慧, 加賀美聡, 稲葉雅幸, 井上博允. 実時間3次元視覚の時空間統合による運動空間再構成法 – ロボット用実時間時空間再構成の研究 (その4) –. 第17回ロボット学会学術講演会予稿集, pp. 32–33, 1999.
- [30] Peter Rander. *A Multi-Camera Method for 3D Digitization of Dynamic, Real-World Event*. PhD thesis, The Robotics Institute Carnegie Mellon University, December 1998. CMU-RI-TR-98-12.
- [31] R. Bolles and J. Woodfill. Spatiotemporal Consistency Checking of Passive Range Data. In T. Kanade and R. Paul, editors, *Robotics Research: The Sixth International Symposium*, pp. 165–183. International Foundation for Robotics Research, 1993.

- [32] P. Fua. A Parallel Stereo Algorithm that Produces Dense Depth Maps And Preserves Images Features. In *Machine Vision and Applications*, pp. 35–49, 1991.
- [33] 松本吉央, 坂井克弘, 稲邑哲也, 稲葉雅幸, 井上博允. PC ベースのハイパーマシン. 第 15 回日本ロボット学会学術講演会予稿集, pp. 979–980, 1997.
- [34] Y. Matsutomo, T. Shibata, K. Sakai, M. Inaba, and H. Inoue. Real-Time Color Stereo Vision System for a Mobile Robot based on Field Multiplexing. In *In Proc. of IEEE Int. Conf. on Robotics and Automation*, pp. 1934–1939, 1997.
- [35] 加賀美聡, 花澤光隆, 岡田慧, 松本吉央, 近野敦, 稲葉雅幸, 井上博允. 脚型ロボット感覚行動統合研究プラットフォーム JROB-1. 日本ロボット学会誌, Vol. 16, No. 5, pp. 47–52, 1998.
- [36] M.Okutomi and T.Kanade. A multiple-baseline stereo. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 15, No. 4, pp. 353–363, 1993.
- [37] 佐川立昌, 岡田慧, 加賀美聡, 稲葉雅幸, 井上博允. 複数視点からの距離画像を用いた高速表面再構築法 – ロボット用実時間時空間再構成の研究 (その 4) –. 第 17 回ロボット学会学術講演会予稿集, pp. 30–31, 1999.
- [38] C.L.Jackins and S.L.Tanimoto. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, Vol. 14, pp. 249–270, 1980.
- [39] P.J.Besl and N.D.Mckay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 14, No. 2, pp. 239–256, 1992.
- [40] Z.Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, Vol. 13, No. 2, pp. 119–152, 1994.
- [41] W.E.L Grimson and T.L.Pérez. Localizing overlapping parts by searching the interpretation tree. *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 9, No. 4, pp. 469–482, 1987.

- [42] A.E.Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Carnegie Mellon University, 1997. CMU-RI-TR-97-47.
- [43] O.D.Faugeras and M.Hebert. The representation, recognition, and locating of 3-d objects. *Int'l Jour. of Robotics Research*, Vol. 5, No. 3, pp. 27–52, 1986.
- [44] D.A.Simon, M.Hebert, and T.Kanade. Real-time 3-d pose estimation using a high-speed range sensor. In *Proc. Int'l Conf. Robotics and Automation(R&A'94)*, pp. 2235–2241. IEEE, 1994.
- [45] J.Friedman, J.Bentley, and R.Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. on Mathematical Software*, Vol. 3, No. 3, pp. 209–226, September 1977.
- [46] P.Heckbert and M.Garland. Survey of polygonal surface simplification algorithms. Technical Report CMU-CS-97-TBD, The School of Computer Science, Carnegie Mellon University, 1997.
- [47] H.Hoppe, T.DeRose, T.Duchamp, J.McDonald, and W.Stuetzle. Mesh optimization. In *Computer Graphics(SIGGRAPH'93 Proceedings)*, pp. 19–26. ACM, August 1993.
- [48] 加賀美聡, 近野敦, 陰山竜介, 椋澤光隆, 稲葉雅幸, 井上博允. 視触覚行動統合研究のための車輪移動上半身型ヒューマノイドH4の設計と開発. 第16回ロボット学会学術講演会予稿集, 第2巻, pp. 835–836, 1998.
- [49] 稲邑哲也, 佐川立昌, 稲葉雅幸, 井上博允. 照明条件の変動下における人間の発見追従行動のための実時間視覚処理. 第16回ロボット学会学術講演会予稿集, 第3巻, pp. 1039–1040, 1998.
- [50] 岡田慧, 加賀美聡, 稲葉雅幸, 井上博允. 再帰相関法とマルチメディア命令による高速オプティカルフロー計算法. 情報処理学会 第115回 CVIM 研究会予稿, pp. 127–132, Mar. 1999.
- [51] 三浦淳, 池内克史. 作業目的を考慮した視覚認識戦略の生成. 日本ロボット学会誌, Vol. 14, No. 4, pp. 574–585, 1996.

-
- [52] 滝沢穂高, 白井良明, 三浦純. 注視・ズームを用いた自律移動ロボットのための3dシーン記述の選択的精密化. 日本ロボット学会誌, Vol. 13, No. 7, pp. 963–970, 1995.
- [53] 伊庭斉志. 環境モデルにおける物体の見え方と見方. PhD thesis, 東京大学大学院工学系研究科情報工学専攻, 1987.