PCクラスタを利用した実時間並列画像処理のための プログラミングツールRPV

[1] 花田武彦 [2] 有田大作 [2] 谷口倫一郎
[1] 九州大学 大学院システム情報科学府
[2] 九州大学 大学院システム情報科学研究院
{hanada,arita,rin}@limu.is.kyushu-u.ac.jp

1 はじめに

画像から複雑な対象を理解する場合, 複数視点からの画像を用いることは, より多くの情報が得られるという点で有効である. 死角を減らしたり, 3次元的な情報を得たりすることが出来る. しかし, このような多視点情報を実時間で処理する場合, 複数のカメラを同時に接続する必要があるため, 一台の計算機では接続台数の制限が問題となる. さらに, 同時に得られる画像が増加するため, 計算機の計算能力の限界も問題となる.

これらの問題に対し, 計算機とカメラの対を複数用意し, それらをネットワークで相互に接続して分散システムとしてコンピュータビジョンシステムを構築しようという研究がなされている [1, 2]. このような分散システムでは, カメラの台数による制約は特になく, 必要に応じて計算機とカメラの対を追加するだけで視点数を増やすことが出来る. また, 視点の数に見合う計算能力も得られるという大きな利点がある. このような背景から, 本研究では, 通常のPCをネットワークで接続したPCクラスタに複数のビデオカメラを接続し, その上で実時間並列画像処理を行うことにした.

しかし, PCクラスタ上で実時間並列画像処理を行なう場合, 実時間性を保証したり, 処理するデータ同士の整合性を取ったりするために同期機構が必要となり, プロセス管理や割り込み処理や排他制御などを考慮しなければならない. 本論文では, PCクラスタを利用する実時間並列画像処理システムの開発を容易にするプログラミングツールRPV (RPV:Real-time Parallel Vision) の概要と, RPVを用いた実アプリケーションについて述べる.

2 PCクラスタを利用した実時間並列画像処理

2.1 PCクラスタの概要

PCクラスタは, 分散型並列計算機の一種で, ノードの計算機にPCを用いたものである. ノードとなるPCはネットワークで相互に接続されている. 本研究で用いるPCクラスタは24台のPC(ノード)から構成されており, 9台のCCDカメラがIEEE1394で接続されている [3, 4]. これらのCCDカメラは, カメラ外部から同期信号を入力して撮影時刻の同期をとっている. 各PCは2個のプロセッサを搭載したPC-AT互換機で, 図1に示すようにMyrinet[5]によって相互に結合されている. また, Myrinetとは別にEthernetでも接続されている. 各PCのオペーレーティングシステムにはLinuxを用いている. また, Myrinet上の通信には, 新情報処理開発機構が開発したPM高性能通信ライブラリ [6] を利用している. 以下に, 使用したハードウェアを列挙する.

- PC
  - MPU : PentiumIII 700MHz (14台)
  - : PentiumIII 450MHz (10台)
  - Memory : SDRAM 384 MBytes
  - IEEE1394 I/F : メルコ IFC-IL3
  - Myrinet I/F : Myricom M2M-PCI64
  - Ethernet I/F : Intel EtherExpress Pro 100+
- Myrinet スイッチ
  Myricom M2M-SW16
- CCD カメラ
  Sony DFW-V500
- 同期信号発生器 (自作)

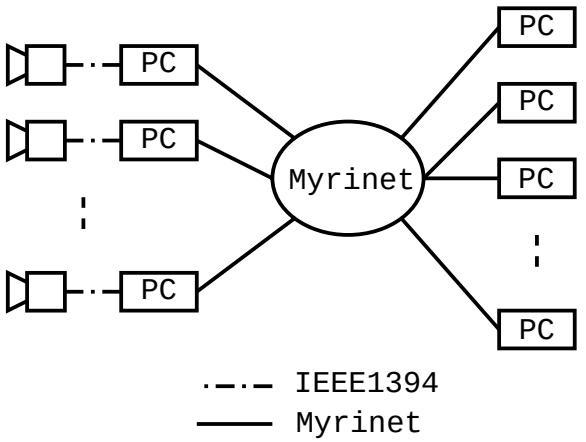


図 1: PC クラスタの構成

2.2 実時間処理

本研究で想定する実時間画像処理とは, ビデオカメラのように画像をフレームデータとして周期的に与える入力に対し, 各フレームデータに対する処理を滞りなく行なうものである. 決められた時間内にフレームデータの処理を終えることが要求されるが, なんらかの理由で時間内に処理を終えることが出来なかった場合は, 処理を中断し, コマ落ちという形でフレームを飛ばしたり途中経過を出力したりして, 遅れ状態から回復する必要がある.

2.3 並列処理

実時間処理システムの性能を評価する場合, 単位時間あたりの処理フレーム数 (スループット) とシステムにデータを入力してから結果が出てくるまでに要するフレーム数 (レイテンシ) を考える必要がある. データや処理アルゴリズムに内在する並列性を生かすことで, 性能を向上させることが出来る. 本研究で想定するような画像処理の場合, 性能を向上させるには, 以下の処理方式が考えられる.

- パイプライン処理
  図 2 (a) に示すように, 処理を複数の段階 (ステージ) に分け, 各ステージ毎にノードを割り当てる方式である. 実時間並列画像処理をパイプライン化する場合, 各ノードに異なるフレームの異なるステージの作業を順次同時に行わせることでスループットが大きくなる. ノードが増えればその分だけ並列に処理されるフレーム数が増えるため, ノード数に比例してスループットが向上する. ただし, パイプラインの最初のノードが処理を始めて, 最後のノードが結果を出すまでの時間もノード数に比例するため, レイテンシが大きくなってしまうという欠点もある.

- データ並列処理
  あるフレームデータを部分データに分割し, 複数のノードに分担処理させる並列処理であり, 図 2 (b) に示すように接続される. ノードを増やすことで, ノード 1 台当たりの処理データの量を減らすことが出来るので, 処理時間が短縮し, レイテンシを増大させることなくスループットを向上させることが出来る. 複数のカメラがあって, カメラ毎に処理するノードを配置する場合も, このデータ並列処理に含まれる. データの内容によって処理時間が変わってくるような場合は, 処理時間の最も長いノードによって全体の処理時間が決定されるため, 適切にデータを分割しなければノードの数に応じた性能向上は得られない.

- 機能並列処理
  ある処理を並列性のある部分処理に分割して, 複数のノードに分担させて処理させる並列処理であり, 図 2 (c) に示すように接続される. ノードを増やすことで, ノード 1 台当たりの処理量を減らすことが出来るので, 処理時間が短縮し, スループットが向上する. 処理の内容に並列性が必要となるので, 処理の内容によっては並列化出来ないこともある. また, 機能並列しているノード間で処理時間が異なると, データ並列処理と同様に, ノードの数に比例した性能向上は得られない.

2.4 同期処理と例外処理

多視点情報を扱う上で, 各視点からの画像が同期していることは重要なことである. 本研究では, 各カメラに外部から同期信号を入力することで, 撮影時刻の同期を取っている. 外部からの同期信号は, 自作の外部同期信号発生器で発生させており, 0.6Hz から 15Hz までの周波数に対応している. なお, 本研究で使用しているカメラは, 同期信号を入力する場合, 15Hz 以下でしか撮影出来ない.

外部同期信号で視点間の同期が取れても, PC クラスタ上でデータを転送・処理を繰り返す間に, 処理遅れやコマ落ちによって処理とデータの同期が取れなくなること (同期外れ) が考えられる. 例えば, コマ落ちによってデータが届かない場合には, 例外処理を行なわなければなら

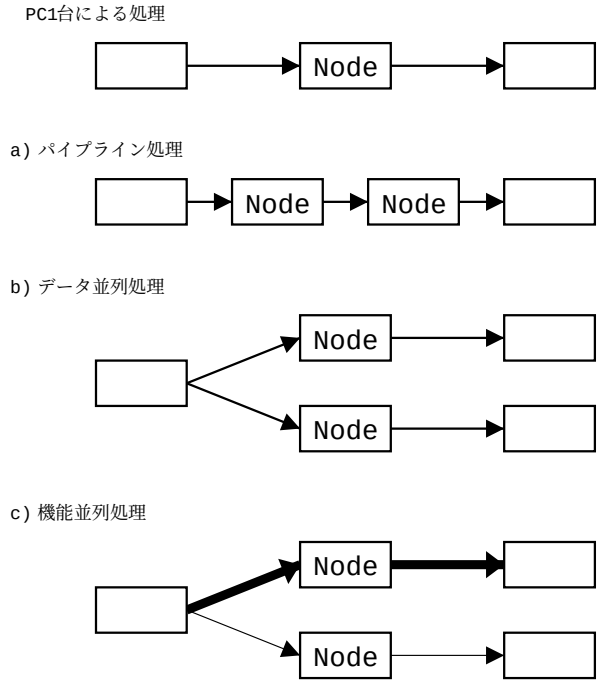


図 2: 並列処理の方式

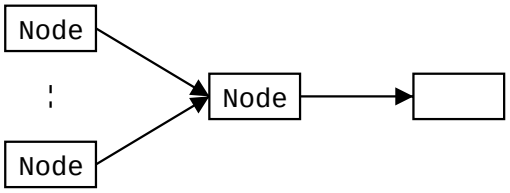


図 3: バリア同期

ない. また, あるノードで処理遅れが生じれば, その処理を打ち切るか, パイプライン上で後続のノードでそれに応じた処理を行なわなければならない. さらに, 図 3 のように複数のパイプラインからのデータを統合して処理するノード, つまり, 多視点情報を統合して認識などを行なうノードでは, 各入力間のバリア同期を考慮する必要が出てくる.

3 プログラミングツール RPV

本研究で開発した RPV は, PC クラスタを利用した実時間並列画像処理システムのためのプログラミングツールであり, PC クラスタ上のアプリケーション構築を容易にする. 本節では, RPV の概要と実現方法, 使用法について詳細を述べる.

3.1 RPV の概要

RPV は, 2 節で述べた実時間並列画像処理を PC クラスタ上で PC クラスタ上で行なうのに必要な, 並列処理のためのノード間のデータ転送, データや処理の同期と同期外れに対する例外処理の機構を提供する. これらの機構により, アプリケーションプログラマは, パイプライン処理を考慮したデータフローの作成と, 並列性のあるデータや処理を分割した実装を行なうだけで, PC クラスタ上にアプリケーションを構築出来る.

3.2 モジュール構成

RPV では, 各ノードの上で 4 種類のモジュールが, 図 4 のように協調して動作している. 各モジュールは LinuxThreads[7] を利用して実装されている. これによりデータの受信, 処理, 送信を並列して行うことが出来, 全体として滞りなく処理が進んで行く. 以下に各モジュールを列挙する.

- Data Receiving Module (DRM)
  データの受信処理を行なう. データを受信後, データを受信キューへ入れ, DPM へ同期のためのシグナルを送信する.
- Data Sending Module (DSM)
  データの送信処理を行なう. 送信キューから, DPM が入れたデータを取り出して送信する.
- Data Processing Module (DPM)
  データの各種同期処理や同期外れ時の例外処理, データ処理を行なう.
- Frame Synchronizing Module (FSM)
  同期処理の基準となるフレーム同期信号 (Frame Synchronizing Signal:FSS) を, 全てのノードに行き渡らせるモジュールである. 他のノードの FSM との間で FSS の送受信を行ない, FSS 送信時には, DPM へ同期のためのシグナルを送信する.

4 時刻管理

実時間並列処理を行なう場合, 時刻の管理は重要である. RPV では, FSM が各ノード間で送受信する FSS に

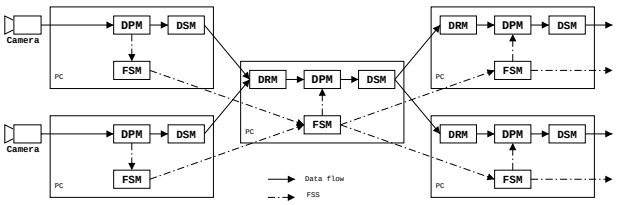


図 4: 各ノード上のモジュール構成

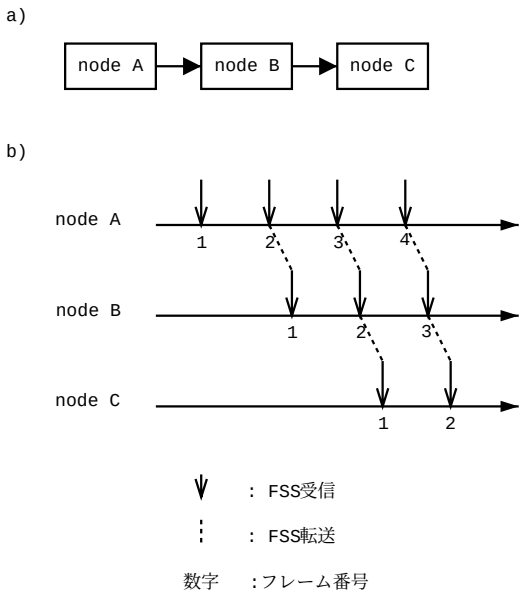


図 5: FSS の伝達

よって時刻を刻んでおり, この時刻は, RPV 上の各種同期処理に利用される. FSS は, パイプラインの入口のノードで, カメラから画像を 1 枚得られる度に一つ発生するので, FSS を用いて同期が取られている処理やデータは, カメラとの同期も取れていることになる.

FSS は, パイプラインの入口のノードで生成され, パイプラインの出口のノードへ順に伝播していく. あるノードにおいて, 入力元ノードが一つの場合, FSS を受信すると, 出力先ノードへ即座に FSS を送信する. 入力元ノードが複数の場合, 全入力元ノードからの FSS を受信してから, 出力先ノードへ FSS を送信する. ただし, どの FSM も, 最初に受信した FSS は伝播させないので, ノードの時刻はパイプラインの入口に近ければ近いほど進んでいる. つまり, 図 5 (a) のようにノードを接続した場合の FSS の伝達状況は, 図 5 (b) の様になる.

4.1 データ転送機構

4.1.1 データ送受信の非同期処理

RPV では, 受信処理を DRM に, 送信処理を DSM に任せ, それぞれをスレッドとして実装している. また, DRM と DSM には転送データを貯めておくキューが備わっている. そのため, DPM によるデータ処理と, DRM と DSM によるデータ送受信を非同期に実行することが出来る. よって, データの送受信によって画像処理が滞ってしまったり, 画像処理によってデータの送受信が滞ってしまったりすることはなく, 次から次へとデータが流れる動画像処理を実現するのに適している.

4.1.2 転送データの管理

DRM と DSM のキューは, フレーム単位で転送データをリストで管理している. アプリケーションプログラマは, 受信したデータをフレームを越えて保持し続ける場合に, そのデータを格納したバッファをリストから切り離しておけばよく, コピーをする必要がない. フレーム間差分のような複数フレームにまたがったデータ処理を行なう場合には, データの無駄なコピーを省くことが出来るため, 特に有効である.

4.1.3 転送方式

RPV は 2 種類のデータ転送方式を実現している. 二つの転送方式の大きな違いは, フレームの処理と転送データが同期しているか否かであり, 同期している転送方式を同期転送方式, 同期していない転送方式を非同期転送方式と呼ぶことにする. ノードへの入力データ毎に, どちらかの転送方式を選択することが出来る. 以下, 両転送方式について説明する.

- 同期転送方式
  動画像や画像処理の結果のように, フレーム毎にデータを転送する場合で, 毎フレーム届く必要がある場合に用いられる. 受信側で遅れや欠落を検出出来るが, 1 フレームに 1 回しか用いることが出来ない. 遅れや欠落を検出した場合, それに対する例外処理を行なうことが出来る (4.2 節). この方式で転送されたデータを同期データと呼ぶことにする.

- 非同期転送方式
  この転送方式の場合, RPV は送受信処理以外の管理

を行なわない．よって, データの着信状況の判断や例外処理などはアプリケーションプログラマに任される．1 フレームに何回でも用いることが出来, また数フレームに 1 回だけしか用いないことも可能である．通常, フィードバックデータや, 協調処理のための制御信号などに利用される．この方式で転送されたデータを非同期データと呼ぶことにする．

4.2 同期機構と例外処理

同期機構の役目は, 転送データとフレームの同期や, データ処理と FSS の同期がとれていない場合を検出して, つまり, データの到着や処理の遅れを検出して, それに対する処理を行なうことである．しかし, 同期外れに対してどういう対処を行うべきかは, アプリケーションによって違う．そこで, RPV では, 様々な状況に対してどう対処するかをアプリケーションが決める仕組みを提供している．アプリケーションは以下に挙げる状況を把握出来る．

- FSS の受信状況
- 任意フレーム番号の同期データの受信状況
  - 受信済
  - 未受信
  - 欠落

アプリケーションが, 上記の各状況を考慮して対処法を以下から選択すれば, RPV はそれに沿って各種処理を行なう．

- データ処理を開始
- 同期データの受信待ち
- コマ落
- アプリケーション終了

4.3 アプリケーションプログラミング

RPV は C++で記述されたクラスライブラリとして提供されている．RPV を用いてアプリケーションを開発する場合, アプリケーションプログラマの作業には, 以下の三つがあるので, 順を追って説明する．

- 設定ファイルの準備
- データ処理モジュール DPM の定義
- RPV 起動処理の記述

4.3.1 設定ファイルの準備

用意する設定ファイルには, 各ノードでどの DPM を動作させるかや, データフローをどのようにするか, と言った画像処理システム全体の構成を記述しておく．必要な情報は, 大きく分けて以下の 3 種類がある．

- ノード情報
- ポート情報
- 接続情報

ノード情報には, 使用するノードの番号 (PM 高性能通信ライブラリ上でノードを識別する番号) とそのノード上で動作させる DPM の名前を記述しておく．オプションで DPM へ渡すパラメータを指定することも出来る．ポート情報には, 各ノードで使用する送受信ポートの属性 (ポート番号, データサイズ, キューの長さ) を記述する．ここで言う送受信ポートとは, データ転送の端点を指し, 実装上は DRM や DSM が持つキューを指す．接続情報には, ポート情報で与えられる各送受信ポート間の接続関係を記述する．例えば, ノード 1 番の 5 番ポートからノード 4 番の 3 番ポートへ接続する, と言った具合である．この場合, ノード 1 番の 5 番ポートのキューは DSM が, ノード 4 番の 3 番ポートのキューは DRM が持つことになる．

4.3.2 データ処理モジュール DPM の定義

アプリケーションプログラマは, 雛型として用意されている DPM の基本クラスを継承して, データ処理モジュールを定義する．必要に応じて以下の関数を作成し, アプリケーション独自の処理を定義出来る．

- 前処理関数
- 後処理関数
- データ処理関数 (必須)
- 例外処理関数

これらの関数は図 6 のフローチャートに示す手順で実行される．

前処理関数は, データ処理が開始される前に 1 度だけ実行される関数である．データ処理に必要な情報を揃えるのに用いられる．例えば, 変換テーブルの作成やデバイ

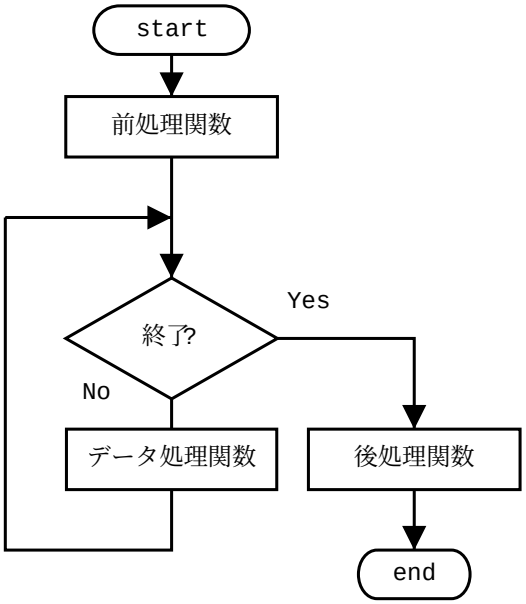


図 6: DPM の処理手順

スのオープン, メモリの確保などを想定している. 後処理関数は, 前処理関数と対になる関数で, データ処理が終了した後に 1 度だけ実行される関数である.

データ処理関数は, 一フレームのデータ処理関数である. この関数には, ノードに入力されたデータに対して, アプリケーション固有の処理施し, 出力データを作成するコードを記述しておく. 受信データを獲得するには, DRM のキューから取り出すだけで良い. また, データを送信するには, DSM のキューに繋ぐだけで良い.

例外処理関数は, 4.2 節で述べた処理を記述しておく関数である. 必要に応じてアプリケーション側の例外処理を記述しても良い. この関数は, RPV の判断で必要に応じて実行される. 例外処理関数のサンプル疑似コードを図 7 に示す.

4.3.3 RPV 起動処理の記述

RPV の起動処理は至って簡単である. アプリケーションプログラマは, RPV から指定された DPM のインスタンスを生成し, RPV を起動する関数を呼び出すだけでよい.

```
// 現在のフレームの受信データの受信状況を獲得する.
state = receive_state(queue, current_frame);

// RPV に対処法を通知する.
switch(state){
case 受信済:
  return データ処理開始;
case 未受信:
  return 受信待ち;
case 欠落:
  return コマ落ち;
default: // 予期せぬ状況
  return アプリケーション終了;
}
```

図 7: 例外処理関数のサンプル疑似コード

5 RPV を用いた実アプリケーション

RPV は, PC クラスタ上の実時間並列画像処理アプリケーション構築を容易にすることを目的としている. そこで, 実際の応用システムとして, シルエット画像を用いた視体積交差法 [8] による 3 次元形状復元システムを構築した.

5.1 視体積交差法

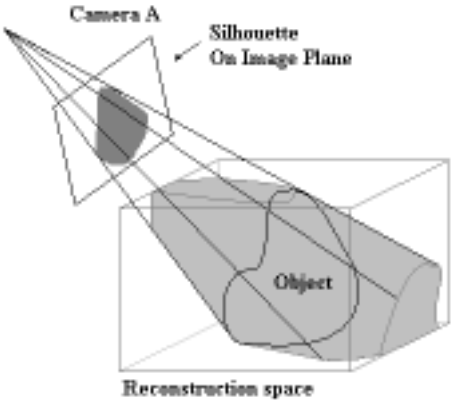
視体積交差法とは, 多視点からの画像から対象物体の 3 次元形状復元を行なう手法である. ある視点から見たシルエットを元に, 対象物体が存在し得る部分空間を求めると, カメラの位置を頂点, その視点から見た対象物体のシルエットを断面とする錐体となる (図 8). これを視体積と呼ぶ. この視体積を各視点毎に求め, それらの交差する部分を抜き出す (視体積交差) ことにより, 3 次元形状を求めることが出来る (図 9).

5.2 3 次元形状復元システム

視体積交差法の処理手順は大まかに以下のようになる.

- 画像獲得, シルエット画像作成
- 視体積 (ボクセルデータ[1]) 作成
- 視体積交差
- 3 次元形状の画面出力

[1] 平面の画像を格子で区切ったものをピクセルと呼ぶように, 空間を格子で区切ったものをボクセルと呼ぶ.



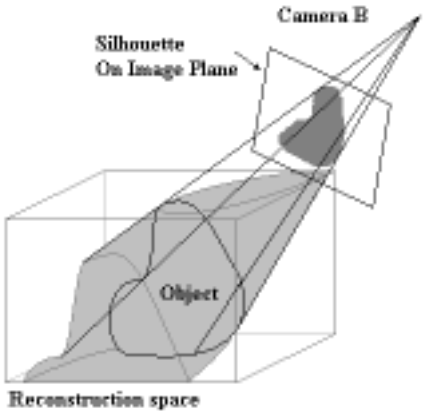


図 8: 視点毎にシルエットから視体積を求める

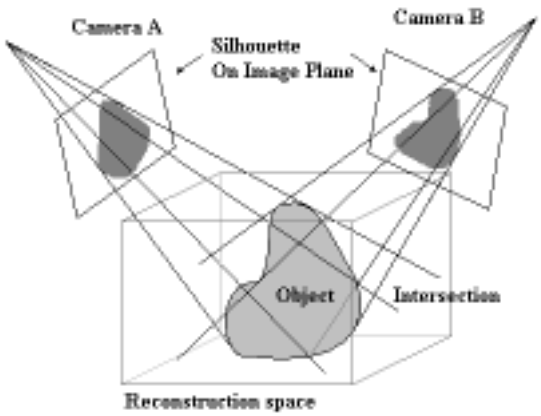


図 9: 視体積交差法による 3 次元形状復元

表 1: 3 次元形状復元を行なう計測空間

計測空間	$2000 \times 2000 \times 2000mm^3$
ボクセル	1 ビット $/25 \times 25 \times 25mm^3$
ボクセル数	$80 \times 80 \times 80$ 個
総ボクセルサイズ	64000 バイト

負荷分散を考えないことにして, これらの処理を一つずつデータ処理モジュールとして実装した. 各データ処理モジュールを割り振ったノードを図 10 のように接続して 9 視点のシステムを構築した. このシステムでは, 2.3 節で説明したパイプライン並列処理とデータ並列処理を適用することで, スループットを向上させている. 形状復元を行なった計測空間の情報を表 1 に示す. 実際に運用してみると, 毎秒 15 フレームで動作した (図 11, 12).

6 考察

5 節において, 実アプリケーションの例によって, 実際に RPV を利用して実時間並列画像処理システムを PC クラスタ上に作ることが出来ることが示せた. また, データフローと例外処理の定義, 及び画像処理の実装を行なうだけで PC クラスタ上で動作するアプリケーションを構築出来た. よって, RPV は PC クラスタを利用した実

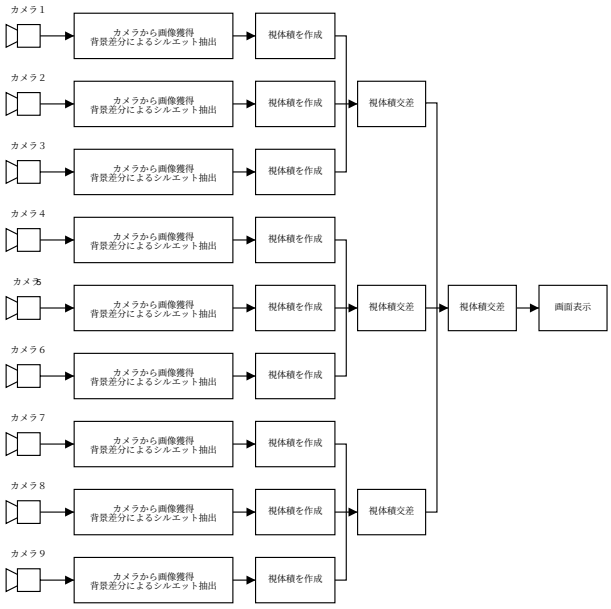


図 10: 3 次元形状復元システムの構成

時間並列画像処理プログラミングを容易にするという役目を果たすことが出来たと言える.

7　おわりに

本研究では, 多視点情報を計算機で処理することを想定し, 分散並列計算機の一種である PC クラスタを利用した実時間画像処理システムを構築するために画像処理以外に必要な機能を挙げた. そして, それらの機能を提供することで容易に実時間並列画像処理システムを構築するためのプログラミングツール RPV を提案した. さらに, 実アプリケーションの実装を行ない, その有用性を示した.

謝辞

本研究は，通信・放送機構による「次世代インテリジェント・マルチメディア情報通信網の基盤技術に関する研究」(No.10080)，および，文部科学省科学研究費補助金基盤研究 (B)「超分散センサーネットワークによる実時間高精細多メディア統合システムの研究」(No.14380167) の補助を受けた.

図 11: 入力画像例

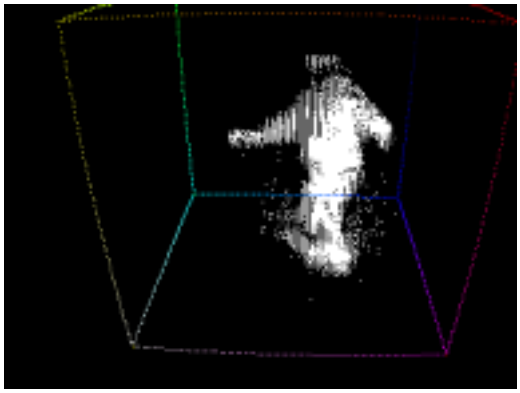


図 12: 復元した 3 次元形状を仮想視点から見た映像

参考文献

[1] 松山, 浅田, 美濃, 和田: “分散協調視覚プロジェクトー分散協調視覚研究, システム開発の概要ー”. 情処研報 CVIM103-4, pp. 25-34, 1997.

[2] 松山隆司: “分散協調視覚ー視覚・行動・コミュニケーション機能の統合による知能の創発”. 画像の認識・理解シンポジウム, 分冊 I, pp. 343-352, 1998.

[3] 1394 Trade Association. *1394-based Digital Camera Specification Version 1.20*.

[4] 吉本廣雅, 有田大作, 谷口倫一郎: “1394 カメラを利用した多視点動画像獲得環境”. 第 6 回 画像センシングシンポジウム講演論文集, pp. 285-290, 2000.

[5] *Myricom. Inc.* http://www.myricom.com/.

[6] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato: “Pm: An operating system coordinated high performance communication library”. In P. Sloot and B. Hertzberger, editors, *High-Performance Computing and Networking*, pp. 708-717. 1225 of Lecture Notes in Computer Science, Springer-Verlab, 1997.

[7] Bil Lewis, Daniel J. Berg, 岩本信一 (訳): “P スレッド プログラミング”. ISBN4-89471-097-8, 株式会社プレンティスホール出版, 1999.

[8] W. N. Martin and J. K. Aggarwal: “Volumetric description of objects from multiple view”. *IEEE Trans. PAMI*, Vol. 5, No. 2, pp. 150-159, 1987.