# IBR に基づく仮想都市のリアルタイムレンダリングの GPU を用いた実装手法の提案

佐藤　亮† 小野晋太郎†† 川崎　洋† 池内　克史††

† 埼玉大学 〒 338-8570　埼玉県さいたま市桜区下大久保 255
†† 東京大学 〒 153-8505　東京都目黒区駒場 4-6-1
E-mail: †{satou,kawasaki}@cgv.ics.saitama-u.ac.jp, ††{onoshin,ki}@cvl.iis.u-tokyo.ac.jp

**あらまし**　都市のような広域空間をコンピュータ内に取り込み仮想空間を構築する手法として，実写画像ベースで描く Image-Based Rendering (IBR) がある．IBR は写実的な再現が行える半面，大量の画像データベースから必要なデータを選択し合成する手法であるため，データサイズが非常に大きくなるという問題がある．著者らは，これまで IBR に基づいた固有空間法による圧縮手法，および圧縮されたデータの実時間復元手法を提案してきた．前者は，時空間画像解析とブロックマッチングを用いたトラッキングにより，より少ない固有画像に圧縮する手法であり．後者は，固有画像と重み係数との積和演算による復元処理を GPU で行うことで，CPU に負荷をかけることなく高速なレンダリングを実現する手法となっている．本稿では，GPU による復元処理について詳しく述べる．

**キーワード**　IBR，全方位画像，圧縮，固有空間法，GPU

# GPU Implementation of Real-time Image Based Rendering Technique for Virtual City Modeling

Ryo SATO†, Shintaro ONO††, Hiroshi KAWASAKI†, and Katsushi IKEUCHI††

† Saitama University 255 Shimo-Okubo, Sakura-ku, Saitama City, Saitama 338-8570, JAPAN
†† Tokyo University 4-6-1 Komaba Meguro-Ku, Tokyo 153-8505, JAPAN
E-mail: †{satou,kawasaki}@cgv.ics.saitama-u.ac.jp, ††{onoshin,ki}@cvl.iis.u-tokyo.ac.jp

**Abstract**　Image-Based Rendering (IBR) is one of the technique for construction of a large scale virtual scene like a city into a computer. IBR methods are effective for photo-realistic rendering, but their huge data sets and restrictions on interactivity pose serious problems for an actual application. The authors have so far proposed a compression method based on IBR using eigen-space method. This can achieve the compression to fewer eigen images because the tracking of frames is done correctly using time-space analysis and blockmacthing technique. The authors have also proposed a method, real-time restoration of the compressed data. This can achieve high-speed rendering without raising a load of CPU because the restoration processing, product sum operation of eigen images and weight coefficient, is done on Graphic Processor Unit (GPU). In this paper, we describe the real-time restoration by GPU for a large scale scene.

**Key words**　Image-Based Rendering, Omni-directional image, Compression, Eigen-space method, GPU

## 1. はじめに

近年，都市空間を計算機内に仮想的に再現する研究が盛んに行われている．これを写実的に実現する手法の 1 つである IBR [3], [4] は，撮影した画像を直接利用するため，モデル化の難しい複雑な物体であっても容易に描画することが可能であるが，再現する空間すべての画像データを保持せねばならず，データの容量的な問題がある．

そこで著者らは，車載カメラで撮影した全方位画像列を対象とした固有空間法による圧縮方法，および任意画像の復元を GPU で処理することで，高速にレンダリングする手法を合わせて提案してきた [1]．本稿では，実際に GPU を用いてリアルタイムに広域空間を復元する手法について述べる．

2 章で [1] で提案するシステムの概要を述べ，3 章でレンダリ

ングにおける形状について，4 章で GPU による圧縮データの復元処理，5 章で実験について述べる．

## 2. 広域空間再現システム

### 2.1 概　　要

広域空間再現システムは，次の 3 つの処理から構成される．

(1) 全方位画像の撮影
(2) 全方位画像データの圧縮
(3) 圧縮データからの自由視点レンダリング

(1) は全方位カメラを車載して計測を行い，全方位画像列を得る．(2) はこのように得た全方位画像列が同じ物体をあらゆる方向から撮影した画像列であり，冗長性が高いという性質を利用して，画像列を圧縮する．

### 2.2 全方位画像の撮影

全方位画像列を獲得するために，図 1(a) のような全方位カメラ Ladybug2 [7] を計測車両に搭載し，走行しながら撮影を行う．等速で走行，あるいは GPS 等による補正により，撮影画像は等間隔に得られているとする．撮影された全方位画像の例を図 1(b) に示す．

(a)ladybug と計測車

(b) 全方位画像の例

(c) 圧縮対象画像

図 1　データ計測車および全方位画像

### 2.3 データの圧縮

冗長性を含んだ全方位画像を効率よく圧縮するため，まず画像列中から類似した部分を探索するためのトラッキングを行う．続いて，その部分に対する画像圧縮を行う．トラッキングおよび圧縮では，撮影車両の進行方向に対して平行な左右 2 平面に投影した画像（図 1(c)）を用いる．これは，トラッキングにおいてオブジェクトの推移が EPI 上で直線となって表れるため，速度の推定が直線検出に置き換えられ，トラッキングが容易になるためである．

#### 2.3.1 トラッキング

トラッキングは EPI（付録参照）によるグローバルなトラッキングとブロックマッチングによる詳細なローカルトラッキングを組み合わせた手法である．グローバルなトラッキングは，画像フレーム間の平均的な移動量を求めることに相当する．これは，図 1(c) の画像列から得た EPI においてハフ変換により直線エッジを検出し，複数ある直線から代表的な傾きを選択することで実現される．その後ローカルなトラッキングでは，1 フレームあたり $m \times n$ 個のブロックに対して前フレームのブロックと相関値を求めることでマッチングを行う．このトラッキングにより図 2 の太線枠のような類似したブロック画像群を得ることができる．

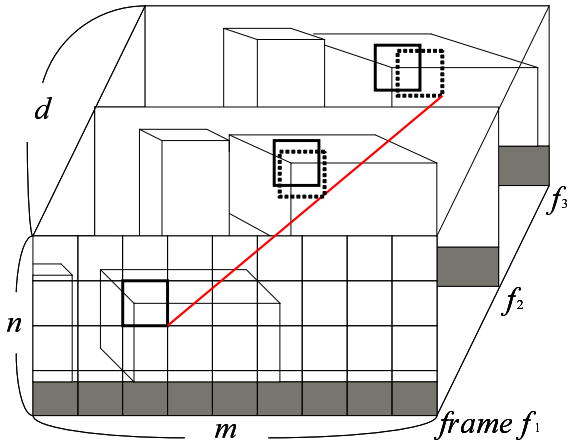



図 2　トラッキング

#### 2.3.2 固有空間法による圧縮

図 2 のように，それぞれのブロック画像に対し，第 $d$ フレームまで 2.3.1 節のトラッキングを適用し，類似ブロック画像集合を得る．このブロック画像集合に固有空間法を適用することで，$r$ 枚の固有画像に圧縮することができる．

### 2.4 自由視点レンダリング

#### 2.4.1 画像の復元

オリジナルの $i$ フレーム目の画像 $X_i$ は以下に示す線形和により復元することが出来る（式 (1)）．ここで，$V_k$ は，$k$ 番目の固有画像，$w_{ik}$ は，フレーム $i$ の重み係数である．また，$r$ は，ある累積寄与率達成するために必要な主成分の枚数である．

$$X_i = \sum_{k=1}^{r} (w_{ik} * V_k) \tag{1}$$

#### 2.4.2 レンダリングアルゴリズム

ここでは，圧縮したデータを用いたレンダリング手法について図 3 を用いて述べる．ある視点 P におけるレンダリングは，撮影経路の A，B，C において撮影された画像群の中から視点 P と壁面中心との成す角に対応するテクスチャをそれぞれ貼ることで実現される [3]．このとき撮影された画像群は圧縮され固有画像となっているので，例えば撮影地点 A におけるテクスチャは，ある累積寄与率を満たす固有画像 $r$ 枚と $\theta_1$ に対応するパラメータとの積和演算で復元することによって得られる

(式 (1))．$m \times n$ のブロック画像に分割しているので，レンダリングは $m \times n$ のブロック画像それぞれを全て復元する必要がある．また，積和演算処理を GPU で行うことで実時間復元が可能となる．

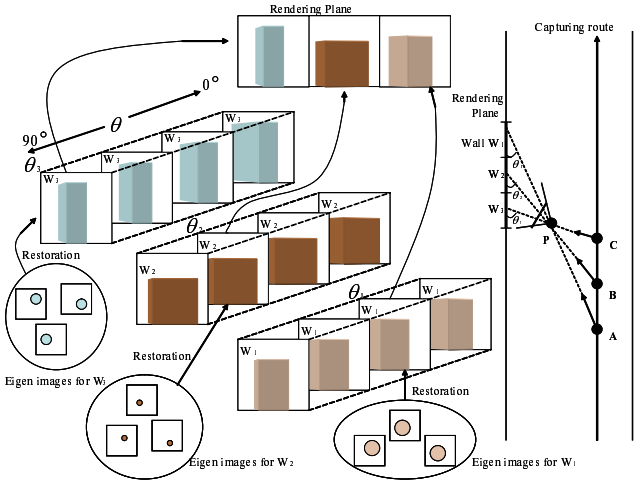



図 3　レンダリングアルゴリズム

## 3. リアルタイムレンダリングに適した形状データ

本手法では，ポリゴン集合に動的にテクスチャマップすることで，広域空間を再現している．本章では，ポリゴン形状と配置について述べる．

### 3.1 形　　状

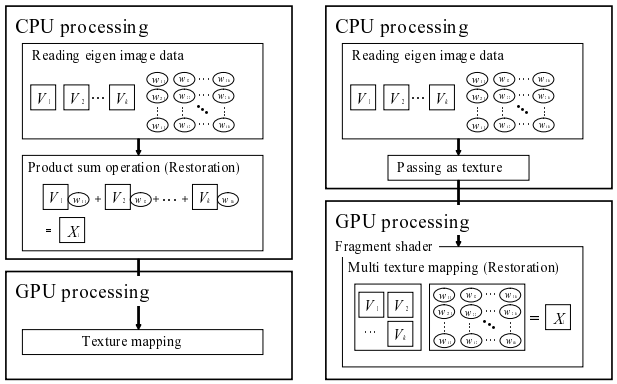
ポリゴンの形状は，図 3 における Wall のようにスリット状であり，図 2 における $n \times 1$ ブロックがそれに対するテクスチャである．このようにするのは，Wall ごとに板と視点との角度が違うため，4.2 節で述べるマルチテクスチャマッピングに都合が良いためである．またスリット形状にすることで，ビルボーディングを実装することが容易となり視覚依存の影響の少ないレンダリングが可能となる．

### 3.2 配　　置

都市空間においては，建物の正面は車道からおおよそ一定距離の平面に面していることが多い．テクスチャマップ用のスリット状ポリゴンは，この平面が存在する位置に置くと歪みが少なく都合が良い．この奥行きは，2.3.1 節のグローバルトラッキングで既に得られているためこれを利用することができる．すなわち，EPI 上で代表的な傾きを持ったエッジは建物正面に相当すると見なせるため，その傾きが表す奥行きを用いることができる．カメラの焦点距離を $f$，カメラの移動速度（車両の速度）を $V$，対象の画像上での 1 フレームあたり移動量を $\Delta u/\Delta z$ とすると，奥行き $d$ は，

$$d = \frac{fV}{\Delta u/\Delta z} \tag{2}$$

と求めることができる．(付録参照)




(a) CPU-based　　(b) GPU-based

図 4　画像復元プロセス

## 4. GPU 処理による圧縮データの復元

### 4.1 CPU 処理と GPU 処理の違い

原画像 $X_1$ を復元する手法を図 4 を用いて説明する．CPU ベースの手法では，はじめに圧縮されたデータ（固有画像 $V_k$，重み係数 $w_{ik}$）をメモリに読み込んでおき，$V_k$ と $w_{1k}$ との積和演算処理を CPU で行う（図 4(a)）．これにより復元された $X_1$ をテクスチャとして GPU に渡し，テクスチャマッピングすることになる．しかし，次に原画像 $X_2$ を復元する場合，重み係数を $w_{2k}$ とし同様の手順が必要となり，処理に時間がかかる．また，テクスチャサイズを $n \times n$ とすると計算量が $O(n^2)$ であるため，テクスチャのサイズが大きくなると CPU の負荷が急激に増加する．

一方，GPU ベースの手法は圧縮されたデータを読み込んだ後，これらをテクスチャとしてあらかじめ GPU に渡しておき，積和演算処理は GPU 内におけるピクセルシェーダのみで行われる（図 4(b)）．次に $X_2$ を復元する場合でも，あらかじめ保持してある $V_k$ と $w_{2k}$ との積和演算をすれば良く，テクスチャを改めて作り直す必要がない．これにより，CPU に負荷をかけることなく圧縮データの復元が可能となる．ピクセルシェーダは高速な並列プロセッサのため画像解像度程度のテクスチャサイズであれば $O(n^2)$ の処理を高速に実行でき，フレームレートでの処理が実現できる．これにより CPU に負荷をかけることなく原画像のリアルタイム復元が可能となる．

### 4.2 マルチテクスチャによる積和演算

提案手法では，GPU で行う固有画像と重み係数との積和演算処理を，ピクセルシェーダによるマルチテクスチャ処理で実現する．マルチテクスチャは，複数のテクスチャユニットの処理結果を合成することで実現されるが，テクスチャユニット数が重ね合わせることができる枚数でありグラフィックスハードウェアごとに異なっている．本節では，3 つ（固有画像，重み係数，平均画像）のテクスチャユニットを用いたレンダリングについて図 5 を用いて述べる．

3.1 節で述べたように本手法では，スリット状のポリゴンの集合体として広域空間を表現する．そこで，固有画像テクスチャの配置は，ある累積寄与率を満たす $r$ 枚の固有画像を $v$ 軸

方向に並べ，さらにそれを $n$ ブロック分を同じく $v$ 軸方向に並べた形となっている．これはテクスチャユニット数が多ければ，テクスチャユニットの数だけ固有画像テクスチャとすることができるものの，実際にはテクスチャユニット数に限りがあるため，こうすることで 1 つのテクスチャユニットで何枚もの固有画像を並べることで利用できるようにするためである．重み係数テクスチャの配置は，$v$ 軸方向は固有画像テクスチャと同様に並べ，$u$ 軸方向に $i$ フレーム分を並べた形となっている．平均画像テクスチャに関しては，$v$ 軸方向に $n$ ブロック分，$u$ 軸方向に $i$ フレーム分を並べた形である．このような 3 つのテクスチャをピクセルシェーダに渡すこととなる．

その後ピクセルシェーダにおいて，対象となるテクスチャ座標（図 5 の塗潰し部分）を計算し，マルチテクスチャ処理をすることで圧縮データの復元およびレンダリングが実現される．図 6 に各種テクスチャの例を示す．固有画像 5 枚，70 フレーム分のデータである．画像サイズは，固有画像テクスチャが（ブロック画像サイズ）×（ブロック画像サイズ×固有画像枚数×$n$ ブロック数），重み係数テクスチャが（フレーム数）×（固有画像枚数×$n$ ブロック数），平均画像テクスチャが（フレーム数）×（$n$ ブロック数）である．

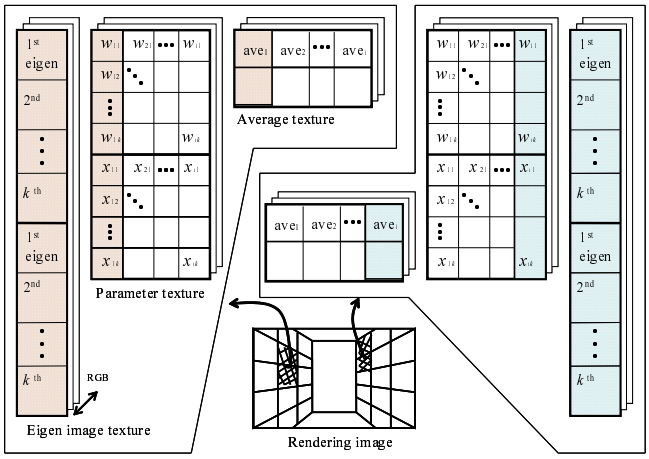



図 5　マルチテクスチャによる積和演算

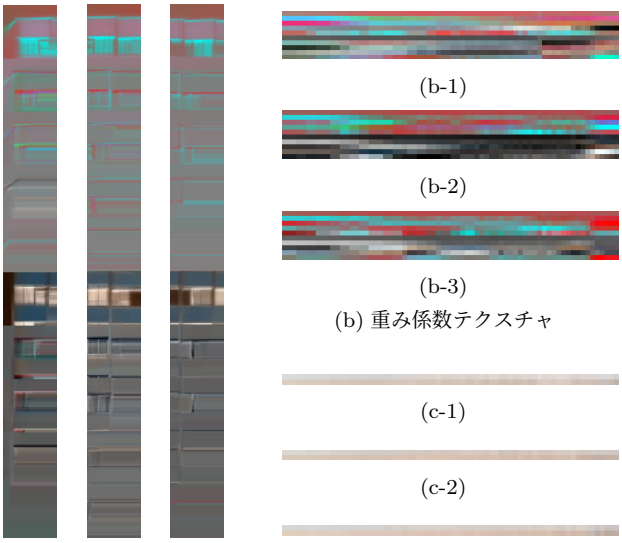
## 5. 実　　験

### 5.1　自由視点レンダリング結果

[1] の手法によるレンダリング結果を図 7 に示す．図 7(a), (b), (c) は図 7(d) において，視点 (a), (b), (c) からの見えをそれぞれレンダリングした結果である．用いた PC のスペックは Intel Core2(2.66GHz)，GPU は Quadro FX 550 であり，CPU にほとんど負荷をかけることなく，30fps で描画することが出来る．また，約 20 %に圧縮したにもかかわらず，高い再現性が実現できていることが分かる．

### 5.2　簡易評価実験

レンダリング結果において，画像の圧縮による影響があるかどうかについて，簡易的な評価実験を行った．実験方法として，撮影された圧縮前の全方位画像列と本手法により 20 %前後に圧縮した画像から生成したレンダリング結果の 2 つの動画を見




(a) 固有画像テクスチャ　(b) 重み係数テクスチャ　(c) 平均画像テクスチャ

図 6　各種テクスチャの例

せ，走行風景として 2 つの動画にどの程度違いがあるかどうかを尋ねた．図 8 は，実験に用いたオリジナル画像と本手法のレンダリング結果である．

実験結果を表 1 に示す．圧縮前の画像と比較したにもかかわらず，66%がそん色ないと回答しており，このことより，本手法の有効性が確認できた．今後は，他の圧縮法との比較などにより詳細な評価を行う必要性があると考えられる．

表 1　評価実験結果

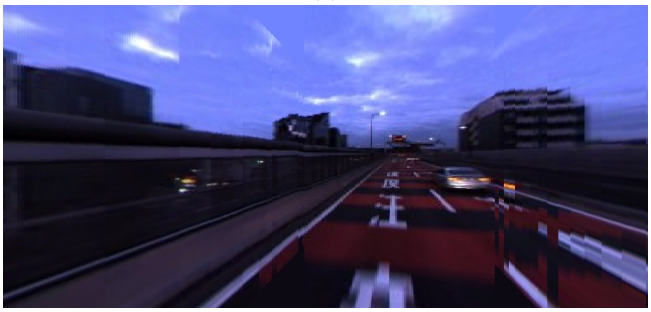
| 評価 | 人数 |
|---|---|
| 同じである | 2 |
| ほぼ同じである | 4 |
| どちらとも言えない | 4 |
| 違う | 5 |
| 全く違う | 0 |

## 6. ま と め

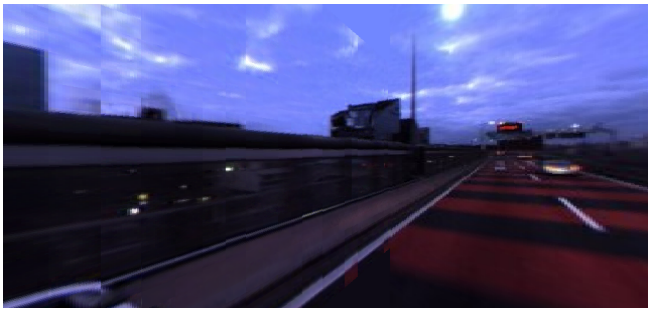
本稿では, [1] で提案された GPU による圧縮データの復元手法を広域空間でリアルタイムでレンダリングする実装方法について述べた．形状をスリット状のポリゴンにすることで，IBR に適したレンダリングを実現した．また，固有空間法により圧縮された画像データは，固有画像と重み係数との積和演算で復元できる．これを，固有画像，重み係数をそれぞれテクスチャとして GPU に渡し，マルチテクスチャ処理をすることで実現した．実際にシステムを用いて評価実験を行ったところ，圧縮の影響がほとんどない結果となり，本手法の有効性が示された．これにより，一般の PC においても広域空間をリアルタイムで写実的に再現することが可能である．

今後の課題は，提案した圧縮手法と他の圧縮法とのより詳細な評価や ITS 分野への応用として運転シミュレータの被験者実験が考えられる．

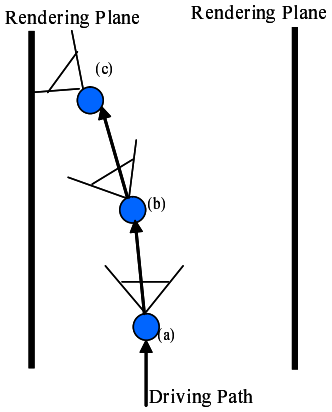
(a)



(b)



(c)

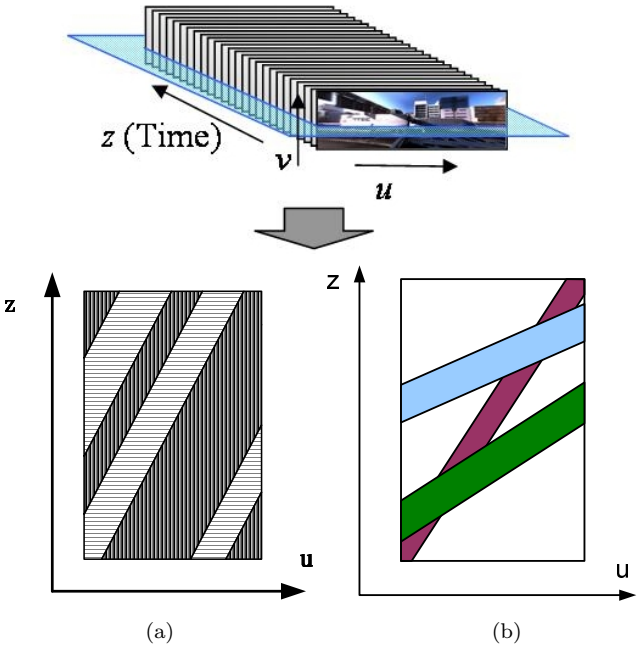



(d)

図 7　自由視点レンダリング結果

## 付録：EPI

EPI（エピポーラ平面画像）は，移動するカメラから得られる画像列をエピポーラ平面で切断した断面に得られる画像である．カメラが水平面上を移動し，かつカメラが進行方向に対して鉛直横向きのときは，画像の横軸に沿って切断することで得られる．EPI には対象物がエッジとなって表れ，その傾きは対象の奥行き（デプス）を表す [6].




(a)　(b)

図 9　(a) デプスが等しい時の EPI，(b) デプスの異なる時の EPI

### 文　献

[1] 佐藤，三上，川崎，小野，池内，”IBR に基づいた仮想都市のリアルタイムレンダリングおよびデータ圧縮の効率化手法の提案”，Meeting on Image Recognition and Understanding 2007 (MIRU2007), pp.1087-1092, 2007.

[2] C. J. Taylor,“ Video plus ”, IEEE Workshop on Omnidirectional Vision, pp. 3-11, 2000.

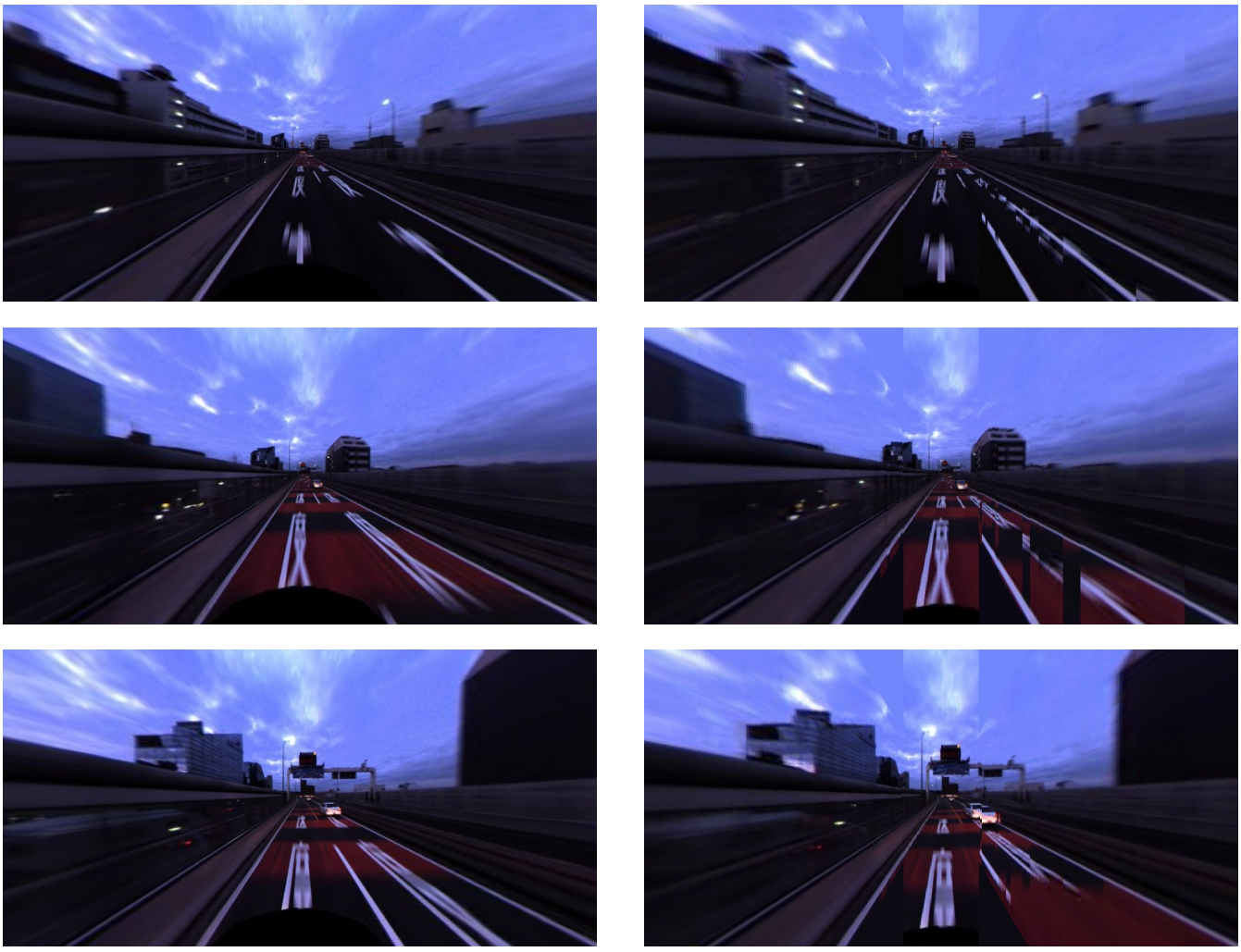
[3] H. Kawasaki, K. Ikeuchi, and M. Sakauchi,“ Light field rendering for image-scale scenes ”, Proc. International Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 2, pp. 64-71, Kauai, Hawaii, US, 2001.

[4] 堀，神原，横矢：”被写体距離を考慮した Image-Based Rendering による広域屋外環境のステレオ画像生成”，IEICE Technical Report，PRMU2006-185，2007.

[5] K. Nishino, Y. Sato, K. Ikeuchi,“ Eigen-texture method: Appearance Compression based on 3D Model ”, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), pp. 618-624, Jun. 1999.

[6] R. Bolles, H. Baker and D. Marimont: “ Epipolar plane image analysis: an approach to determining structure from motion ”, Int. J. of Computer Vision, Vol. 1, pp. 7?55, 1987.

[7] P.G.R.Inc.:Ladybug：`"http://ptgrey.com/."`



(a) オリジナル画像（圧縮前）　　(b) レンダリング結果（圧縮後）

図 8　オリジナル画像との比較