2007－CVIM－158 (13)
2007／3／19

実時間自由視点映像生成におけるフレームレート安定化 —多重解像度処理による三角パッチ数安定化—

鍋嶋 累[1] 有田 大作[2] 谷口 倫一郎[1]

[1] 九州大学
[2] 財団法人九州システム情報技術研究所

概要 本研究では，複数台のカメラによって撮影される映像から3次元モデルを復元することで，自由な視点からの映像を実時間で生成することを目指している．この自由視点映像生成システムの処理は，視体積交差法を用いた形状復元 (ボクセル表現)，ボクセル表現から三角パッチ表現への変換，三角パッチの色付けの大きく3段階に分かれる．しかしこのシステムには，対象物体の表面積が増加すると三角パッチの数が増加するため，処理時間が増加してしまうという問題点があった．これまでの研究で，形状復元時に空間解像度を対象にあわせて変化させ，多重解像度処理を行うことで，処理時間による処理の打ち切りを可能とすることで，処理時間を安定化させてきた．本稿ではさらに，三角パッチ数による処理の打ち切りを導入し，さらに処理時間を安定化させる．提案手法をPCクラスタ上に実装し，処理時間が安定することを確かめた．

Frame Rate Stabilization for Real-time Free-viewpoint Video Generation —Stabilizing the Number of Triangle Patches—

Rui Nabeshima[1] Daisaku Arita[2] Rin-ichiro Taniguchi[1]

[1]Kyushu University
[2]Institute of Systems & Information Technologies/KYUSHU
{nabeshima, rin}@limu.is.kyushu-u.ac.jp, arita@isit.or.jp

Abstract This paper proposes a method to stabilize the frame rate of real-time free-viewpoint video generation. When we apply real-time free-viewpoint video to live video programs, the frame rate of video processing should be constant. Otherwise, presented video gives viewers unnatural feeling due to time inconsistency. Our method to generate free-viewpoint video is based on 3-D shape reconstruction and visualization of the 3-D shape by CG technique, whose processing time mainly depends on the number of triangle patches of 3-D shape reconstructed. Therefore, processing time of the video generation, i.e., the frame rate, is not constant. To solve this problem, we propose a new method which flexibly varies the space resolution of the 3D-shape reconstruction stage and stabilizes the number of triangle patches, and which makes the processing time of free-viewpoint video nearly constant. We have implemented our method on a PC-cluster and realized real-time processing of the free-viewpoint video generation, whose experimental results show the effectiveness of our method.

1 はじめに

情報通信技術の進展に伴い，インターネット環境の整備や地上ディジタル放送の開始といった新しいメディアが普及してきている．これらの新しいメディアには，従来メディアでは扱うことができなかった新しいリッチコンテンツが期待されている．このようなコンテンツの一つとして自由視点映像が挙げられ，自由視点映像に関する研究が盛んに行われるようになってきている [1, 2, 3, 4, 5, 6, 7]．自由視点映像とは，複数のカメラで撮影した映像をもとに生成される，実際にはカメラのない位置（仮想視点）からの映像のことである．

自由視点映像に関する数多くの研究の中で，われわれは自由視点映像のライブ配信を目指し，これまで自由視点映像の実時間生成について研究を行ってきている [8]．自由視点映像のライブ配信が実現され

ると，野球や格闘技を自由視点映像として生中継したり，仮想空間を経由したコミュニケーションにおける遠隔地映像の生成などの応用が考えられる．

われわれが提案した実時間自由視点映像生成システム [8] は，対象物体を取り囲むように配置された複数のカメラで撮影された映像から，対象物体の 3 次元形状および色情報を再構築し，それを仮想視点から見た映像を生成するものである．この処理は，PC クラスタを用いて以下の手順で実現されている．

形状再構築 視体積交差法により多視点映像からボクセル表現された対象物体の 3 次元形状を再構築する．

ボクセル表現から三角パッチ表現への変換 3 次元形状をボクセル表現から三角パッチ表現に変換する．

色付け 各カメラから可視の三角パッチ頂点に色を付け，3 次元色形状モデルを再構築する．

描画 仮想視点から見た 3 次元色形状モデルを描画する．

自由視点映像の実時間生成を実現するためには，これらそれぞれの処理を実時間で行わなければならない．つまり，対象物体の数や形状といった入力データの条件にかかわらず，あらかじめ決められた時間（1 フレーム時間）以内に各フレームに対する処理が終了しなければならない．われわれはこれまでの研究で，3 次元形状再構築処理については多重解像度の視体積交差を用いることによって実時間処理を実現した [9]（これを旧システムと呼ぶことにする）．旧システムでは，計測空間のボクセル解像度を徐々に上げながら視体積を構築していき，1 フレーム時間が経過したところでこの処理を打ち切り，その時点での視体積を出力している．

3 次元形状再構築の実時間処理を実現すると，次に問題となる（処理時間が変動する）処理は色付け処理である．色付け処理は入力映像を基に三角パッチの各頂点に色を付けていく処理であり，その処理量は三角パッチ数に比例する．さらに自由視点映像の配信を考えると，3 次元色形状モデルが配信されることになり，そのデータサイズも三角パッチ数に比例する．つまり，三角パッチ数が変動すると，色付け処理時間およびデータ転送時間が変動することになり，これがシステムの実時間性を低下させる原因となってしまう．

そこで本稿ではこの問題を解決するために，三角パッチ数を安定させる手法を提案し，それを実現した新システムの性能評価を行う．以下，2 節で，佐藤らによって提案された 8 分木を用いた視体積交差法 [10] について簡単に紹介する．この手法を基にして新旧システムでの多重解像度処理による 3 次元形状再構築を実現している．3 節で PC クラスタを用いた新旧システムの構成を示す．4 節および 5 節で，旧システムにおける多重解像度 3 次元形状再構築手法について述べ，6 節で新システムにおける三角パッチ数安定化の手法について述べる．7 節で実験結果を示し，8 節でまとめる．

2 8 分木を利用した視体積交差法

新旧システムにおける 3 次元形状復元では，佐藤らが提案した 8 分木を利用した視体積交差法 [10] を用いている．これは，形状再構築の空間解像度を徐々に上げていき，1 フレーム時間が経過するとそこで処理を打ち切ることによりフレームレートの安定化を図るためである．本節では，この 8 分木を利用した視体積交差法について簡単に説明する．

2.1 パスアルゴリズム

前処理として，計測空間を表す木を作成する．根は計測空間全体を表すノード (=ボクセル)n^0 とし，これをあらかじめ決められた大きさ（対象物体がすっぽりと入ってしまうことのない大きさ）のボクセルに分割したものをノード n^1（以下、初期ノードと呼ぶ）とする。次に，初期ノード n^1 を 8 分木構造にしたがって $n^l \in \{n \mid Level(n) = l\}$（$l$: 木の深さ=解像度レベル）に分割していく．ここで，$Level(n)$ はノード n の解像度レベルを表す．これを $l = Level_{max}$(最大解像度) まで繰り返すことにより木を作成する．

パスアルゴリズムについて説明するために，以下のような表記法を導入する．まず，ノード n に対応するボクセルの頂点 $v \in V_n$ をカメラ c からの入力画像に投影した点が前景であるか背景であるかを表す式を以下のように定義する．

$$Proj(v,c) = \begin{cases} \mathbf{FG}\ (\text{前景の場合}) \\ \mathbf{BG}\ (\text{背景の場合}) \end{cases} \quad (1)$$

次に，すべてのカメラ C_* からの入力画像を用いて頂点が前景であるか背景であるかを表す式は，式 1 を用いて以下のように定義できる．

$$Proj(v,C_*) = \begin{cases} \mathbf{FG}\ (if(\forall_c \in C_*) Proj(v,c) = \mathbf{FG}) \\ \mathbf{BG}\ (otherwise) \end{cases} \quad (2)$$

さらに，各ノードを以下の式によって，すべての入力画像において8頂点すべてが前景に投影される白ボクセル $\mathbf{W}$，一つ以上の入力画像において8頂点すべてが背景に投影される黒ボクセル $\mathbf{B}$，境界部分に投影される灰色ボクセル $\mathbf{G}$ の3種類に分類する．

$$Label(n, C_*) = \begin{cases} \mathbf{U}\ (未判定) \\ \mathbf{W}\ (if(\forall_v)(\forall_c)Proj(v, c \in C_*) = \mathbf{FG}) \\ \mathbf{B}\ (if(\forall_v)(\exists_c)Proj(v, c \in C_*) = \mathbf{BG}) \\ \mathbf{G}\ (otherwise) \end{cases} \quad (3)$$

パスアルゴリズムでは，まずすべてのノードについて $Label(n, C_*) \leftarrow \mathbf{U}$ とする．次に初期ノードに対して式3による判別を行う．その後，$Label(n^l, C_*) = \mathbf{G}$ であれば，その8個の子ノード n^{l+1} に対して，式3による判別を行うという処理（これを分割処理と呼ぶ）を，最大解像度 $Lebel_{max}$ まで再帰的に適用する．ここで，解像度レベルの低いノードから最大解像度レベルのノードまで繰り返し行われる分割をパスと呼び，このアルゴリズムをパスアルゴリズムと呼ぶ．

2.2 マルチパス再分割アルゴリズム

上記の判別方法は大変単純なものであるので，実際には誤りがしばしば起こる．例えば，頂点を避けるような細長い先端部分は実際には灰色ボクセルであり分割されるべきであるが，黒ボクセルと判定されてしまい分割が行われない（図1）．このような分割されるべきであるのに分割されないボクセルのことを失敗ボクセルと呼ぶ．

このような誤りに対応するために，以下のマルチパス再分割アルゴリズムを用いる．

StepA-1　初期ノード n^1 にパスアルゴリズムを施す

StepA-2　失敗ボクセルを探索し，発見された場合はStepA-3へ進み，失敗ボクセルが発見されない場合は処理を終える

StepA-3　発見されたすべての失敗ボクセルを8分割し，新たに得られたボクセル $n^{l+1} \in \{n^{l+1} \mid isFailed(Parent(n), C_*) = \mathbf{T}\}$ に対してパスアルゴリズムを施し，StepA-2へ戻る

ここで，ノード n が全カメラ C_* による形状復元において失敗ボクセルであるかどうかを $isFailed(n, C_*) = \{\mathbf{T}, \mathbf{F}\}$ で表す（具体的な失敗ボクセル探索手法については文献[9]参照）．

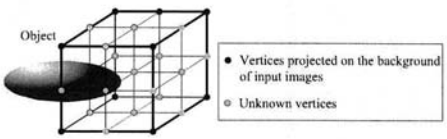

図1　失敗ボクセル

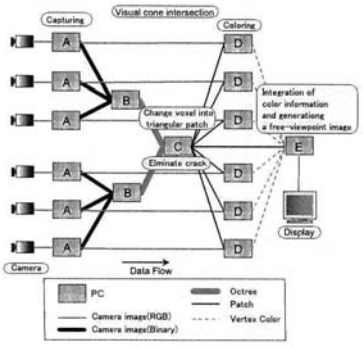

図2　PCクラスタによるシステム構成

3 PCクラスタ構成

新旧システムは，その計算量および入力データ量の多さのため1台の計算機での実現は困難なことから，PCクラスタを用いた並列分散処理によって実現されている．その構成は図2のようになっており，各PCでの処理内容は以下のとおりである．

ノードPC-A　カメラ画像を取得し，その中から背景差分によって対象物体を抽出する．対象物体を抽出した画像をノードPC-B，ノードPC-Dに送る．ただしノードPC-Bは視体積構築のためだけに用いるので，データ量を少なくするために白黒画像を送る．一方ノードPC-Dでは色付けを行うためカラー画像を送る．

ノードPC-B　ノードPC-Aから送られてきた抽出画像を用いて視体積を求め，8分木としてノードPC-Cに送信する．ノードPC-Bではカメラグループ C_k に対する視体積を構築するので，式2と式3の C_* を C_k に置き換えたものを用い，マルチパス再分割アルゴリズムを適用する．なお，送信するのは，8分木の各ノード n が持つ頂点情報 $Proj(v \in V_n, C_k)$ である．ノードの判別情報ではなく頂点情報を送るのは，ノードPC-Cにおいて離散マーチン

グ・キューブ法 (DMC) 法 [11] を用いて三角パッチ表現の形状モデルを作成するためには頂点情報が必要であるためである．

ノード PC-C　まず，ノード PC-B から送られてくる視体積の交差（共通部分）を求める．次に，得られた多重解像度のボクセルデータに対して，DMC 法を適用することで三角パッチ表現の形状モデルを生成する．そしてノード PC-D と E に三角パッチデータを送る．

ノード PC-D　ノード PC-C から送られてきた，三角パッチに，ノード PC-A から送られてきた画像を基に色を付ける．得られた色情報をノード PC-E に送る．

ノード PC-E　ノード PC-C からの三角パッチデータとノード PC-D からの色データとユーザから入力された仮想視点位置から，重み付き色付き対象形状を生成，すなわち対象物体の自由視点映像を生成する．

以下 4 節と 5 節では，ノード PC-B とノード PC-C における視体積構築処理および視体積交差処理についてそれぞれ述べる．

4　多重解像度での視体積構築

4.1　処理の概要

8 分木を利用した視体積交差において，初めから最大解像度までマルチパス再分割アルゴリズムを適用するのではなく，それよりも低い空間解像度（以下，マルチパス再分割アルゴリズムを適用する空間解像度を解像度レベルと呼ぶ）まで適用し，その処理が終了したら解像度レベルを高くする処理を繰り返す．そして，解像度レベルが最大解像度に到達するか，ある一定の時間（打ち切り時間）が経過するかした時点で処理を打ち切り，その時点での形状を出力とする．処理が打ち切られたときの解像度レベルを打ち切り解像度と呼ぶこととする．これにより 1 フレームあたりの処理時間の変動が小さくなり，フレームレートを安定させることができる．処理が途中で打ち切られた場合は空間解像度の低い復元形状となるが，空間解像度を徐々に高めていっているため計測空間全体に対して形状復元を行うことができる．

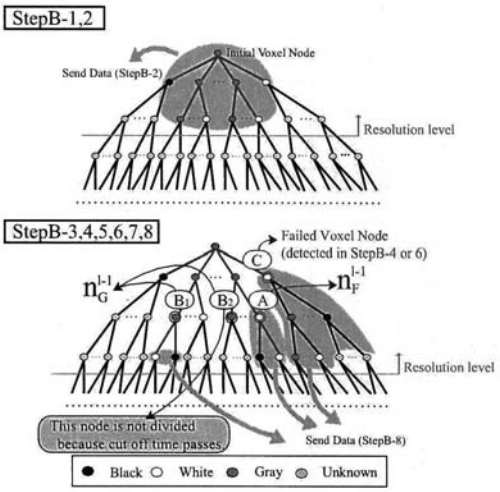

図 3　視体積構築の流れ

4.2　視体積構築処理

視体積構築処理を以下に示す．図 3 上は StepB-1, 2 を示し，図 3 下は StepB-3〜8 を示す．

StepB-1　ある解像度レベル l までマルチパス再分割アルゴリズム (StepA-1〜StepA-3) を施す

StepB-2　処理結果を出力する

StepB-3　解像度レベルを高くする ($l \leftarrow l+1$)

StepB-4　解像度レベルが高くなったことにより新たに分割できるようになった失敗ボクセル $n_F^{l-1} \in \{n^{l-1} | isFailed(n^{l-1}, C_k) = \mathbf{T}\}$ に対してマルチパス再分割アルゴリズムを施す (図中の A のボクセル)．すなわち，StepA-1 において「初期ノード」を「分割可能になった失敗ボクセル」に変更してマルチパス再分割アルゴリズムを施す

StepB-5　打ち切り時間が過ぎていれば StepB-8 へ進む

StepB-6　解像度レベルが高くなったことにより新たに分割できるようになった灰色ボクセル $n_G^{l-1} \in \{n^{l-1} | Label(n^{l-1}, C_k) = \mathbf{G}\}$ のうちの M_G 個（今回の実験では $M_G = 100$）に対してマルチパス再分割アルゴリズムを施す (図中の B のボクセル)

StepB-7　打ち切り時間が過ぎた場合，あるいは StepB-6 において灰色ボクセルを全て分割した場合は StepB-8 へ進む．そうでなければ

StepB-6 に戻る

StepB-8　StepB-4, 6 で得られた処理結果を出力する

StepB-9　$l = Level_{max}$ であれば終了し，そうでなければ StepB-3 に戻る

StepB-8 での処理結果の出力では，前回の出力との差分情報のみを出力する．差分情報とは，StepB-4 や StepB-6 で分割された失敗ボクセルを根とする部分木の各ノード $n \in N_{T_{N_F}}$ の頂点情報 $Proj(v \in V_n, C_k)$ と StepB-6 で灰色ボクセルを分割して得られたノード $n \in N_{T_{N_G^{l-1}}}$ の頂点情報 $Proj(v \in V_n, C_k)$ である．ただし，N_{T_n} はノード n を根とする木 T_n のノードの集合を指す．また送信データには，黒ボクセルの子ノードや白ボクセルの子ノードを含まないため，データ転送量を削減することができる．

5　多重解像度での視体積交差

5.1　処理の概要

ノード PC-C では，視体積交差処理，およびボクセル表現の 3 次元形状モデルから三角パッチ表現の 3 次元形状モデルへの変換処理を行う．これは以下のような手順で行われる．

1. 視体積交差
   徐々に解像度レベルを高めながら送られてくる視体積どうしの共通部分を求める．
2. クラックパッチング
   ボクセル表現された 3 次元形状モデルは多重解像度であるため，そのまま DMC を適用すると三角パッチに裂け目（クラック）が発生してしまう．そこで，クラックが発生してしまう位置を検出し，クラックが発生しないようにボクセルを分割する（クラックパッチング処理の詳細については [9] を参照）．
3. 三角パッチ表現への変換
   ボクセル表現された 3 次元形状モデルに対して，DMC を適用することによって三角パッチ表現へ変換する．

本節では，三角パッチ数安定化の説明に必要な視体積交差処理について詳しく述べる．

5.2　視体積交差処理

複数のノード PC-B で求められた視体積はそれぞれにおける打ち切り解像度の違いから空間解像度が異なる場合があり，それらの視体積の積演算を行うためには，複数の空間解像度の混在に対応した，つまり多重解像度の視体積交差手法が必要となる．また，視体積構築においては解像度レベルを上げるごとにその時点での視体積を送信するので，視体積交差では最初の視体積を受信するとすぐに積演算を開始する．そして新たな視体積が得られるたびに，必要なノードだけに対してのみ積演算を行う．このようにノード PC-B における視体積構築処理とノード PC-C における視体積交差処理をオーバーラップさせて実行することで，システム全体の遅延時間の削減を実現している．

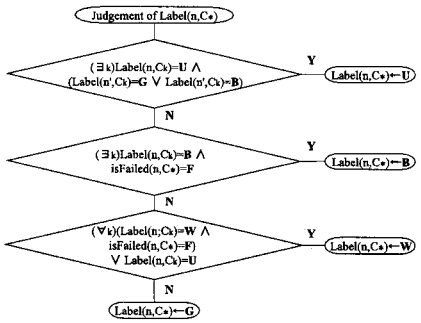

図 4　ノードの判定

8 分木中の同じ位置のノードは同じ空間位置のボクセルを指すので，ノードどうしの積演算を行うことで視体積交差を行うことができる．具体的には図 4 にしたがって，8 分木の各ノード n を判別する．ただし，n' は，ノード n の祖先のノードの内の打ち切りノードを指す．つまり，$Anc(n)$ をノード n のすべての祖先のノードの集合を表すとすると，ノード n' は $n' \in Anc(n) \cap N_{cutoff}^{C_k}$ を満たす．

ノード PC-B からは，初期ノード n^1 を根とする 8 分木 T_{n^1} と，失敗ボクセル n_F を根とする 8 分木 T_{n_F} が送られてくる．ノード PC-C では，これらの 8 分木の根から図 4 の判定を行っていき，**G** と判定されたノードについてのみ，その子ノードについても再帰的に判定を行う．

6　三角パッチ数の安定化

三角パッチ数を安定化させるために，視体積交差処理中に三角パッチ総数があらかじめ決められた打ち切り三角パッチ数に到達したら視体積交差処理を打ち切る処理を追加する．具体的には，ノード PC-C

における視体積交差処理において，以下のように処理が行われる．

視体積交差処理では，ノードが分割されるべきかどうかが図4によって判定されるとともに，(5.2節の最後でも述べたように) 式2によってそのノードの頂点の判定も同時に行われる．頂点の判定が得られると，そのノードを三角パッチに変換したときにいくつの三角パッチが生成されるかがわかる．この数を三角パッチ総数に加算していく．ただし，解像度レベルが上がり，ノードが分割された場合は，そのノードはDMCの対象ではなくなる (DMCはN_{cutoff}にのみに適用される) ので，そのノードから生成されるとした三角パッチの数を三角パッチ総数から減算する．この処理をノードの判定が行われるたびに行い，三角パッチ総数が打ち切り三角パッチ数に到達したら，視体積交差処理を打ち切る．ただし，1個の親ノードを分割することによって得られた8個の子ノードに対する処理の途中は打ち切りを禁止しておき，それらに対する処理が終了した時点で，視体積交差処理打ち切りの判定を行う．

また，失敗ボクセルが発生した場合は，(現在の解像度レベルよりも低い解像度レベルである) その失敗ボクセルを根とする部分木に対して視体積交差処理のやり直しが必要となる．したがって，本来は，やり直しとなる部分木内の三角パッチ数を三角パッチ総数から減算した上で，その部分木に対する視体積交差処理によって生成された三角パッチ数を加算していかなければならない．しかし，この減算をおこなうためには部分木内の三角パッチ数を求めなければならず，これは時間のかかる処理である．そこで本システムではこの減算は行わず，加算のみを行っている．したがって，求まった三角パッチ総数は，実際の三角パッチ総数よりも多くなる．しかし，失敗ボクセルはそれほど頻繁に発生するわけではないため，誤差もそれほど大きいわけではなく，また，打ち切り判定における三角パッチ総数は厳密である必要はないことから，問題はないと考えている．なお，失敗ボクセルを根とする部分木に対する視体積交差やり直しの間も打ち切り禁止であり，それらに対する処理が終了した時点で，視体積交差処理打ち切りの判定を行う．

なお，三角パッチ数による視体積交差処理の打ち切りを行ったあとでクラックパッチングを行うので，(上で述べた失敗ボクセル処理による三角パッチの誤差に加えて) クラックパッチングによる三角パッチ数の増加があり，最終的な三角パッチ数は変動することになる．

7 実験と考察

7.1 実験環境

本実験では16台のPC (Intel Pentium4 3GHz，メモリ1GB，RedHat Linux9) からなるPCクラスタを用いた．各PCはMyrinetによって相互に結合されており，100MB/s以上で通信が可能である．さらに対象を取り囲むように配置した6台のIEEE1394ディジタルカメラが接続されており，全てのカメラは同期信号発生装置により同期がとられている．カメラ画像の解像度は640×480である．最大空間解像度は256^3，最小ボクセルの一辺を$1cm$，木の深さは5，初期ノードの空間解像度は8^3として実験を行った．また，出力解像度は64^3，128^3，256^3の3段階とした．さらに，打ち切り時間は125msecとしている．

多重解像度による自由視点映像生成では，クラックパッチングを行わない場合 (手法1)，クラックパッチングを行う場合 (手法2)，クラックパッチングに加えて三角パッチ数による打ち切りを行う場合 (提案手法) について計測を行った．また比較対象として，固定解像度による自由視点映像生成についても計測を行った．これは，提案システムにおいて，打ち切り時間による処理打ち切りを行わず，3種類の最大空間解像度64^3，128^3，256^3まで処理を続けるようにしたものである．

これらにより，二人の人物が計測空間から出たり入ったりする様子の自由視点映像を生成する．人数の変化により処理量の変化を起こしている．

7.2 三角パッチ数の変動

図5に生成された三角パッチ数の変動を示す．このグラフから，提案手法での三角パッチ数の変動が小さいことがわかる．

7.3 処理時間の変動

ノードPC-C，ノードPC-D (手法2との比較)，ノードPC-E (手法2との比較) における処理時間の変動を，それぞれ図6，図7，図8に示す．

図6から，手法2と比較すると提案手法の処理時間は安定していることがわかる．しかし，手法1と比較すると提案手法や手法2では処理時間が大幅に長くなり，その変動も大きいことがわかる．これは

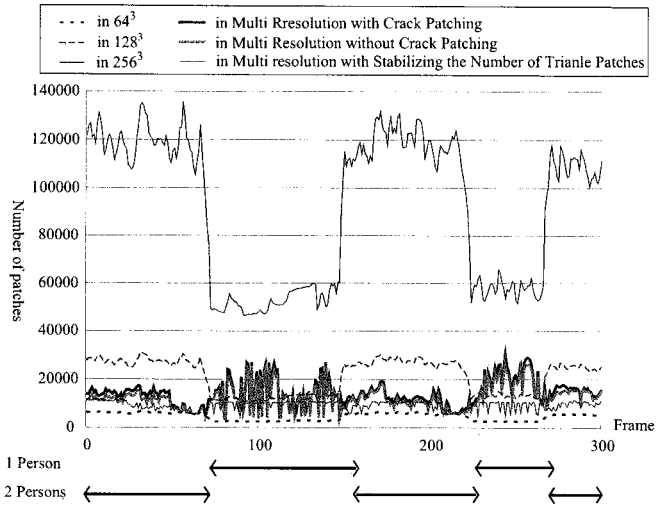

図 5　三角パッチ数の変動

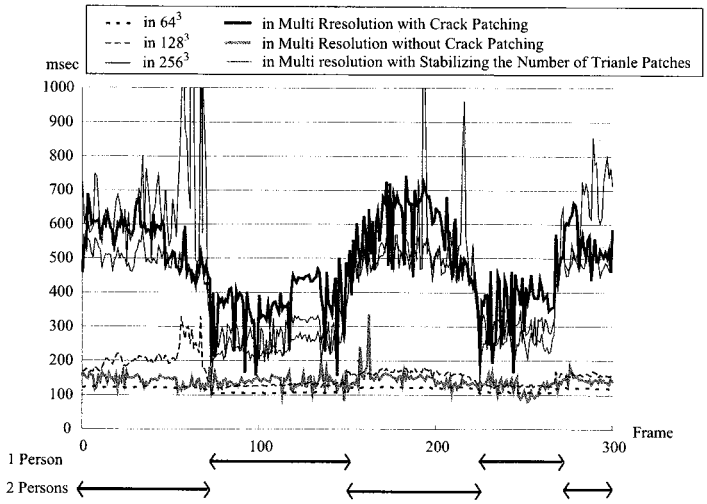

図 6　ノード PC-C の処理時間

クラックパッチング処理に時間がかかっていること，その処理時間が大きく変動することを意味している．その理由は，（詳しい理由はページ数の制約のため割愛するが）クラックが発生しているかどうかを検出する処理量が低解像度のボクセルが多いほど増大することに起因していると考えられる．人物が 1 人だけの場合は処理時間に余裕があり，最大解像度まで処理が進むことから，低解像度のボクセルが少なくなり処理時間が短くなる．一方，人物が 2 人の場合は，処理が最大解像度まで進まずに打ち切られ，低解像度のボクセルが増え処理時間が長くなる．このため，処理時間の変動が大きくなってしまう．さらに，ノード PC-C の処理時間がシステム全体の中でもっとも長いことから，ノード PC-C の処理時間の変動によりシステム全体のフレームレートが変動してしまっている．アルゴリズム見直しなどによってクラックパッチングの処理時間を短縮することによってこの問題を解決することが，今後の課題となっている．

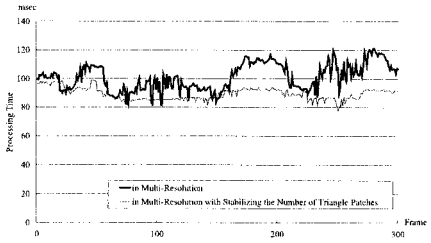

図 7　ノード PC-D の処理時間

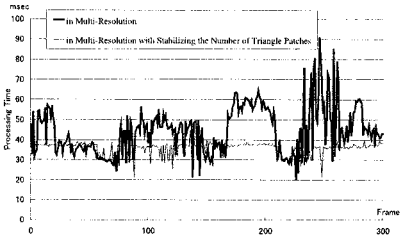

図 8　ノード PC-E の処理時間

図 7 および 8 から，三角パッチ数が安定したことにより，ノード PC-D およびノード PC-E の処理時間が安定したことがわかる．

8　おわりに

本稿では，自由視点映像のライブ配信を目指し，自由視点映像生成のフレームレートの安定化を図った．具体的には，生成される三角パッチ数を安定させることで，処理時間を安定させる手法を提案した．提案手法を実装し，実験を行ったところ，従来手法よりも三角パッチ数およびフレームレートが安定した．しかし，多重解像度の 3 次元形状復元によって発生する三角パッチの隙間をふさぐ処理（クラックパッチング処理）時間が長く，システム全体の性能を低下させてしまっている．

今後の課題は，アルゴリズム見直しなどによってクラックパッチング処理の高速化を図ること，最大解像度，打ち切り時間，打ち切り三角パッチ数のバランスをとることでシステム性能を向上させること，ライブ配信に向けた遠隔地通信実験を行うことなどが挙げられる．

謝辞

本研究の一部は，財団法人大川情報通信基金平成 17 年度研究助成「多視点映像からの自由視点映像の実時間生成」の補助を受けた．

参考文献

[1] T. Kanade, P. W. Rander, P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, Vol. 4, No. 1, pp. 34–47, 1997.

[2] T. Matsuyama, X. Wu, T. Takai, and S. Nobuhara. Real-time 3dshape reconstruction, dynamic 3d mesh deformation, and high fidelity visualization for 3d video. *International Journal on Computer Vision and Image Understanding*, Vol. 96, No. 3, pp. 393–434, 2004.

[3] 北原格, 大田友一, 斎藤英雄, 秋道慎志, 尾野徹, 金出武雄. 大規模空間における多視点映像の撮影と自由視点映像生成. 映像情報メディア学会誌, Vol. 56, No. 8, pp. 120–125, 2002.

[4] M. Levoy and P. Hanrahan. Light field rendering. In *Proc. of SIGGRAPH*, pp. 31–32, 1996.

[5] 斉藤英雄, 木村誠, 矢口悟志, 稲木奈穂. 射影幾何に基づく多視点カメラの中間視点映像生成. 情報処理学会論文誌, Vol. 43, No. SIG 11(CVIM 5), pp. 21–32, 2002.

[6] B. Goldlucke and M. Magnor. Real-time microfacet billboarding for free-viewpoint video rendering. In *Proc. IEEE International Conference on Image Processing*, Vol. 3, pp. 713–716, 2003.

[7] W. Matusik, C. Buehler, R.Raskar, S. Gortler, and L. McMilla. Image-based visual hulls. In *Proc. of SIGGRAPH*, pp. 369–374, 2000.

[8] 上田恵, 有田大作, 谷口倫一郎. 多視点動画像処理による 3 次元モデル復元に基づく自由視点画像生成のオンライン化 － pc クラスタを用いた実現法－. 情報処理学会論文誌, Vol. 46, No. 11, pp. 2768–2778, 2005.

[9] 鍋嶋累, 上田恵, 有田大作, 谷口倫一郎. 実時間自由視点映像生成のフレームレート安定化 形状復元の多重解像度処理. 画像の認識・理解シンポジウム, pp. 642–647, 2006.

[10] Hidenori Sato, Hiroto Matsuoka, Akira Onozawa, and Hitoshi Kitazawa. Image-based photorealistic 3d reconstruction using hexagonal representation. 情報処理学会論文誌, Vol. 46, No. 2, pp. 639–648, 2005.

[11] 剣持雪子, 小谷一孔, 井宮淳. 点の連結性を考慮したマーチング・キューブ法. 信学技報 PRMU98-218, pp. 197–204, 1999.