基礎論文

GPUを用いた全周囲立体CG映像の実時間生成

城 堅誠[*1]　南澤 孝太[*1]　新居 英明[*1]　川上 直樹[*1]　舘 暲[*1]

A GPU-based Real-Time Rendering of Immersive Stereoscopic Images

Kensei Jo[*1], Kouta Minamizawa[*1], Hideaki Nii[*1], Naoki Kawakami[*1], Susumu Tachi[*1]

Abstract – **The goal of this study is to render immersive stereoscopic images in real time for realization of interactive CG worlds. In our proposed method, the vertices are moved to their corresponding locations by using a vertex shader according to the geometry of the display. Next, the polygons are subdivided by a geometry shader in order to reduce distortions. This method enables fast rendering of immersive stereoscopic images with less distortion. Furthermore, the rendering feature that transform verteces data for immersive stereoscopic displays is implemented on GPU. This implies that the method allows CG creators to create contents using typical 3D CG APIs without considering the display ′ s geometries.**

Keywords : **Real-time Rendering, Computer Graphics, Programmable Shader, Immersive Stereoscopic Display, Virtual Reality**

1　はじめに

近年，高い臨場感を持つ映像を提示するために，観測者の全周囲をディスプレイ面で覆う没入型ディスプレイが多数開発されてきた．これらの没入型ディスプレイは視界のほぼ全域に対し映像を提示できるため，空間全体を映像提示する用途や，テレイグジスタンス [1] 等の用途に用いられている．その中で図 1 に示す TWISTER:Telexistence Wide-angle Immersive STEReoscope[2] は，裸眼で全周囲の立体映像を見ることができる没入型ディスプレイである．

これらの没入型ディスプレイは，CAVE[3] や CABIN[4] のように周囲を平面ディスプレイで覆うディスプレイと，TWISTER や EnspheredVision[5] などのように周囲を曲面状のディスプレイ面で覆われたディスプレイとに分類できる．前者は平面ディスプレイ用の映像を容易に使用することができるものの，後者の非平面の提示面を持つ没入型ディスプレイにおいては，平面ディスプレイ用の映像を直接用いることはできない．そのため，いくつかの描画手法がこれまで用いられてきた．

観測者の周囲をドーム状のスクリーンで覆うことで視界全体に映像を投影する EnspheredVision では，光学系の光線追跡のシミュレーションを行い，プロジェクタの出力画像の座標（提示画像座標とする）と，実際にスクリーン上で投影される点の座標（ディスプレイ面座標とする）の対応を求め，一度描画した画像に歪み補正を施すことで映像を生成している．また橋本ら [6] はカメラを用いて対応の取得を自動化する手法も提案している．現実世界のスクリーンと双対な 3 次元モデルを CG(Computer Graphics) 空間で構築し，CG 中での映像の見え方を元に投影映像を生成する手法も提案されれている [7, 8, 9, 10].

図 1　TWISTER の外観（左）, 内部の映像（右）.
Fig. 1　Appearance of TWISTER IV.

しかしながら，TWISTER で用いられる全周囲立体映像は，全周囲にわたって視差を正しく反映させる必要があり，観測者の両眼の位置が，頭部の向きに依存することも考慮しなければならない．そのため平面ディスプレイ用に描画した 1 枚の映像に対し，前述の歪み補正の例のような幾何変形を施しても，必要な映像を生成することはできない．

全周囲立体映像の場合，観測者の頭部の向きによって眼球の位置も変化するため，両眼の視差を画像にどのように反映させればよいかは自明ではない．TWISTER では観測者の正面方向を，常に正しく立体視できるようにし，正面方向以外の光線情報は他の方向へ提示し

[*1] 東京大学大学院情報理工学系研究科
[*1] Graduate School of Information Science and Technology, The University of Tokyo

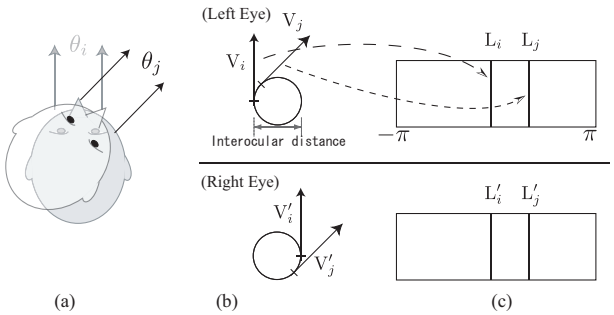


図 2　(a) 人の頭部．(b) 頭部の回転によって左右の目の位置の変化．(c) 対応する画像．
Fig. 2　(a)Head.　(b)Eye positions according to head rotation.　(c)Created picture.

た光線情報を利用する近似手法が用いられている．この近似を満たす映像として同心モザイク (Concentric Mosaic)[11] が用いられ，田中ら [12] によって, 同心モザイクを使用したときの誤差が TWISTER における立体視に深刻な問題を引き起こさない，と解析的に確認されている．

これまで同心モザイク映像を描画するためには，図 2 に示すように CG 空間上で，直径が眼間距離と等しい円の円周上にカメラを配置し，そこで得たスリット状の画像を並べることで映像を生成していた．例えば観測者が θ_i 方向を向いていた条件において，その時の CG 上での左右の目に相当するカメラの位置をそれぞれ V_i, V'_i とする．V_i, V'_i の位置で描画されたスリット状の画像 L_i, L'_i を出力画像に貼り付ける．全ての方位に対しこの操作を繰り返すことで，左右の目の同心モザイク画像が生成される．この手法で, 十分な精度の画像を得るには，スリットの幅を小さくし，描画回数を増やす必要があるため，平面ディスプレイ用の映像生成時に比べ描画に時間がかかる点が問題であった．また，既存のアプリケーションを TWISTER に必要な全周囲立体映像に変更しようとした場合，それぞれのアプリケーションごとに描画部を解析し，全周囲立体映像用の描画処理に書き換えなければいけないため，コンテンツの生成に手間がかかることが大きな問題であった．

そこで本研究では，プログラマブルシェーダを用いることで，既存の CG コンテンツを TWISTER に必要な全周囲立体映像に直接変換し，実時間で高精細な映像を描画する手法を提案することを目的とする．本手法では，通常の 3DCG で用いられる透視投影や正射影の代わりに，TWISTER に必要な投影変換をプログラマブルシェーダ上で施し，ポリゴンの頂点位置を提示画像中で適切な位置に移動させる．ただし，移動させた頂点から辺を直線により描画すると，2.2 節で説明する理由によりポリゴンの辺の中央部位に歪みが発生する．この問題を解消するため，大きなポリゴンを GPU(Graphics Processing Unit) 上で分割する手法も導入する．その結果，通常の描画命令で出力されたポリゴンの頂点データ群を，TWISTER に応じた頂点データ群に GPU 上で変換することが可能となる．

本手法のメリットは，大きく分類して二つ挙げられる．一点目は，後から歪み補正や合成を行うことなく映像を直接生成できるため，描画速度の向上につながる点である．特に同心モザイク映像などのように，1 フレームの映像生成のために多くの描画が必要な場合では，顕著な速度向上が期待できる．二点目は，従来手法に比べて，既存のアプリケーションを TWISTER 用に書き換えるために必要な手間が，大幅に削減できる点である．提案手法では，プログラマブルシェーダを用いて GPU 上で変換処理を実行するため，変換処理の核となるコードは異なるアプリケーションであっても共通である．このため，元々プログラマブルシェーダを使用していないアプリケーションならば，従来手法に比べて容易に TWISER 用のコンテンツとして移植することが可能となる．

本論文では，以下 2 章にて提案手法の詳細を述べ，3 章にて構築例を述べる．その後，4 章にて評価を行う．

2　提案手法

本章では，提案手法の詳細を順を追って説明する．提案手法は,「投影変換関数の決定」,「頂点の移動」,「ポリゴン分割」の 3 つに分けられる．

2.1　投影変換関数の決定

本手法では，透視投影や正射影の代わりに，TWISTER に必要な投影変換関数を用いて，ポリゴンの頂点を画面上の適切な位置に移動させ描画する．ここで投影変換関数を，カメラを原点とした座標系におけるポリゴン頂点の座標を入力とし，ディスプレイの提示画像座標における，その頂点を投影するべき座標を出力情報とする関数と定義する．

なお TWISTER においては同心モザイク画像が必要であるため，以下，投影変換関数として同心モザイクの投影変換関数を導出する．

TWISTER における投影変換関数の導出

まず，TWISTER に必要な投影変換関数を導出する．導出のためには，ある点 **A** が与えられた時に，その点を TWISTER の提示画像座標上のどこに描画すればよいか計算することにより求めることができる．

与えられた点 **A** と TWISTER のディスプレイ面の関係を図 3 の模式図に表す．また TWISTER のディスプレイ面座標と提示画像座標の対応関係は図 4 で表されるように，ディスプレイ面座標の円筒座標表記と提示画像座標が一致するように定める．観察者の頭部を

TWISTER の中心部に置き，両眼は半径 d_e の円周上に必ず存在すると仮定する．また点 **A** の TWISTER 中心部，すなわち CG 上でカメラを配置している点からの座標を $\mathbf{A}(x_A, y_A, z_A)$ とし，点 **A** を円筒座標で表した座標を $\mathbf{A}'(r_{A'}, \theta_{A'}, z_{A'})$ とする[1]．

$$\begin{cases} r_{A'} &= \sqrt{x_A^2 + z_A^2} \\ \theta_{A'} &= \mathrm{atan2}(x_A, -z_A) \\ z_{A'} &= y_A \end{cases} \tag{1}$$

また OpenGL の標準的なカメラは Z 軸の負の方向を向いているため，θ の零点は図 3(b) のように Z 軸負方向に定める．

同心モザイクの原理により，この点 **A** を正面から見る左右の目の位置は図 3(b) の点 $\mathbf{E_L}, \mathbf{E_R}$ であるため，TWISTER のディスプレイ面上で点 **A** を投影するべき位置は $\mathbf{A_L}, \mathbf{A_R}$ となる．このときの $\mathbf{A_R}$, および $\mathbf{A_L}$ の円筒座標は

$$\left(R_T, \quad \theta_{A'} \pm \alpha, \quad \frac{R_T}{r_{A'}} z_{A'} \right) \tag{2}$$

となる．ただし R_T は TWISTER の半径を表し，α は以下に表される．

$$\alpha = \cos^{-1}\left(\frac{d_e}{r_{A'}}\right) - \cos^{-1}\left(\frac{d_e}{R_T}\right) \tag{3}$$

このとき TWISTER に投影する提示画像座標の u 値および v 値は，前述のようにディスプレイ面座標の θ と Z の値にそれぞれ対応しているため，点 **A** を表す投影画像座標は以下になる．

$$(u_A, v_A, d_A) = \left(\theta_{A'} \pm \alpha, \quad \frac{R_T}{r_{A'}} z_{A'}, \quad r_{A'} \right) \tag{4}$$

ただし，d_A は陰影処理に必要なカメラからの奥行き距離の値を表し，TWISTER の中心軸から点 A までの距離である $r_{A'}$ を用いた．

本論文では以上の座標変換を TWISTER 投影変換と呼ぶ．

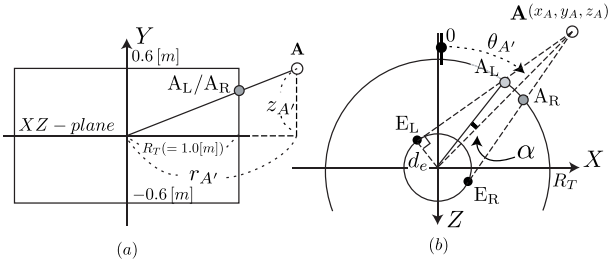


図 3　TWISTER を側方から見た座標系 (a) と，上面から見た座標系 (b).

Fig. 3　Coordinate system definition of TWISTER.

[1] $\mathrm{atan2}(y, x)$ は，$\frac{y}{x}$ の逆正接を返す関数とする．

2.2　頂点の変換

次にプログラマブルシェーダを用いて変換処理を開始する．まず頂点シェーダを用いて，入力された全ての頂点の座標に対し，上記の投影変換関数を施し提示画像の座標平面に出力する．

図 5(b) は図 5(a) の全頂点に TWISTER 投影変換を施した画像を表す．前方の小さなタイル群の並びなど，元々直線的に並んでいたポリゴンの頂点が，図 5(b) では曲線状に配置されていることが紙面上からも確認できる．しかし，大きなポリゴンの辺の部位では歪みが生じている．これは，頂点と頂点の間の辺の部位が自動的に直線補間されることに起因している．TWISTER 投影変換は非線形変換であるため，本来ならば辺は直線ではなく曲線で補間されるべきである．

2.3　ポリゴンの分割

大きなポリゴンでは辺に歪みを生じるため，ジオメトリシェーダを用いてポリゴンを分割する．さらに分割によって生じた新たな頂点に対して TWISTER 投影変換を施し，元々の大きなポリゴンを，歪みの無視できる程度の小さなポリゴンの集合に変換する．その結果，歪みの目立たない高精細な映像に自動的に変換することが可能となる (図 5(c))．

次に，本手法で用いたプログラマブルシェーダ上でポリゴンを分割する具体的手法について述べる．

先行研究および本研究で求められる機能

ポリゴン分割は複雑な問題であり，古くから有限要素法や CG などの分野において，様々な手法が提案されている．

有限要素法においては，解析的に解くことが難しい微分方程式の近似解を数値的に得るため，領域をメッシュ分割して数値計算を行う．そのメッシュ形状が計算精度に影響を与えるため，最適なメッシュ分割法等の研究が多く行われている．しかしながら，我々の用途のように高速に分割することや，GPU 上で分割を実装することはまれである．

また CG においても，Cutmull-Clark 法 [13] など, 少

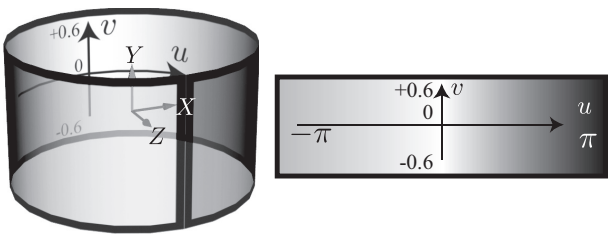


図 4　TWISTER のディスプレイ面座標 (a) と提示画像座標 (b) の対応関係

Fig. 4　Correspondence relationship between display surface(a) and output picture(b).

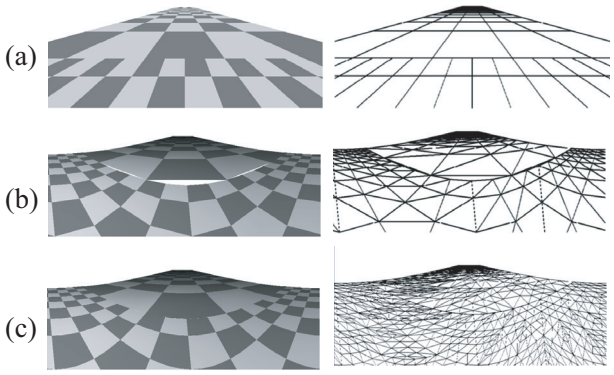


図 5　(a) 元画像（通常の透視投影で描画.）(b) 全頂点に TWISTER 投影変換を施した結果．(c) その後，ポリゴン再分割した結果.
Fig. 5　(a)Original image. (b)First, polygon vertices are relocated to their corresponding positions. (c)Next, the polygons are subdivided according to their size in order to reduce the distortion.

数の頂点データで滑らかな物体を描画するために，古くから用いられている [14, 15]．特に 2006 年に，GPU 上で頂点数をプログラマブルに増減することが可能なジオメトリシェーダ [16] が登場してからは，GPU を用いる手法が多く研究されている．例えば，プログラマブルシェーダを用いて GPU 上で分割することや，CG 上で 3 次元物体の形状を動的に操作する手法も提案されている [17]．しかし，これらの手法の多くは全てのポリゴンに対し同一の分割法で分割するために，元々小さいポリゴンでもさらに細かく分割される．一般にポリゴンが小さく，頂点が密な領域ほど急激に形状が変化し，疎な領域では元々滑らかなため，物体を滑らかにする用途においては理にかなっている分割であるが，我々の目的である TWISTER においてインタラクティブな CG コンテンツの描画する用途においては無駄が多い．

そこで我々は，GPU 上でポリゴンの大きさに応じて動的にポリゴン分割を行うアルゴリズムを用いた．本手法におけるメッシュ分割に要求される事項は，優先度が高い順に以下が挙げられる．

(i) ジオメトリシェーダ上で高速に実行できる変換であること．
(ii) 隣り合うポリゴンの共有する辺の節が重なること．
(iii) 少ない頂点数，短い辺の長さの三角形で，大きな面積を占めること．

以下ポリゴン分割のアルゴリズムを「辺の分割」，「ポリゴン内部の分割」の 2 点に分けて解説する．

辺の分割

辺の分割は条件 (ii) を満足しなければならない．しかしながら，ジオメトリシェーダ上では隣接するポリゴンの情報を得ることは困難である．

そこで提示画像の座標で辺の長さがある一定値 (L) 以下になるように，辺を均等分割する．隣接するポリゴンの間では，共有する辺の位置や長さは等しいため，それぞれの辺の節の位置も必ず一致する．

ただしジオメトリシェーダの制限上，無限に頂点数を増やすことはできないので，分割数に限界値 ($N_{\max}$) を設ける．すなわち，長さ l の辺の分割数 N は以下に定められる．なお，以下ポリゴンの 1 辺の分割数を辺分割数と定義する[2]．

$$N = \begin{cases} \mathrm{ceil}\left(\frac{l}{L}\right) & (\mathrm{If}\ \ \mathrm{ceil}\left(\frac{l}{L}\right) < N_{\max}) \\ N_{\max} & (\mathrm{Others}) \end{cases} \quad (5)$$

例えば，3 辺の長さが a, b, c の △ABC の，それぞれの辺分割数を N_A, N_B, N_C とすると，辺 BC は N_A 個に均等分割して節を作り，辺 CA, 辺 AB はそれぞれ N_B, N_C 個に均等分割する (図 6(1),(2))．

ポリゴン内部のメッシュ分割

次に (iii) を満たす最適なメッシュの並びである正三角形を敷き詰めた空間上で，長さ N_A, N_B, N_C で囲まれる領域を作成し，その内部の三角形のメッシュ形状を元にポリゴンを分割する (図 6(3))．この図で正三角形の 1 辺が，新たに生成されるポリゴンの 1 辺に対応している．次に，このメッシュ構造を実際のポリゴンに反映させる (図 6(4))．なお，新たに作成された節の位置の頂点は，投影変換関数を施し位置を計算した後に出力を行う (図 6(5)(6))．

3　実装

CG 描画のために実際に使用した PC のスペックは以下のとおりである．CPU: Xeon 3.2GHz (Nocona),

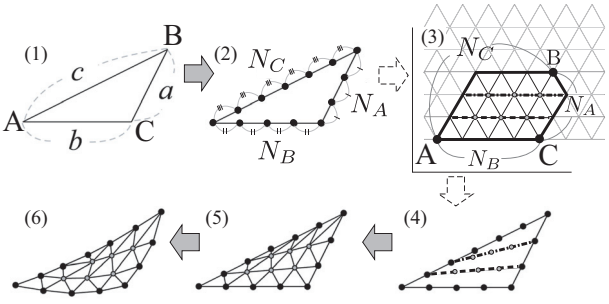


図 6　ポリゴン分割のアルゴリズム概要
Fig. 6　Polygon subdivision algorithm.

[2] ceil(x) は，x の小数点以下を切り上げる関数を表す．

RAM:2GB, GPU:NVIDIA GeForce 8800GT．プログラミング言語としてはOpenGL およびGLSL (OpenGL Shader Language) を使用した．なお変換処理の核となるジオメトリシェーダ部のコードは 400 行程度であった．また，TWISTER が表示できる画像の輝度値に制限があるため，提案手法を実装したシステムでは輝度補正の処理も含めている．

TWISTER の仕様にあわせるため，全周囲立体映像の画像サイズは片眼につき 3168×600 [pixel] とし，全周囲を前方 180 度と後方 180 度に分けて描画出力した．すなわち，1 フレームにつき 1584×600[pixel] の画像を 4 枚出力した．

以後，上記で実装したシステムを用いて提案手法の評価を行う．

4　評価

本章では，提案手法によって高精細な全周囲立体映像を実時間で描画できることを定量的に評価する．

4.1　評価 1：ポリゴン分割と歪みの関係の評価

本手法では，ポリゴンの頂点は TWISTER 投影変換で得られる位置に移動するものの，ポリゴンの内部のピクセルは直線補完した位置に描画される．そのため，小さなポリゴンに分割するとはいえ，ポリゴン内部のピクセルではずれが生じることが考えられる．そこでポリゴンの分割によって，歪みがどの程度まで軽減可能かを評価し，TWISTER においてポリゴンの辺分割数の最大値をいくつに定めればよいかを考察するために以下の実験を行った．

本実験では，ポリゴンの辺分割数を増加させるにしたがって，提示した映像の表示位置の誤差がどの程度増減するかを比較した．この際，画像 1[pixel] ごとに，実際に表示された位置と TWISTER 投影変換で理想的に表示されるべき位置との距離を表示位置誤差とし，ピクセルシェーダを用いて表示位置誤差を算出し，記録した．

実験に用いた指標ポリゴンとしては，観測者から奥行き距離 1, 3, 5[m] の位置で，TWISTER の提示画面内で表示できる最大の大きさの正三角形を配置したものを選んだ．提案した分割手法では，三角形ポリゴンの内部のうち，各頂点の位置では表示位置誤差はゼロとなり，各頂点から離れた部位であるほど表示位置誤差は大きくなる．よって，辺の長さが一定値以下の三角形のうち表示位置誤差が最も生じやすいポリゴンは，各頂点間の距離が最も大きい正三角形のポリゴンである推測される．そのため今回は，全ての CG ポリゴンの基礎となる三角形ポリゴンのうち，歪みの目立つ例として正三角形ポリゴンを選んだ．なお，正三角形の奥行き距離は TWITSER の立体視可能な位置を考慮して定めた．

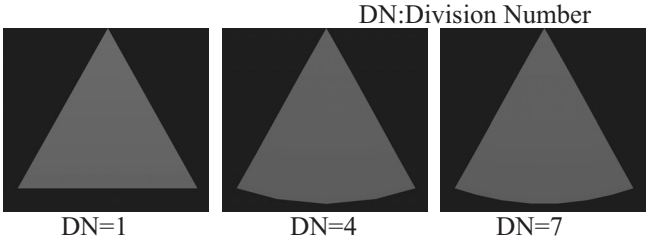


図 7　正三角形の指標ポリゴンを異なる辺分割数で描画した結果．
Fig. 7　Equilateral triangle polygon for evaluation experiment 1.

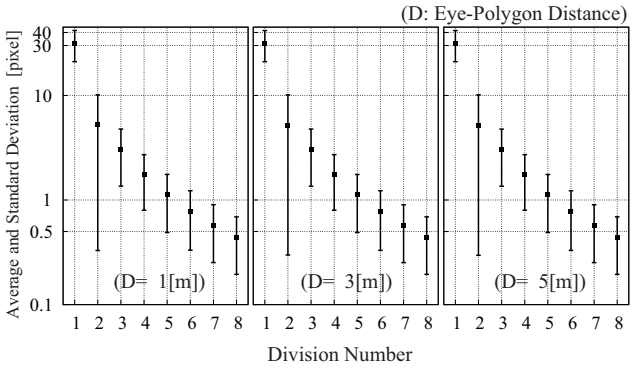


図 8　ポリゴンの辺分割数と表示位置誤差の関係．ドットは平均値，バーは標準偏差を表す．
Fig. 8　Relationship between displayed location error and division number of polygon side. A Dot shows average error, and bar meens standard deviation.

通常ならば，本手法ではポリゴンの大きさによって分割の仕方は動的に変化するが，本実験では辺分割数と表示位置誤差の関係，および本手法における最大分割数のパラメータを決定するために，全てのポリゴンに対し，同一の分割をおこなった．

結果・考察

図 7 に，奥行き距離 3[m] で，辺分割数 1, 4, 7 のときに描画した映像を示し，定量的に評価したグラフを図 8 に示す．グラフの横軸が辺分割数を表し，縦軸が表示位置誤差の平均値，及び標準偏差を表す．ポリゴンの 3 辺を同一の辺分割数 N で分割すると，ポリゴンは N^2 だけ生成される．

本結果によると，分割を行わない場合では正三角形の奥行き距離にかかわらず平均値で約 31[pixel] だけ表示位置誤差が生じた．しかし分割を行うことで，その表示位置誤差は平均 1[pixel] 以下にまで低減できることが確認できた．TWISTER の解像度は人間の視力換算で 0.148 であり，提示面のドットの表示位置が 1[pixel] 以上ずれないように，装置の機械振動や LED の発光タイミングなどを各部位を設計している．すなわち，1[pixel] 未満では，機械振動などの他の誤差要

図 9　本手法をオープンソースの FPS ゲームである Cube に用い，全周囲立体映像で描画した結果．この地点を評価 2 の実験において指標オブジェクトとして用いた．
Fig. 9　We selected the open source first person shooting game Cube and rendered it by using our method.

因もあるため，提示画像の表示位置誤差が 1[pixel] 未満であれば，TWISTER においては，十分に許容できる誤差であるといえる．特に辺分割数 7 では，標準偏差も考慮すると，提示画像のほとんどのピクセルで，表示位置誤差は 1[pixel] 以下であると推測できる．

以上より，1 辺の最大辺分割数を 7 以上になるよう設定を行うことにより，提示画面内に収まるポリゴン[3]であれば，全ピクセルに対し TWISTER 投影変換を施したときに近い画像が得られることがわかる．

また，奥行き距離が 1[m]，3[m]，5[m] の正三角形ポリゴンをそれぞれ比較すると，非常に類似した結果となった．これらのポリゴンは異なる奥行きの位置に配置しているため，表示される位置は相対的にずれているものの，画面最大となるようにそれぞれ大きさをそろえており，提示画面上ではほぼ等しい大きさとなる．本手法では，画面上でのポリゴンの大きさを基準にポリゴン分割を行っているため，上記の 3 つのポリゴンの分割の仕方がほぼ同一となる．そのためポリゴンの内部の表示位置のずれ方も類似し，表示位置誤差の結果もほぼ等しい値となったのだと考えられる．以上より，提案手法の表示位置誤差は表示オブジェクトの奥行きよりも画面上でのポリゴンの大きさに依存することがわかる．

なお，以後の実験では，ポリゴンの辺分割数の最大値を 7 となるようにパラメータを適切に設定した上で評価を行う．

4.2　評価 2：描画速度と歪みの評価

前実験では，ポリゴンの辺分割数を固定して実験を行った．しかし提案手法では高速化のためにポリゴンの辺の長さに応じて辺分割数を動的に決めている．そこで次に，ポリゴンの辺分割数を動的に決めたときに，速度と表示位置誤差がどのように変化するかを，辺分割数を固定して描画した結果と比較しながら評価する．手順は以下の通りである．

[3]TWISTER は全周囲を見ることができるため水平方向に無限に長いポリゴンも提示画面内に収まるが，ここで言う提示画面内に収まるポリゴンとは，1 辺の長さが TWISTER の縦方向のピクセル数 (600[pixel]) 以下のポリゴンと定める．

まず，指標オブジェクトを提示する．そのときの描画速度と表示位置誤差を，提案手法により描画したとき，辺分割数を固定して描画したとき，ジオメトリシェーダを用いずに頂点シェーダのみで描画したときで記録した．

指標オブジェクトとしては以下のものを選んだ．

1. TWISTER から 3[m] の距離の位置に，ポリゴン数 100 の球 (以下, Sphere100), および 1024 の球 (以下，Sphere1024) を画面最大にしてそれぞれ表示したもの．
2. TWISTER から 3[m] の距離の位置に，ポリゴン数 16301 の Stanford Bunny を表示したもの (以下，Bunny)．
3. 周囲を FPS ゲームである Cube[18] の CG 空間で覆ったもの．

TWISTER は遠隔地コミュニケーションを目的として作られている．そのため，コミュニケーションの相手となる人物や，CG 空間に配置されたオブジェクトを TWISTER を通じて見ることを考慮して，指標オブジェクトは TWISTER から 3[m] の位置に配置したものを選んだ．また指標オブジェクトは，リアルタイム CG を目的とした場合に，ポリゴン数の少ないものから多いものまで偏りがないように選定した．

また，TWISTER で CG 空間を構築し，その CG 空間内でコミュニケーションする場合も考慮し，CG 空間を指標オブジェクトに加えた．CG 空間としては，提案システムをオープンソースの FPS ゲームである Cube に実装し，Cube の中の特定の地点を選定したものを選んだ (図 9)．この地点は，近景から遠景まで含まれた景色で，この Cube の CG 空間内でポリゴン数が比較的多い地点である．

結果・考察

図 10 に結果のグラフをまとめた．左列のグラフが，表示位置誤差を表し，右側のグラフが 1 フレーム当たりの描画時間を表す．また提案手法による描画時間，およびジオメトリシェーダによるポリゴン分割を行わ

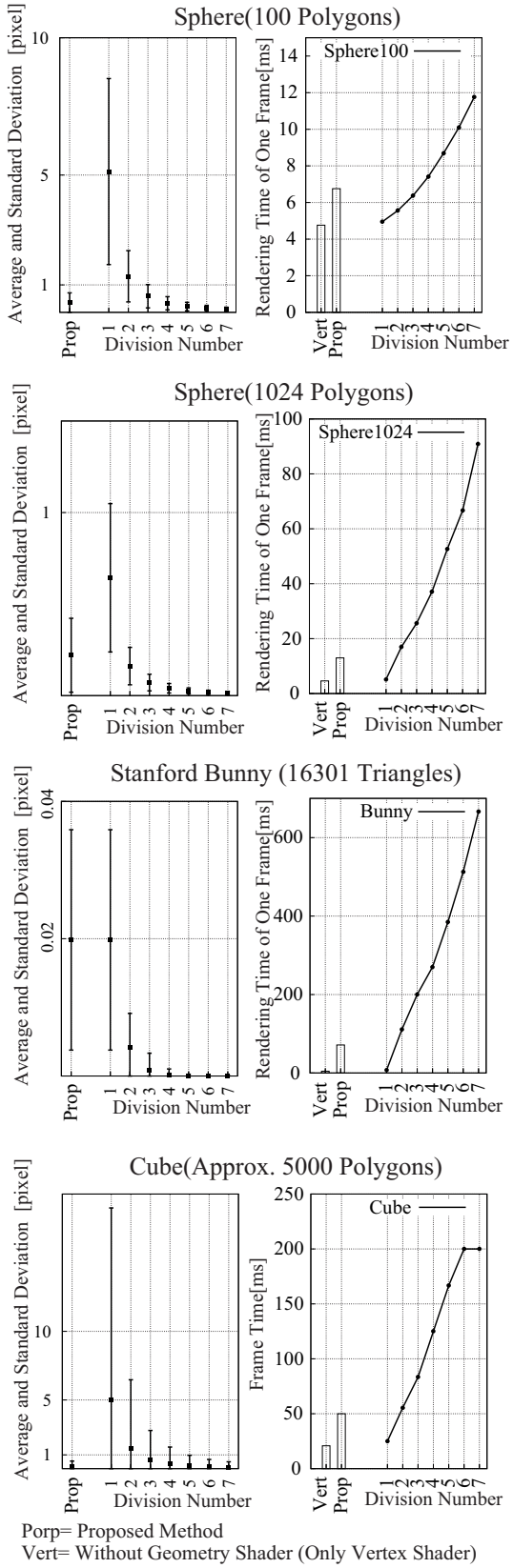


図 10　表示位置誤差および描画時間の関係．左側のグラフは表示位置誤差を表し，右側のグラフは描画時間を表す．

Fig. 10　Displayed location error and Rendering Speed. Left graph shows errors and right shows framerate.

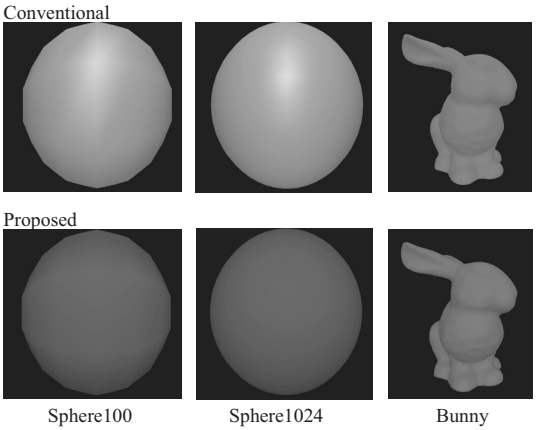


図 11　評価 2 で使用した指標オブジェクトを従来手法でスリット幅 1 で描画した結果と提案手法で描画した結果．

Fig. 11　Displayed CG Objects for evaluation experiment 2. Left images are displayed by conventional methods and Right images by proposed method.

ず，頂点シェーダのみで描画したときの描画時間をそれぞれ Prop, Vert として棒グラフで描画した．

図 11 に，描画した指標オブジェクトのうち Sphere100, Sphere1024, Bunny を描画した映像を示す．左列が従来手法でスリット幅を 1 [pixel] に設定して生成した理想的な映像を表し，右側が提案手法で描画した結果を表す[4]．この図より，従来手法の理想的な映像と，提案手法で生成した映像を比べた場合，オブジェクトの表示される位置や形状に極端な差異がないことが確認できる．

また図 10 グラフの提案手法の表示位置誤差の結果を見ると，全ての指標オブジェクトにおいて, 提案手法の表示位置誤差は，平均値が 1 [pixel] 未満となっており，提案手法により表示位置誤差を十分に軽減できていることが定量的にも確認できる．

また，辺分割数を増やしポリゴンを細かく分割した場合，表示位置誤差が減少するものの，同時に 1 フレームを描画するために必要な時間も大きくなることが図 10 右列のグラフから確認できる．そのため，例えば全てのポリゴンの辺分割数を 7 に設定した場合，提示画面に収まるポリゴンであれば表示位置誤差は 1 [pixel] 以下になるものの，同時に処理時間も大幅に増加し実時間性が失われてしまう．それに対し，提案手法では表示位置誤差の生じやすい大きなポリゴンを中心に分割を行っているため，表示位置誤差を十分に抑え，かつ処理時間も押さえられており，効率的な手法といえる．

また，提案手法の核となるジオメトリシェーダ部で

[4]なお提案手法を実装したシステムでは，3 章で述べたとおり輝度補正処理が含まれているため，従来手法を実装したシステムで描画した結果よりも映像が暗くなっている．この色の違いは実装上の差であり，本質的な問題ではない．

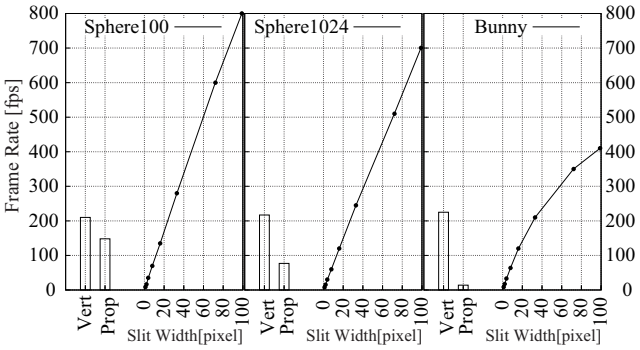
の処理時間の増加は，頂点シェーダのみで描画した場合（グラフ中の Vert）と，ジオメトリシェーダも用いて描画した場合（グラフ中の Prop）との差で確認できる．Sphere100: 2 [ms]，Sphere1024: 8 [ms], Bunny: 67 [ms]，Cube: 30 [ms] という結果となった．リアルタイム CG の目的としてはポリゴン数の多い Bunny では，提案手法での描画速度は 14 [fps] となり，実時間性にやや欠けた結果となった．しかし，より実用に近い状況の Cube の例では，この指標オブジェクトに用いた地点以外でも描画速度が 20–40[fps] となることが確認でき，提案手法によって CG 空間を実時間で描画可能であることを示す一つの実例ともなった．

Bunny では，提案手法は辺分割数 1 で固定の状態と全く同じ表示位置誤差を示している．これは Bunny を構成するポリゴンが小さく，提案手法では分割を行わなかったためだと考えられる．しかしながら，提案手法と，辺分割数を 1 に固定した場合（すなわち分割を全く行わなかった場合）とでは描画速度に大きな差がある．提案手法では約 14 [fps] に対し, 辺分割数 1 に固定の場合は 134 [fps] となった．この二つの状況では，辺分割数を決定する判定命令が有るか否かの差しかない．そのため，辺分割数を決定する部位で大きなオーバーヘッドが生じていることが分かる．一般に GPU 上の処理では，条件文などの分岐処理が非常に遅いうえ，Bunny はポリゴン数が多いため，このような結果が生じたのだと考えられる．しかし，近年 GPU 上で C 言語を動かす Cuda をはじめ，GPU を汎用的に使用できるように発展しており，この判定処理のオーバーヘッドの問題は，ハードウェアの進歩で将来的に十分解決できる問題だと考えている．

以上をまとめると，Bunny のようにポリゴン数が多い描画対象の場合，辺分割数判定処理の部位のオーバーヘッドが問題になり実時間性に欠けてくる．しかし，リアルタイム CG を念頭に置いた場合，一般的に Bunny のような精密な CG モデルは使用されず，Cube のゲームのようにポリゴン数が少ない CG 空間が構成される．このような描画対象の場合，提案手法は，表示位置誤差を減少させ，かつ描画時間も抑えた効率的な手法といえる．

4.3　評価 3：従来手法との比較

従来手法と提案手法の性能の比較を行う．ここで述べる従来手法とは，1 章で記述したように，CG 空間上で，全周囲 360 度を複数回に分けて描画することで得たスリット状の映像を並べることで，全周囲に渡りで視差を持った画像を生成する手法である．従来手法の実装は，提案手法のシステムと同様に OpenGL，及び GLUT ライブラリを用いて行った．複数のスリットを描画する際には，同一オブジェクトを多数描画する際の一般的な高速化手法であるディスプレイリストを用いた．全周囲を何分割するかは可変とし，スリットの幅を変更できるようにした．



Porp= Proposed Method
Vert= Without Geometry Shader (Only Vertex Shader)

図 12　提案手法と従来手法の描画速度の比較．線グラフは，従来手法で 1 スリットの幅とフレームレートの関係をあらわす．提案手法，およびジオメトリシェーダを用いずに分割を行わなかったときの描画時間は，それぞれ Prop,Vert の棒グラフで示している．

Fig. 12　Comparison of frame rate. Lines show relationship between frame rate and width of one slit.

従来手法ではプログラマブルシェーダを用いていないため，表示位置誤差をピクセルシェーダを用いて測定することが難しい．そこで，以下の手順で評価を行った．

1. 提案手法を用いたときの描画速度を測定する．
2. 従来手法で，映像の分割数を変えて描画したときの描画時間を測定する．
3. 提案手法と同程度の時間で描画できる従来手法のスリット幅を記録する．

指標オブジェクトとしては，前実験と同様の Sphere100, Sphere1024, Bunny を選定した．ただし前実験でもちいた Cube は本実験には含めなかった．その理由は, 従来手法を Cube に実装するためには Cube の描画処理部を大幅に書き換えて実装しなければならない上に，Cube は描画処理以外にもゲームのための処理も多く含まれているので，従来手法との比較を目的とする本実験には適さないと判断したためである．

結果・考察

結果のグラフを図 12 にまとめる．Sphere100 では，提案手法の描画速度は，従来手法のスリット幅約 18 [pixel] の状態の描画速度と同程度であり，Sphere1024 では，スリット幅約 10 [pixel] のときと同程度であった．提案手法は表示位置誤差は 1 [pixel] 以下であることに対し，従来手法のスリット幅 18 [pixel] は提案手法より大きいといえる．理想的には画像の横幅は 3156

[pixel] なので，従来手法で 3156 等分しスリット幅を 1 [pixel] として描画すれば生成された画像の表示位置誤差はゼロとなる．Sphere100 のスリット幅 1 [pixel] の状態では 8.5 [fps] となり，提案手法に比べ非常に低速である．

ただし，描画するオブジェクトのポリゴン数が増加するにつれ，提案手法と従来手法の描画速度の差異は減少する．従来手法は，描画オブジェクトのポリゴン数が増加してもそれほど速度は低下しないが，提案手法では，ポリゴン数の増加に伴い速度も大きく低下していることが, それぞれの指標オブジェクトの結果を比べることで読み取れる．例えば Sphere100 に対する Bunny の描画速度を比較すると，従来手法の Bunny の描画速度は Sphere100 の約半分になったが，提案手法では約 10 分の 1 程度になる．一方，頂点シェーダのみで描画した場合 (図 12 中の Vert) では，指標オブジェクトのポリゴン数にかかわらずほぼ一定の描画速度である．よって提案手法の，ポリゴン数増加に伴う速度低下はジオメトリシェーダ部でもたらされることが分かる．特に，前実験で明らかになった辺分割数判定処理の部位の遅延が速度低下の大きな原因の一つではないかと推測される．

以上より，ポリゴン数が Bunny 程度のオブジェクトでは提案手法と従来手法の差がほぼなくなることがわかった．しかし，Bunny よりもポリゴン数が少ないオブジェクトでは，同程度の表示位置誤差の映像を生成するために必要な時間は，提案手法の方が短いと確認した．

5　おわりに

本研究では，全周囲立体映像ディスプレイである TWISTER に対し高精細な映像を，実時間で描画するための手法を提案した．また，CG 空間内を移動できるコンテンツである Cube を TWISTER で描画し，提案手法により，CG 空間を実時間で描画可能である事の実例を示した．ポリゴン数が Bunny よりも十分少ないオブジェクトで，かつ画面内に収まるオブジェクトであれば，従来手法と比較しても，同一時間で，歪みの少ない映像を描画可能である事を確認した．リアルタイム CG を念頭に置いた場合には，一般的に Bunny のような精密な CG モデルは使用されず，Cube のゲームのようにポリゴン数が少ない CG 空間が構成される．そのため，提案手法はリアルタイム CG を描画するうえで，従来手法より効率的な手法だと考えられる．

1 章で述べたように，本手法は，既存のアプリケーションのうちプログラマブルシェーダを用いていないアプリケーションならば，従来手法よりも容易に TWISTER 用の映像として描画することが可能である．頂点シェーダやジオメトリシェーダを既に使用しているアプリケーションに対しては，本手法をそのまま適用することが難しいが，近年では頂点シェーダやジオメトリシェーダで演算した結果を再びビデオメモリ側に書き戻す Stream Output の機能も登場している [16]．そのため，頂点シェーダやジオメトリシェーダを用いたアプリケーションであっても，アプリケーション側で出力した結果を再びビデオメモリに戻し TWISTER 投影変換を施すことも可能ではないかと考えている．

今後は，本手法を改良し更なる高速化，最適化を行う予定である．今回は画面内に収まるポリゴンにおいて評価を行ったが，当然ながら画面サイズよりもはるかに大きなポリゴンの場合では，提案手法で表示位置誤差を十分に削減することは困難であると考えられる．そのため，TWISTER を用いて，TWISTER の提示画面内に収まる CG オブジェクトを観察する場合には問題は生じないが，空のポリゴンや，海のポリゴンなど，画面外にはみ出すほど大きなポリゴンを観察する場合には，歪みが目立つ結果が予想される．この点に関しては，提案手法のポリゴン分割法を改良し，画面をはみ出す大きさのポリゴンの場合には，画面内に表示されている部分のみポリゴン分割を行うことで，対処できると考えている．

提案手法では従来手法と同様に，観測者の頭部位置は TWISTER の中心部に常に一致しているという仮定を用いた．しかしながら，観測者の頭部位置と TWISTER の中心部が常に一致しているという状況は現実的ではなく，わずかな視点変位による影響が，従来から TWISTER の問題点として挙げられていた．そこで本手法をさらに応用し，観測者の頭部位置の変位を考慮した投影変換関数を用いることで，観測者の頭部が動いた際の動きに合わせたインタラクティブな CG を生成することが可能となる．今後，観測者の頭部の位置計測システムと組み合わせ，頭部位置に応じて投影変換関数を変化させる描画システムの研究開発を行うことが有効であると考えられる．

また，本論文では TWISTER を対象として評価を行ったものの，提案手法は TWISTER 以外にも様々なディスプレイに容易に応用することが可能であるとも考えている．他のディスプレイに応用するためには，CG 空間上の座標と対象のディスプレイの提示面の座標の対応関係を数式として算出し，投影変換関数を作成すればよい．特に非平面のディスプレイ面を持つシステムや TWISTER のように画像中で複数の映像視点を持つ映像を要するシステムに対しても，ポリゴンを細かく分割することで歪みの少ない映像を提示することが可能と推測できる．そのため，このような特殊

なディスプレイへの応用も有効であると考えている．

参考文献

[1] Susumu Tachi. *Telecommunication, Teleimmersion and Telexistence*. Ohmsha, IOS Press, 2003. ISBN4-274-90586-1 (Ohmsha), ISBN1-58603-338-7 (IOS Press).

[2] Susumu Tachi. TWISTER: Immersive Ominidirectional Autostereoscopic 3D Booth for Mutual Telexistence. In *Proceedings of ASIAGRAPH 2007*, pp. 1–6, Tokyo, Japan, 2007.

[3] Carolina Cruz-Neira, Daniel J. Sandin, and Thomas A. DeFanti. Surround-screen projection-based virtual reality: the design and implementation of the CAVE. In *Proceedings of ACM SIGGRAPH '93*, pp. 135–142, New York, NY, USA, 1993. ACM.

[4] 廣瀬通孝, 小木哲朗, 石綿昌平, 山田俊郎. 没入型多面ディスプレイ (CABIN) の開発. 日本バーチャルリアリティ学会第 2 回大会論文集, pp. 137–140, 1997.

[5] 橋本渉, 岩田洋夫. 凸面鏡を用いた球面没入型ディスプレイ:Ensphered Vision. 日本バーチャルリアリティ学会論文誌, Vol. 4, No. 3, pp. 479–486, 1999.

[6] 橋本渉, 吉田恭平. 壁面と凸面鏡を用いた可搬型没入ディスプレイ環境. 日本バーチャルリアリティ学会論文誌, Vol. 10, No. 2, pp. 183–189, 2005.

[7] 柴野伸之, 柏木正徳, 澤田一哉, 竹村治雄. 小型半球面スクリーンを用いた没入型視覚ディスプレイの開発. 日本バーチャルリアリティ学会大会論文集, Vol. 6, pp. 393–396, 2001.

[8] 近藤大祐, 木島竜吾. 双対レンダリングを用いた自由曲面ディスプレー. 日本バーチャルリアリティ学会大会論文集, Vol. 7, pp. 465–468, 2002.

[9] 橋本直己, 倉橋雅也, 佐藤誠. 曲面スクリーンを用いたマルチプロジェクションディスプレイにおける任意視点での歪みのない映像提示手法. 映像情報メディア学会誌, Vol. 58, No. 4, pp. 507–513, 2004.

[10] 小木哲朗, 林正紘, 藤瀬哲朗. 簡易没入型ディスプレイ CC Room の開発と映像生成手法. 日本バーチャルリアリティ学会論文誌, Vol. 11, No. 3, pp. 387–394, 2006.

[11] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. In *Proceedings of ACM SIGGRAPH '99*, pp. 299 – 306, Los Angeles, CA, USA, 1999.

[12] Kenji Tanaka, Junya Hayashi, Masahiko Inami, and Susumu Tachi. TWISTER: An Immersive Autostereoscopic Display. In *Proceedings of the IEEE Virtual Reality 2004*, pp. 59 – 66, Chicago, IL, USA, 2004.

[13] E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design*, Vol. 10, pp. 350–355, 1978.

[14] Tony DeRose, Michael Kass, and Tien Truong. Subdivision surfaces in character animation. In *Proceedings of ACM SIGGRAPH '98*, pp. 85–94, New York, NY, USA, 1998.

[15] Le-Jeng Shiue, Ian Jones, and Jörg Peters. A real-time GPU subdivision kernel. *ACM Transactions on Graphics*, Vol. 24, No. 3, pp. 1010–1015, 2005.

[16] D. Blythe. The Direct3D 10 system. *ACM Transactions on Graphics*, Vol. 25, No. 3, pp. 724 – 34, 2006.

[17] Kun Zhou, Xin Huang, Weiwei Xu, Baining Guo, and Heung-Yeung Shum. Direct manipulation of subdivision surfaces on GPUs. *ACM Transactions on Graphics*, Vol. 26, No. 3, p. 91, 2007.

[18] Cube Engine Games. Available from `http://cubeengine.com/`. Accessed March 29, 2008.

(2008 年 3 月 31 日)

［著者紹介］

城 堅誠　（学生会員）

2007 年東京大学工学部計数工学科卒業．現在，同大学大学院情報理工学系研究科システム情報学専攻修士課程在学中．バーチャルリアリティ，拡張現実感の研究に従事．

南澤 孝太　（学生会員）

2005 年東京大学工学部計数工学科卒業．2007 年同大学大学院情報理工学系研究科システム情報学専攻修士課程修了．同年同専攻博士課程進学，日本学術振興会特別研究員 (DC1)．ハプティックインタフェース，テレイグジスタンスシステムの研究に従事．2007 年日本バーチャルリアリティ学会学術奨励賞受賞．

新居 英明　（正会員）

1995 年東京工業大学大学院理工学研究科博士前期課程制御工学専攻修了．同年株式会社トキメック入社．2003 年 4 月同社退社．2003 年から 2006 年まで電気通信大学大学院電気通信学研究科博士後期課程機械制御工学専攻在籍．現在，東京大学大学院情報理工学系研究科システム情報学専攻助教．情報投影技術を利用したヒューマンインタフェースの研究に従事．

川上 直樹　（正会員）

平 8 東工大・理工・電気電子修士課程修了．平 11 東大・工・先端学際工博士課程修了．工学博士．同年東大院・工・計数工助手，平 14 東大院・情報理工・システム情報学専攻講師．バーチャルリアリティの研究に従事．

舘暲　（正会員）

昭 43 東大・工・計数卒．昭 48 同大大学院博士課程修了．工学博士．同年同大助手．昭 50 通産省工技院機械技研研究員，マサチューセッツ工科大学客員研究員を経て，平 1 東大先端科学技術センター助教授，平 4 同センター教授，平 6 同大・工・計数工教授，平 13 同大大学院・情報理工・システム情報学専攻教授．テレイグジスタンス，人工現実感の研究に従事．IEEE/EMBS 学会賞，通産大臣表彰，国際計測連合 (IMEKO) 特別功労賞，IEEE-VR Career Award など受賞．IMEKO ロボティクス会議議長，計測自動制御学会会長，日本バーチャルリアリティ学会初代会長．