

ネットワークカメラアレイを用いた実時間全焦点自由視点映像合成システム

田口 裕一[†] 高橋 桂太[†] 苗村 健[†]

[†] 東京大学大学院情報理工学系研究科
〒113-8656 東京都文京区本郷 7-3-1

E-mail: †{yuichi,keita,naemura}@hc.ic.i.u-tokyo.ac.jp

あらまし 64 台のネットワークカメラアレイを用いた実時間全焦点自由視点映像合成システムについて報告する。本システムでは、所望の視点位置における画素単位の奥行きマップを実時間で推定し、高品質な全焦点自由視点映像を合成する。画像合成処理をすべて GPU 上で実装することにより、CPU と GPU で効率的な並列計算を行うシステムを 1 台の PC で実現した。これにより、解像度 QVGA の入力多視点映像を用いて、フレームレート 24 fps での自由視点映像合成が可能となった。さまざまなシーンにおける高品質な自由視点映像を実験結果として示す。

キーワード 実時間自由視点映像合成, カメラアレイ, 奥行き推定, GPGPU

Real-Time All-in-Focus Video-Based Rendering Using a Network Camera Array

Yuichi TAGUCHI[†], Keita TAKAHASHI[†], and Takeshi NAEMURA[†]

[†] Graduate School of Information Science and Technology, The University of Tokyo
7-3-1, Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan
E-mail: †{yuichi,keita,naemura}@hc.ic.i.u-tokyo.ac.jp

Abstract We present a real-time video-based rendering system using a network camera array. Our system consists of 64 commodity network cameras which are connected to a single PC through Gbit Ethernet. To render a high-quality novel view, we estimate a view-dependent per-pixel depth map in real-time based on a layered representation. The rendering algorithm is fully implemented on GPU, which allows our system to efficiently perform independent and parallel processing of CPU and GPU. Our system renders free-viewpoint videos at 24 fps using QVGA input video resolution. Experimental results show high-quality images synthesized from various scenes.

Key words Real-time video-based rendering, Camera array, Depth estimation, GPGPU

1. はじめに

Image-based rendering とは、あるシーンを複数の視点位置から撮影した多視点画像を入力として、任意の視点位置における写実的な画像を合成する技術である [1]。これは、3 次元空間の視覚情報を光線として記述する枠組みに基づいた技術であり、7 次元で光線を表現する Plenoptic function [2] に始まり、ある平面を通過する光線群を 4 次元で表現する光線空間 [3] や light field [4], [5] などと呼ばれる記述体系が議論されてきた。初期の研究では静止したシーンが対象となっていたが、近年では、動空間の光線を取得し自由視点映像合成を行うため、複数のビデ

オカメラを用いたシステムと、それに伴う Video-based rendering [6] の技術が検討されるようになった。

本稿では、ネットワークカメラアレイを用いた実時間自由視点映像合成システムについて報告する。図 1 に示すように、本システムでは、64 台のネットワークカメラが 8×8 の 2 次元アレイ状に配置され、すべてのカメラはギガビットイーサネットを通して 1 台の PC に接続されている。これらはキャストで移動可能なフレーム上に構築されており、簡易に移動とセットアップができるため、さまざまなシーンの撮影を行うことが可能である。高品質な合成映像を得るため、本システムでは、奥行きレイヤモデルに基づき、視点位置に応じた画素単位

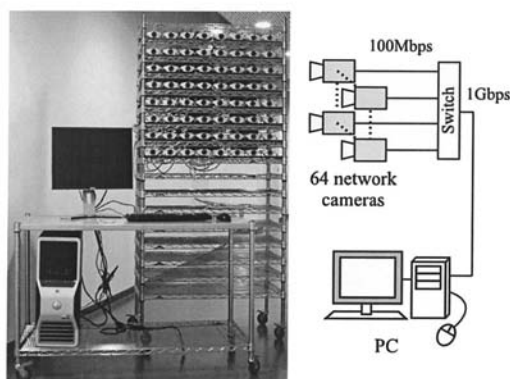


図1 カメラアレイシステムの概要。
Fig.1 Our camera array system.

の奥行きマップを実時間で推定する。画像合成のアルゴリズムは、GPGPU (General-Purpose computation on GPUs) の手法を利用してすべて GPU 上に実装される。このような GPU 上での実装には、以下のような2つの利点がある。1) GPU は同一の命令を画素ごとに並列に実行する処理に優れており、我々の画像合成アルゴリズムを効率的に実行することが可能となる。2) 画像合成以外の処理を CPU で実行でき、GPU と CPU を並列かつ独立に利用した効率的なシステム実装が可能となる。このような特徴を利用して、本システムでは、解像度 QVGA (320×240 ピクセル) の 64 視点分の映像を入力として、フレームレート 24 fps (frames per second) での自由視点映像合成が可能となった。

本論文の構成は以下の通りである。2. では、関連研究について述べ、本システムとの比較を行う。3. では、実時間での奥行き推定と自由視点映像合成手法について説明し、4. でシステム実装について述べる。5. では実験結果を示し、最後に 6. でまとめを述べる。

2. 関連研究

本節では、関連するカメラアレイシステムと画像合成手法について概観し、提案システムとの比較を行う。初期のカメラアレイシステムの代表例として、Kanade らが提唱した Virtualized Reality における 3D Dome がある [7]。このシステムでは、51 個のカメラをドーム状に並べ、その内部の空間を撮影する。このようなカメラ配置は、全方向からの映像をもとに物体の 3 次元モデルを構築するような用途として主に利用される。一方、提案システムを含む以下で説明するシステムは、シーン全体を記録・再生するような用途を想定している。Zitnick らは、8 台のカメラをアーク状に並べたシステムを構築した [8]。

カメラの個体差を考慮したセグメンテーションベースのステレオと、オブジェクトの境界において Matting を利用することにより、高品質な画像合成を実現した。Wilburn らは、100 台の同期可能なカスタムビデオカメラを利用して、さまざまな形状のアレイを構築した [9], [10]。カメラの撮影タイミングを少しずつずらすことにより、空間的および時間的に最適な光線のサンプリング手法と、そのようなデータからの画像補間手法を提案した。また、高性能カメラを模擬するさまざまな撮影法を開発した。谷本らは、100 台の HD カメラアレイを構築し [11]、Dynamic Programming を用いた奥行き推定に基づく画像合成手法を提案した [12]。本段落で紹介したシステムは、大容量の多視点映像データを扱っているため、あるいは、計算コストの高い幾何モデル推定手法を利用しているため、撮影後のデータを一度オフラインで処理したのち、画像合成のみを実時間でインタラクティブに行う。

上記のようなシステムに対して、本稿の提案システムのように、多視点画像の撮影から自由視点画像の合成までをすべて実時間で行うシステムもこれまでに提案されている。我々のグループでは、16 眼のカメラアレイに奥行き推定を行う専用ハードウェアを組み合わせた実時間システムを構築した [13]。推定された奥行きマップを 3 つのレイヤで近似することにより、10 fps での画像合成が報告されている。Yang らは、64 台の FireWire カメラを利用し、18 fps での画像合成を実現した (カメラの撮影速度は 15 fps である) [14]。画像合成の際、彼らはシーンの幾何モデルを 1 枚の平面 (focal plane) で近似しているため、その平面付近の物体は鮮明に合成されるが (in-focus)、それ以外の物体はぼけや 2 重像を伴って合成される [15]。Zhang と Chen は、48 台のネットワークカメラをそれぞれバンおよび横方向に移動可能な台の上に設置し、シーンの複雑度に応じてカメラの位置を適応的に制御可能な Self-reconfigurable camera array を構築した [16]。彼らは多重解像度を持つ 2 次元メッシュの頂点のみで奥行きを推定し、4–10 fps での画像合成を行っている。一方、本稿の提案システムでは、画素単位の奥行きマップを推定することにより、より良い合成画像品質を実現する。

Yang らは、GPU の機能を利用して奥行き推定と画像合成を高速化する手法を提案した [17]。彼らは 5 つのカメラを入力として利用し、15 fps で実時間画像合成を行った。我々の提案手法では、Yang らと同様の高速化に加え、より新しい GPGPU の手法を用いて、より大規模な 64 台のカメラアレイを対象とした実時間画像合成を行う。最後に、我々のグループでは、合焦判定法に基づく自由視点画像合成手法を提案している [18], [19]。これは、奥

行きレイヤモデルに基づく手法であり、レイヤごとに合成された画像から焦点の合っている (in-focus) 領域を抽出し、すべての領域に焦点の合う (all-in-focus) 画像を合成する。本システムの画像合成法はこの手法に基づいているが、さらに時間軸の奥行き連続性を考慮して奥行き推定を安定化し、また、画像合成におけるすべての処理を GPU 上で実行する。また、文献[18]では、静止多視点画像 81 枚を入力として 13.9 fps での画像合成を報告したが、本稿では、カメラアレイからの実時間ビデオ入力を対象としたシステム実装について述べる。

3. 実時間自由視点画像合成手法

本システムで用いる自由視点画像合成手法は、2 次元アレイ状に並べられたカメラによる多視点画像を入力として想定する。また、各カメラは事前にキャリブレーションされているとする。このような入力データから自由視点画像を合成する場合、光線の色を正しく補間するため幾何モデルの推定を行う必要がある。我々の手法は、奥行きレイヤモデルに基づき、画像を合成する視点位置に依存した画素単位の奥行きマップを実時間で推定する。本節では、まず初めに、多視点入力画像からの視点依存奥行きマップの推定手法と、それに基づく光線の補間方法について説明する。次に、時間方向の奥行き連続性を考慮した、奥行きを安定に推定するための制約について示す。本手法は、合成画像の各画素に対して独立な処理を行うため、4.4 で述べるように、GPU 上で効率的に実装可能である。

3.1 レイヤモデルに基づく奥行き推定と色の補間

本手法では、図 2 に示すような奥行きレイヤモデルを利用する。各レイヤは入力カメラ面とおおよそ平行になるよう配置される。被写体空間の奥行きの最大値を z_{max} 、最小値を z_{min} とした場合、 N 枚の奥行きレイヤ $z = \{z_n | n = 1, 2, \dots, N\}$ は以下のように配置される。

$$\frac{1}{z_n} = \frac{1}{z_{max}} + \frac{n-1/2}{N} \left(\frac{1}{z_{min}} - \frac{1}{z_{max}} \right) \quad (1)$$

これは、視差空間を等間隔に分割していることに相当する。2 次元正格子状のカメラ配置を対象とした場合、カメラ間隔と補間に必要なレイヤ枚数の関係は、Chai らにより議論されている [20]。

我々の手法は、画像を合成する視点位置における画素ごとの奥行きを推定する。これは、図 2 において、すべての target light ray ($\mathbf{r}(\mathbf{x})$) に対して最適な奥行き z を割り当てることに相当する。ここでは、文献[16],[18]と同様に、target light ray に近接する k 台のカメラ (参照カメラ) における reference light ray ($\mathbf{r}_i(\mathbf{x}, z)$) を参照し、color consistency コストを評価する。この reference light ray は、各奥行きレイヤと target light ray の交点

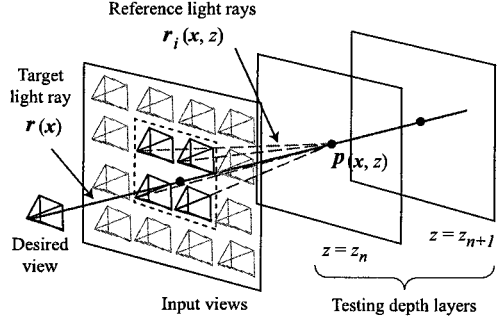


図 2 奥行きレイヤモデル。

Fig. 2 Layered depth model for synthesizing a desired view.

($p(\mathbf{x}, z)$) を参照カメラ i に射影することにより求められる。このように近接するカメラのみを参照するのは、オクルージョンの影響を抑えるため、また、計算コストをカメラの台数によらず一定に保つためである。以上より、color consistency コストは以下の式で与えられる。

$$C(\mathbf{x}, z) = \text{consistency}(I(\mathbf{r}_i(\mathbf{x}, z))_{i \in V}) \quad (2)$$

ここで、 V は参照カメラのインデックスのセット、 $I(\cdot)$ は光線の色を示す。今回の実装では、color consistency コストとして、reference light ray の RGB 各色の分散の和を利用した。また、図 2 に示されるように、 $|V| = k = 4$ とした。

次に、求められた color consistency コストを、奥行きレイヤごとに平滑化する。これは、画素ごとに求められた評価値は各種ノイズの影響により不安定になりやすいためである。本手法では、以下に示す基本的なブロックフィルタを利用する。

$$\bar{C}(\mathbf{x}, z) = \frac{1}{|W|} \sum_{\mathbf{x}' \in W} C(\mathbf{x}', z) \quad (3)$$

ここで、 W は画素 $\mathbf{x}$ を中心とした矩形領域である。実験においては、 11×11 画素の平均をとるブロックフィルタを利用した。

最後に、平滑化されたコストの最小値探索を行い、各 target light ray に対して最適な奥行き $z_{opt}(\mathbf{x})$ を割り当てる。

$$z_{opt}(\mathbf{x}) = \arg \min_z \bar{C}(\mathbf{x}, z) \quad (4)$$

また、target light ray の色の補間は、最適な奥行きにおける reference light ray の色の双線形補間により、以下のように求める。

$$I(\mathbf{r}(\mathbf{x})) = \sum_{i \in V} w_i(\mathbf{x}) I(\mathbf{r}_i(\mathbf{x}, z_{opt}(\mathbf{x}))) \quad (5)$$

ここで $w_i(\mathbf{x})$ は、参照カメラ i の reference light ray に対する重みであり、target light ray と参照カメラの位置に応じて 0 から 1 の値をとる。つまり、target light ray が参照カメラ i の位置を通過する場合 $w_i(\mathbf{x}) = 1$ 、その近隣のカメラの位置を通過する場合 $w_i(\mathbf{x}) = 0$ となるような線形に変化する。また、 $\sum_{i \in V} w_i(\mathbf{x}) = 1$ である。

上記の合成手法において、参照カメラセット V は、target light ray が入力カメラ面を通過する位置に応じて異なる。このため、1 枚の画像合成のために何台の入力カメラが利用されるかは、視点位置により変化する。しかし、我々の手法では、各 target light ray に対してコストと補間色を求めるため、入力カメラの台数によらず計算コストは常に一定である。計算コストは、合成画像の解像度 (target light ray の本数) と奥行きレイヤ数の積により決定される。

また、本合成手法は、特にテクスチャがない領域において、正しくない奥行きの値を最適値として割り当てる可能性がある。しかし、色の分散を最小化しているため、視覚的には正しい補間を行うことが可能である (テクスチャがない領域では、どの奥行きに対しても補間される色は類似している)。また、本合成手法では、色の補間に双線形補間を利用しているため、カメラの個体差の影響を抑えるような画像合成を行うことが可能である。

3.2 時間軸の奥行きの連続性を考慮した制約

上記の手法によりそれぞれの画像合成時に独立に奥行きマップを求める場合、カメラのノイズなどの影響により、特にテクスチャがない領域において奥行き推定が不安定となる。このため、合成された映像上でフリッカノイズが見られることがあった。この問題を抑えるため、時間軸の奥行きの連続性を利用し奥行きマップを安定に推定する。1 つ前のフレームで推定された画素ごとの奥行きマップを $z_{opt}^{t-1}(\mathbf{x})$ とした場合、以下の式によりこの制約を与える。

$$\hat{C}(\mathbf{x}, z) = \begin{cases} \tilde{C}(\mathbf{x}, z) & \text{if } z^t(\mathbf{x}) = z_{opt}^{t-1}(\mathbf{x}) \\ \tilde{C}(\mathbf{x}, z) + \lambda & \text{otherwise} \end{cases} \quad (6)$$

ここで、 λ は小さな正の定数である。式 (4) のかわりに、このコスト $\hat{C}(\mathbf{x}, z)$ に対して最小値探索を行い奥行きを決定する。この制約は、推定される奥行きは 1 つ前に推定された奥行きに類似するはずであると仮定している。さらに、 $z_{opt}^{t-1}(\mathbf{x})$ 以外の z に対してはすべて同一のコスト λ が加えられるため、対象の動きによる奥行きの急激な変化があった場合にも対応できる。

一般に、ユーザがインタラクティブに視点を動かして画像合成を行う場合、その視点位置は滑らかに動くと仮定できる。そのような滑らかな視点移動に対しても、奥行きは連続的に変化するため、上記の制約を利用するこ

とができる。一方、視点位置が急激に変化するようなアプリケーションの場合にはこの制約は無視される。

4. 実装

4.1 システム構成

カメラアレイシステムの構築には、64 台の Axis 210 ネットワークカメラを用いた (図 1)。カメラは 8×8 の 2 次元アレイ状に配置した。カメラ間の距離は上下左右とも 100 mm 程度である。このネットワークカメラは、最大で解像度 640×480 の動画を 30fps で撮影することができる。また、HTTP サーバ機能を有しており、PC からのリクエストに応じて撮影した動画を Motion JPEG として送信する。JPEG の圧縮率はリクエストで指定することができ、0 (最高品質) から 100 (最低品質) までの値をとる。今回の実験においては、撮影の解像度を 320×240 、JPEG 圧縮率を 10 に設定した。

各カメラは 100 Mbps のイーサネットポートを備えている。すべてのカメラはギガビットイーサネットスイッチに接続され、さらにそこから 1 台の PC へと接続される。カメラアレイから出るケーブルは、イーサネットケーブル 1 本と電源ケーブル 1 本のみにとめられているため、システム全体を簡易に移動・セットアップすることが可能である。PC には、2 個の Intel Xeon 5160 (3 GHz) CPU、3 GB のメインメモリ、また、グラフィックスカードには、NVIDIA GeForce 8800 Ultra GPU、768 MB のビデオメモリが搭載されている。

4.2 カメラキャリブレーション

システムのセットアップ時には、カメラの幾何キャリブレーションを行う必要がある。一般に利用される Zhang の手法 [21] は、キャリブレーションボードを置いた周辺の奥行きでの精度は高いが、それ以外の奥行きでは精度が低くなることが観察された。(文献 [22] でもこのことが指摘されている)。広い撮影範囲を想定する我々のカメラアレイでは、Tsai の手法 [23] の精度がより高かったため、これにより幾何キャリブレーションを行った。具体的には、チェッカーボードを数点の既知の位置に置き、抽出した特徴点からカメラパラメータを計算した。また、我々はこれまでに、システムの移動に伴い数台のカメラの位置のみがずれてしまった場合に、その周囲のずれていないカメラを参照してインタラクティブにキャリブレーションを行う手法も提案している [24]。

今回の実験においては、カメラの色キャリブレーションは行わず、各カメラに対して自動ホワイトバランスと露光制御の設定を行った。これは、今回用いたカメラでは、撮影パラメータの詳細な設定ができないこと、また、レンズぼけなどによる画像のばらつきも大きいため、精密な色合わせの効果が見込まれなかったためである。こ

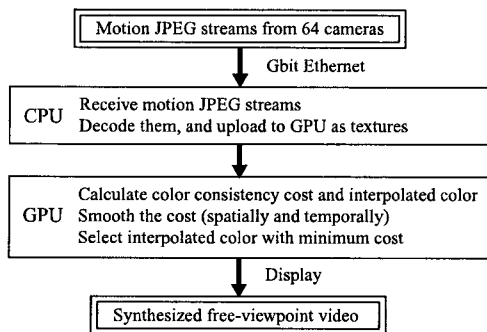


図3 ソフトウェア構成とデータの流れ。

Fig.3 Software architecture and data flow of our system.

のような個体差の大きい入力多視点画像に対しても、提案した画像合成手法は視覚的に良好な画像を合成できることを5.で示す。

4.3 多視点映像データの取得

本システムのソフトウェア構成とデータの流れを図3に示す。64台のカメラからMotion JPEGを受信するネットワークI/O処理とJPEGのデコードは、CPU上で並列に行われる。デコードされた画像はGPUのメモリにテクスチャとして伝送される。以降の画像合成処理はすべてGPU上で実行され、GPUからCPUへのデータ伝送はない。このため、本システムではCPUとGPUを並列かつ独立に利用した、効率的な処理が可能となる。

今回用いたカメラは、撮影タイミングの同期機能を持っていないため、画像合成時には、各カメラから受信される最も新しい画像を利用するアプローチをとる。現在のところ、3.1で議論したように最小の分散を持つ色で補間を行うこと、また、5.で示されるように比較的高速なフレームレートの撮影を行っていることにより、このアプローチにより視覚的に良好な品質を達成できる。

4.4 GPU上での画像合成

GPU上での画像合成アルゴリズムの実装には、OpenGLと、Cg (C for graphics) [25] によるフラグメントプログラムを利用した。画像合成時には、まず最初に各奥行きレイヤにおけるcolor consistencyコストの計算(式(2))と色の補間(式(5))を行う。この計算には、文献[17], [19]と同様に、射影テクスチャマッピング(projective texture mapping)と、フラグメントプログラムによる画素ごとの計算を利用する。このフラグメントプログラムは、コスト計算と色の補間を同時に行い、補間色をRGBチャンネルに、コストをアルファチャンネルに出力する。文献[17], [19]では、色の補間をreference light rayの色の平均として求めていたが、本手法では双線形補間を行うことにより合成品質を向上させる。式(5)にお

ける重み $w_i(\mathbf{x})$ は、重みの値を指定する別のテクスチャを同時にマッピングすることによりフラグメントプログラムに与えられる。GPU上では、計算結果は0から1の間におさまるようにクリッピングされるため、一定の値以上のコストは自動的に丸められる。これにより、以降の平滑化のステップにおいて、ノイズや外れ値の影響を抑える効果が得られる。

これ以降の処理は、2次元画素配列のすべての画素に対して同一の処理を行う、一般的なGPGPU処理である。これは、OpenGLの変換行列を2次元正射影とし、処理したいデータのテクスチャマッピングを繰り返すことにより実行する。まず、求められた各奥行きレイヤのコストの、空間的および時間的な平滑化を行う(式(3)および(6))。ここでは、フラグメントプログラムにより2次元配列上で近隣のコストの平均を求め、さらに、1フレーム前に推定された奥行きと現在対象としている奥行きが異なる場合にコストを加える処理を行う。次に、最小のコストを持つ奥行きの値(式(4))、およびそれに対応する補間色を求める。現在対象としているレイヤよりも以前に計算された最適な奥行きと補間色は、現在の最小コストとともにそれぞれテクスチャに保存されており、対象レイヤで計算された値を比較することで順次これを更新する。すべての奥行きレイヤに対する処理ののちに、最小コストを持つ奥行きと合成画像が得られる。以上の処理の疑似コードを付録に記載する。

5. 実験結果

入力画像の解像度を320×240ピクセル、出力合成画像の解像度を300×300ピクセル、奥行きレイヤの枚数を15とした場合、本システムではフレームレート24fpsでの自由視点映像合成が達成できた。JPEGの圧縮率を10とした場合、ネットワーク帯域の総量は200–240 Mbpsであった。本カメラアレイシステムは簡易に移動とセットアップを行えるため、以下で示すような様々なシーンの撮影を行うことができた。

図4に、Meeting roomシーンにおける画像例を示す。我々の奥行き推定手法により、すべての領域が鮮明な全焦点画像が合成されることがわかる。3.1で議論したように、テクスチャのない領域では奥行き推定が間違っていることがあるが、視覚的に正しい色が補間される。また、図4(a)に示されるように、カメラの個体差により、入力画像の色は大きなばらつきを持っていることがわかる。図5に、補間方法の違いによる合成画像例を示す。図5(a)は文献[17], [19]のような平均補間による合成画像、図5(b)は双線形補間による合成画像、また、図5(c)は、合成画像中で各入力カメラ画像がどの部分に寄与しているか

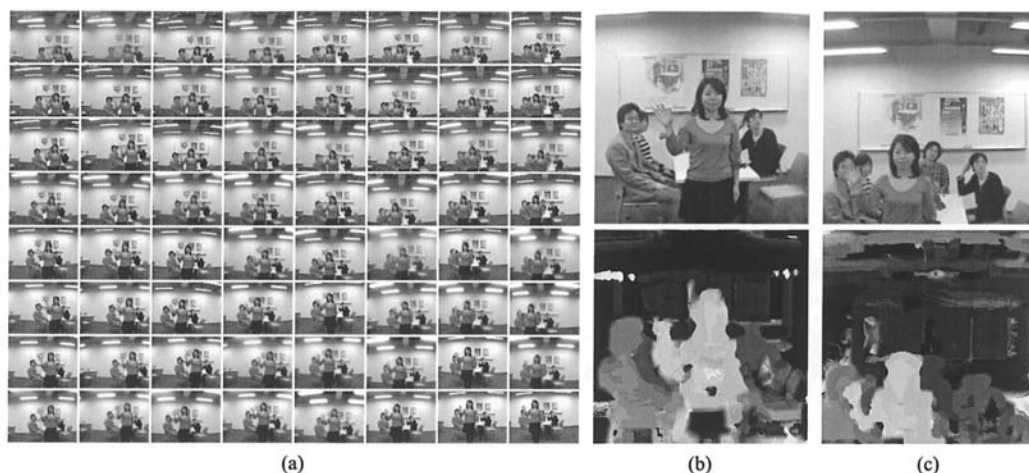


図 4 *Meeting room* の画像例. (a) 入力 64 視点画像. (b, c) 異なる視点位置における合成画像と対応する奥行きマップ.
Fig. 4 Example images from *Meeting room* sequence. (a) Input 64 images. (b, c) Output synthesized images and their corresponding depth maps from different viewpoints.

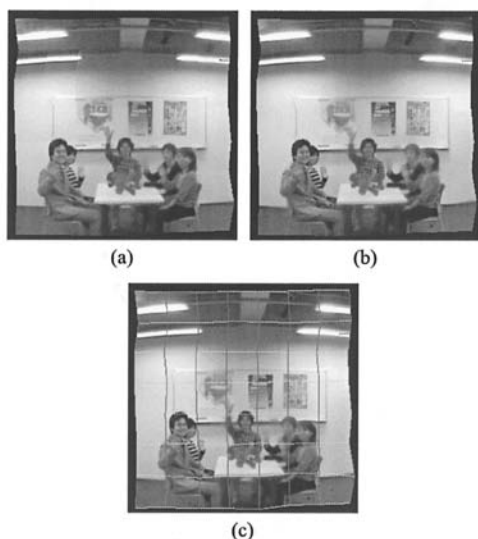


図 5 色補間手法の比較. (a) 平均補間, (b) 双線形補間による合成画像. (c) カメラグリッド.
Fig. 5 Comparison of color interpolation method. Synthesized images using (a) average interpolation and (b) bilinear interpolation. (c) Camera grid.

を示すため、カメラグリッドを表示したものである。このように、平均補間では参照するカメラが切り替わる位置で合成画像の色が大きく変化してしまうことがわかる。一方、提案手法が利用する双線形補間では、色の切り替わりが滑らかで自然な画像が合成できる。

図 6 に動きの速いシーンに対する合成画像例を示す。

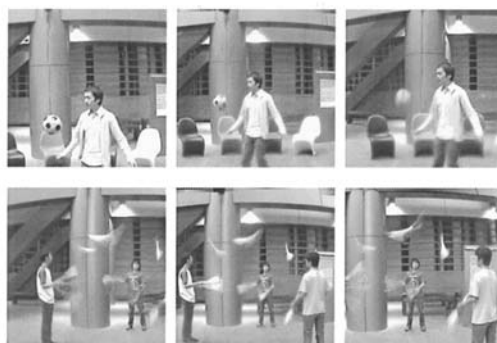


図 6 動きの速いシーン (*Soccer* (上段), *Juggling* (下段)) に対する合成画像例.
Fig. 6 Synthesized images from *Soccer* sequence (top row) and *Juggling* sequence (bottom row).

本カメラアレイシステムは同期機能を持たないため、画像合成時に利用される入力画像は異なるタイミングで撮影されている可能性がある。このため、合成される映像から各フレームを切り出して見た場合、動きの速い部分における合成画像上でのぼけや 2 重像が確認される。しかしながら、自由視点映像として合成結果を見た場合には、これらは視覚的に許容範囲にあるといえる。

本システムにより合成された自由視点ビデオ、および時間軸の連続性を考慮したコストを利用することによる奥行き推定の安定化を示すビデオは、下記の URL から閲覧可能である。

<http://www.hc.ic.i.u-tokyo.ac.jp/project/camera-array/>

6. ま と め

本稿では、64 台のネットワークカメラアレイを用いた実時間自由視点映像合成システムの報告を行った。提案システムでは、画素単位の奥行き推定を実時間で行うことにより、高品質な全焦点画像を合成する。画像合成アルゴリズムはすべて GPU 上で実装されているため、CPU と GPU の効率的な並列計算を行うことができ、1 台の PC でフレームレート 24 fps での自由視点映像合成が達成された。また、本システムは簡易に移動・セットアップ可能であるため、さまざまなシーンの多視点映像を撮影することができた。カメラの個体差による入力画像の色のばらつきや、同期機能がないために生じる撮影タイミングのずれ問題があるが、提案する画像合成手法では視覚的に良好な画像合成を行うことができた。現在は、カメラアレイとインテグラルフォトグラフィ方式の裸眼立体視 3 次元ディスプレイを利用して、実時間 3 次元映像伝送を行うシステムの開発に取り組んでいる。

謝辞 本研究に関して有益なご助言をいただいた原島博教授、カメラアレイシステムの構築に協力いただいた王金戈氏に感謝致します。また、ジャグリングの撮影に協力いただいた一野史也氏、木田巧氏、橋本英樹氏に謝申し上げます。

文 献

- [1] H.-Y. Shum, S. B. Kang, and S.-C. Chan: "Survey of image-based representations and compression techniques," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 11, pp. 1020-1037, Nov. 2003.
- [2] E. H. Adelson and J. R. Bergen: "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing* (Eds. by M. Landy and J. A. Movshon), MIT Press, pp. 3-20, 1991.
- [3] 藤井俊彰, 金子正秀, 原島博: "光線群による 3 次元空間情報の表現とその応用", *テレビ誌*, vol. 50, no. 9, pp. 1312-1318, Sept. 1996.
- [4] M. Levoy and P. Hanrahan: "Light field rendering," in *Proc. ACM SIGGRAPH'96*, pp. 31-42, Aug. 1996.
- [5] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen: "The lumigraph," in *Proc. ACM SIGGRAPH'96*, pp. 43-54, Aug. 1996.
- [6] 苗村健, 原島博: "Video-Based Rendering の基礎検討", 3 次元画像コンファレンス'98, pp. 165-170, July 1998.
- [7] T. Kanade, P. Rander, and P. J. Narayanan: "Virtualized reality: Constructing virtual worlds from real scenes," *IEEE Multimedia*, vol. 4, no. 1, pp. 34-47, Jan. 1997.
- [8] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, and R. Szeliski: "High-quality video view interpolation using a layered representation," in *Proc. ACM SIGGRAPH 2004*, pp. 600-608, Aug. 2004.
- [9] B. Wilburn, M. Smulski, H.-H. K. Lee, and M. A. Horowitz: "The light field video camera," in *Proc. SPIE Media Processors 2002*, vol. 4674, pp. 29-36, Jan. 2002.
- [10] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy: "High performance imaging using large camera arrays," in *Proc. ACM SIGGRAPH 2005*, pp. 765-776, July 2005.
- [11] M. Tanimoto: "FTV (free viewpoint television) creating ray-based image engineering," in *Proc. IEEE Int. Conf. Image Processing (ICIP 2005)*, vol. II, pp. 25-28, Oct. 2005.
- [12] 福岡慶繁, 園道知博, 藤井俊彰, 谷本正幸: "Multi-Pass Dynamic Programming による光線空間補間", *信学論*, vol. J90-D, no. 7, pp. 1721-1725, July 2007.
- [13] T. Naemura, J. Tago, and H. Harashima: "Real-time video-based modeling and rendering of 3D scenes," *IEEE Comput. Graph. Appl.*, vol. 22, no. 2, pp. 66-73, Mar. 2002.
- [14] J. C. Yang, M. Everett, C. Buehler, and L. McMillan: "A real-time distributed light field camera," in *Proc. 13th Eurographics Workshop on Rendering*, pp. 77-85, June 2002.
- [15] A. Isaksen, L. McMillan, and S. J. Gortler: "Dynamically reparameterized light fields," in *Proc. ACM SIGGRAPH 2000*, pp. 297-306, July 2000.
- [16] C. Zhang and T. Chen: "A self-reconfigurable camera array," in *Proc. 15th Eurographics Symposium on Rendering*, pp. 243-254, June 2004.
- [17] R. Yang, G. Welch, and G. Bishop: "Real-time consensus-based scene reconstruction using commodity graphics hardware," in *Proc. Pacific Graphics 2002*, pp. 225-235, Oct. 2002.
- [18] K. Takahashi and T. Naemura: "Layered light-field rendering with focus measurement," *EURASIP Signal Processing: Image Commun.*, vol. 21, no. 6, pp. 519-530, July 2006.
- [19] 高橋桂太, 苗村健: "視点依存奥行きマップ実時間推定に基づく多眼画像からの自由視点画像合成", *映像学誌*, vol. 60, no. 10, pp. 1611-1622, Oct. 2006.
- [20] J.-X. Chai, X. Tong, S.-C. Chan, and H.-Y. Shum: "Plenoptic sampling," in *Proc. ACM SIGGRAPH 2000*, pp. 307-318, July 2000.
- [21] Z. Zhang: "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, no. 11, pp. 1330-1334, Nov. 2000.
- [22] V. Vaish, B. Wilburn, N. Joshi, and M. Levoy: "Using plane + parallax for calibrating dense camera arrays," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR 2004)*, vol. 1, pp. 2-9, June 2004.
- [23] R. Y. Tsai: "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *IEEE J. Robot. Automat.*, vol. 3, no. 4, pp. 323-344, Aug. 1987.
- [24] 王金戈, 田口裕一, 高橋桂太, 苗村健: "実時間自由視点画像合成のためのカメラアレイシステムの構築とキャリブレーション手法の検討", 3 次元画像コンファレンス 2007, pp. 97-100, July 2007.
- [25] <http://developer.nvidia.com/page/cg-main.html>.
- [26] http://www.opengl.org/registry/specs/EXT/framebuffer_object.txt.

```

// Allocate texture memory on GPU
Texture tLayer, tCurOptDepth, tCurOptImage, tPrevDepth;
Texture tInputImage [64];
Texture tBilinearWeight; // Init'd with weight values

// Start rendering a novel view
RenderNovelView (viewpoint) {
    for_each (inputCamIdx)
        UploadTexture (tInputImage [inputCamIdx]);

    for_each (depthLayer) {
        // Render a layer
        SetFragmentProg (CALC_COST_INTERP_COLOR);
        // Neighboring 4 cameras are used for the target light
        // rays passing within the positions of the cameras
        for_each (neighbor4Cameras) {
            for (idx = 0; idx = 4; ++idx) {
                camIdx = FindCamIdx (neighbor4Cameras, idx);
                SetMatrices (viewpoint, CalibParams [camIdx]);
                ProjTextureMap (tInputImage [camIdx]);
                // The weight texture is mapped with different
                // directions for each index
                ProjTextureMap (tBilinearWeight);
            }
        }
        ReadFrameToTexture (tLayer);

        // Set matrices to orthogonal, for applying the same
        // instructions for each pixel in the textures
        SetMatrices (Orthogonal2D);

        // Smooth the cost, and select the depth with the
        // minimum cost
        SetFragmentProg (SMOOTH_COST_RECON_DEPTH);
        TextureMap (tLayer, tCurOptDepth, tPrevDepth);
        ReadFrameToTexture (tCurOptDepth);
        // Store the estimated depth map for the next iteration
        if (last depth layer)
            ReadFrameToTexture (tPrevDepth);

        // Select the color with the minimum cost
        SetFragmentProg (COMPOSITE_LAYER);
        TextureMap (tLayer, tCurOptDepth, tCurOptImage);
        ReadFrameToTexture (tCurOptImage);
    }
}

```

図 A・1 画像合成アルゴリズムの擬似コード。

Fig. A・1 Pseudo code of our rendering algorithm.

付 録

画像合成のための OpenGL 命令の擬似コードを図 A・1 に、フラグメントプログラムの詳細を図 A・2 に示す。図 A・2 におけるテクスチャ名は図 A・1 で定義されたものに

```

CALC_COST_INTERP_COLOR {
    // Calculate color consistency cost
    cost = Sumc=r,g,b (Variancei (tInputImage[i].c));
    // Interpolate color
    color = Sumi (tInputImage[i].rgb * tBilinearWeight[i]);

    // Output the calculated values to (RGB, A) channels
    // of the frame buffer
    Output (color, cost);
}

SMOOTH_COST_RECON_DEPTH {
    // Smooth cost in the neighborhood window W
    smoothCost = Averagew (tLayer.a);

    // Temporal depth smoothing
    if (current depth value != tPrevDepth)
        smoothCost += λ;

    if (smoothCost < tCurOptDepth.a)
        Output (current depth value, smoothCost);
    else
        Output tCurOptDepth;
}

COMPOSITE_LAYER {
    if (tCurOptDepth.a < tCurOptImage.a)
        Output (tLayer.rgb, tCurOptDepth.a);
    else
        Output tCurOptImage;
}

```

図 A・2 フラグメントプログラムの擬似コード。

Fig. A・2 Pseudo code of the fragment programs.

準じているが、フラグメントプログラム内では出力画像の各画素に対応する RGBA のデータがこれらのテクスチャから読み込まれ、計算が行われる。擬似コード中でフレームバッファからテクスチャにデータをコピーする部分は、EXT_framebuffer_object [26] を利用してテクスチャに直接描画を行うことでも実現可能であるが、今回の実装環境においては大きな性能の変化は見られなかった。