Foveated Rendering による動的被写界深度効果の検討

中川伶丞[1]　植村　匠[1]　尾島修一[1]

概要：本研究では，被写界深度効果を付与しつつ，フレームレート低下を起こさないために，Foveated Rendering を導入した VR システムを提案する．提案した VR システムのリアルタイム性の評価について報告を行う．

キーワード：コンピュータグラフィックス，注視点，被写界深度，フォービエイティッドレンダリング

Study on Dynamic Depth of Field Effect Using Foveated Rendering

RYOSUKE NAKAGAWA[†1]　TAKUMI UEMURA[†1]
SHUICHI OJIMA[†1]

Abstract: In this research, we propose a VR system that introduces Foveated Rendering in order to prevent the frame rate from decreasing while giving the depth of field effect. We report on the evaluation of the proposed VR system in real time.

Keywords: Computer Graphics, Gazing Point, Depth of Field, Foveated Rendering

1. はじめに

VR(Virtual Reality)技術は近年急速な発展を遂げている．特に，「VR 元年」の 2016 年前後から現在に至るまで，多くの Virtual Reality に関する技術が発表されている．中でも HMD の技術は大きく発展しており，HMD の欠点の一つとされてきた，HMD 本体の価格や，重量，解像度や，表示視野角等, 様々な項目で性能の向上が図られている. 更には，HMD のワイヤレス化や AR(Augmented Reality)製品の登場により，HMD 技術の発展は VR 及び HMD の市場規模拡大の一端を担っている．

しかし，VR 酔いの抑制や没入感の向上，コンピュータへの高負荷など，VR 技術には未だ多くの解決すべき課題が残されており，これらの課題をいかに改善できるかが更なる VR の普及及び発展の鍵となっている．これらの課題を解決する手段はいくつか存在するが，被写界深度効果や Foveated Rendering 技術の導入は，課題解決のための有効な手段の一つであると考えられている．以下，被写界深度効果，Foveated Rendering について説明する．

1.1 被写界深度効果

被写界深度とは，画像・映像においてピントが合っていると判断できる範囲のことである．被写界深度効果とは，ピントが合っている領域以外をぼかすことで奥行き感を与え没入感を向上させる効果で，Hillarie らの研究により有効性が示されている[1]．

ピントが合っている領域以外をぼかすためには，錯乱円を考えることが重要である．カメラにおいてピントが合っている時，ある被写体の点はカメラの撮像面上に点として収束する．実際には撮像面は 1 画素以下の大きさのものは検出できないので，錯乱円の大きさが 1 画素以下であればピントが合っていると見なすことができる．この錯乱円を許容錯乱円という．図 1 に示すように，前側被写界深度と後側側被写界深度の間は，撮像面上で許容錯乱円以下の大きさで像を結ぶので，この領域が一般的に被写界深度と呼ばれる．したがって，前側被写界深度より遠いか，または，後側被写界深度より近い領域にある物体はぼかす必要がある．

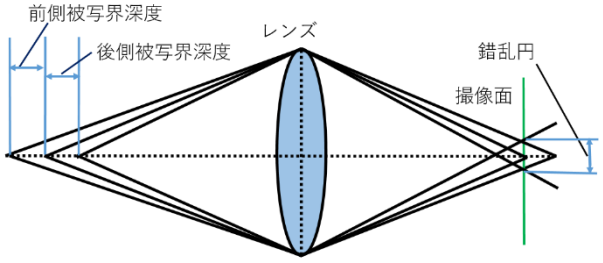

図 1　被写界深度と錯乱円の関係図

1.2 Foveated Rendering

眼の中心には中心窩と呼ばれる領域がある中心窩で捉えることのできる範囲はごくわずかであるが，生物は何かを見るときに主に中心窩を使用して見ている．なぜならば，中心窩で捉えた領域のみ高精細に見ることができ，中心窩付近の中心視野から周辺視野に向けて離れる程，精細に見ることができなくなっていくからである．Foveated

1 崇城大学
Sojo University

Rendering は，このような眼の構造を利用した技術である．視線追跡装置を利用して，ユーザーが凝視している凝視点のみを高解像度で表現し，凝視点から視線の外周に向けて離れるほど，低解像度で表現される．結果として，図 2 に示すように周辺視野は低解像度で描写したとしても，中心視野は常時高解像度で描写しているので，ユーザーにとって VR で見ている映像は高解像度なものとなり，CG のレンダリングを簡素化し高速表示を可能とする．

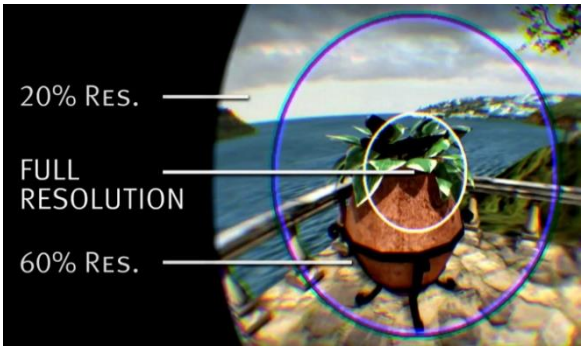

図 2　Foveated Rendering の概念図[2]

本研究では，被写界深度効果を付与しつつ，fps(frame per second)を低下させないために Foveated Rendering を導入した VR システムを提案する．VR システムの具体的な提案手法と，リアルタイム性の評価の結果について報告を行う．

2. 関連研究

CG の立体感を向上させるため，被写界深度効果を導入した研究として吉柳らの研究がある[1]．吉柳らは奥行き知覚の強調，および VR の現実感の向上を試みるため，両眼視差および輻輳を考慮した没入型 VR システムにおいて，ユーザーの視線に応じて動的に変化する被写界深度効果の提案を行った．

また，Vive Foveated Rendering[3]や Tobii Spotlight Technology[4]など，Foveated Rendering を使用してコンピュータへの負荷を低減させ，リアルタイム性の向上を試みる技術も多く存在する．

以下，動的被写界深度効果の実装手法と Foveated Rendering の手法について述べる．

2.1 動的被写界深度効果の実装方法

吉柳らは，リアルタイムで視線に応じて被写界深度を変化させる VR システムを提案した際に，IPD(Immersive Projection Display)を使用し VR システム及びリアルタイム性の検証を行っている．以下に吉柳らに提案された VR システムの実装手法を示す．

① 視線追跡装置を使用し，時間的に変化する三次元空間の凝視点位置を測定する．
② 凝視点の位置を基に錯乱円の直径を計算する．
③ 計算した錯乱円の直径に基づきぼけを計算し，ぼかした結果の画像を出力する．

上記の実装手法に基づき，吉柳らはリアルタイムに適用可能な動的被写界深度効果を実装したが，表示オブジェクト数が増加するに伴い，コンピュータへの負荷が高くなりフレームレート数を維持することが困難となることを示した．したがって，動的被写界深度効果の画質を向上させるためには CG 表示時にコンピュータへかかる負荷を低減する必要がある．

2.2 Foveated Rendering の実装方法

Foveated Rendering の実装方法はいくつか存在するが，ここでは Guenter らの手法[5]を紹介する．Guenter らの手法では，図 4 に示すように 3 層のレイヤー構造に分けて，レイヤー毎に別の処理をした後，各レイヤーをバランスよく統合する．1 層目から 3 層目にかけてレイヤーが大きくなるにつれて解像度を低下させて行く．

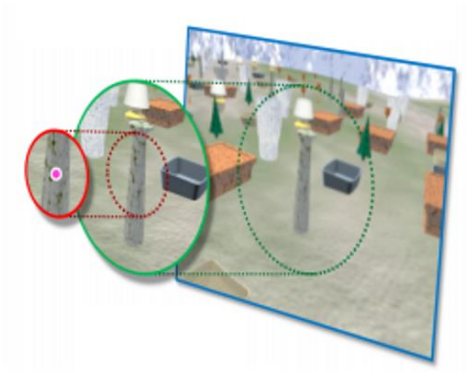

図 3　レイヤー構造の Foveated Rendering の概要図

3. 提案手法

本研究では，動的被写界深度効果を適用させつつ，フレームレート数を低下させないために，リアルタイムで視線に応じて被写界深度を変化させる VR システムに Foveated Rendering を適用する．すなわち，周辺視野の処理を部分的に削除し，VR 体験時のコンピュータにかかる負荷の軽減を図る．以下，本研究での動的被写界深度効果の実装方法，ポアソンディスクサンプリングについて述べる．

3.1 動的被写界深度効果の実装方法

本研究では，吉柳らの手法[**エラー! ブックマークが定義されていません。**]と Riguer らの手法[6]を参考にすることで，動的被写界深度効果の導入を行った．導入の手順は大きく 3 つのステップに分けることができる．

① 錯乱円直径を求める．
② 錯乱円直径を基にポアソンディスクサンプリングを行い，サンプリングするピクセルの座標を求める．

③　求めたピクセルの座標を基に分散を求め，ガウスぼかしを使用する．

3.2 ポアソンディスクサンプリング

ポアソンディスクサンプリングは，2 次元の平面上において，全ての点同士が一定距離以上離れているサンプリングである[7]．ポアソンディスクサンプリングのアルゴリズムを以下に示す．

① 確定点リスト・将来性点リストを用意する．
② 一様乱数を基に抽出したランダムな点を一つ生成し，この生成した点を基準点とする．（基準点は確定点リスト・将来性点リストに入る．）
③ 基準点の半径$r \sim 2r$以内に点を生成する．（10～30 回程度）
④ 新しく生成した点と確定点の間隔がrよりも，
- 近い：新しく生成した点は破棄する．
- 同じ，または遠い：確定点リスト・将来性点リストに入る．

⑤ もしステップ③～④の処理が終わった際に，新しい点が一つも生成されていなかった場合には，基準点は確定点リスト・将来性点リストから削除される．
⑥ 次の基準点を将来性点リストの最後尾として，ステップ③～⑤を繰り返す．
⑦ 将来性点リストが空になった時点で終了．

上記のアルゴリズムにおいて，将来性点リスト・確定点リストは，サンプリングした点の座標を格納する配列である．確定点リストには最終的に座標として決定した点を格納する．また，将来性リストには，確定点リストに入るかを判別する候補の点の座標を格納する．また，乱数の範囲は 0 からサンプリングする円の半径とする．

本研究では，ガウシアンフィルタで使用する分散を求めることに使用する．本研究でのポアソンディスクサンプリングの使い方の特徴として，以下に二つ挙げる．

① サンプリングする点を増やすことでぼけの品質を上げることができる．
② サンプリングする円の範囲により，ぼけは大きくなる．この時，錯乱円直径を，サンプリングする範囲の円の大きさとする．

4. 実験

実験では，リアルタイム性の検証のため，fps の計測を行う．比較検討する 4 種類のレンダリングは以下の通りである．これらの間での fps の比較を行い，動的被写界深度効果と Foveated Rendering の併用が，リアルタイム性を向上させることを示す．

（条件 A）通常のレンダリング
（条件 B）被写界深度効果のみを適用したレンダリング
（条件 C）Foveated Rendering のみを適用したレンダリング
（条件 D）被写界深度効果及び Foveated Rendering 及び被写界深度効果を適用したレンダリング

4.1 実験環境

本研究でのプログラムの実装は, Unity を用いて行った．使用する視線情報は，カメラが向いている向きを視線の方向として使用し，この時の視線の原点は，カメラ位置の空間座標を設定した．

動的被写界深度効果は，3 章の手法と Unity のポストプロセス機能を利用し導入した．Foveated Rendering の導入には，Unity の Asset Store で Vive Software から配布されている Vive Foveated Rendering を使用した．このプラグインは§2.2 で述べたようなレイヤー構造を使用している．そこで，本研究ではレイヤーの正規化された半径を，1 層目を 0.25，2 層目を 0.35，3 層目を 1.0 となるように設定した．

実験を行うにあたり，用意した環境を表 1 に示す．図 5 に示したように 9 つのオブジェクトを適当に配置した後，オブジェクト位置とカメラ位置を固定した状態でレンダリングを行った．実験に使用したコンピュータの性能を表 2 に示す．

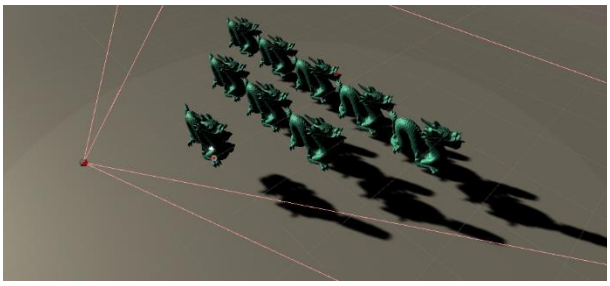

図 4　実験時のオブジェクト配置図

表 1　実験環境

解像度	1200×600 pixel
オブジェクトの頂点数 (1 個あたり)	2,614,242
オブジェクトの三角形面数 (1 個あたり)	871,414
設置したオブジェクトの数	9

表 2　コンピュータの性能

プロセッサ	Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz 3.19GHz
GPU	NVIDIA GeForce RTX 2060 super 6GB
RAM	16GB

本実験では, fps の平均値･最高値･最低値で比較を行う．また，各レンダリングでの fps の変動をグラフ化し，fps の安定性などを客観的に評価する．

実験は，カメラ位置を変えて，それぞれの位置で 4 種類のレンダリングで実験を行う．カメラ位置を変更することにより，カメラに映るオブジェクトの三角形面数と頂点数が変動し，レンダリング処理に必要な負荷も変動する．そのため，計測 fps とレンダリングするポリゴン数の関係性を確認することが可能である．

実験を行うカメラ位置と，それぞれの位置でのレンダリングする三角形面数，頂点数を表 3 に示す．カメラ位置は描写するオブジェクトのポリゴン数を基に決定した．これらの条件で，700 フレーム分の fps を取得した．その後，fps の推移，平均 fps，最高 fps，最低 fps の出力を行った．

表 3　カメラ位置ごとの三角形面数と頂点数

	三角形面数	頂点数
Position 1	5480 万	2760 万
Position 2	6260 万	3150 万
Position 3	7480 万	3760 万

4.2 実験結果

Position 1 から Position3 での平均 fps，最高 fps，最低 fps をそれぞれ表 4 から表 6 に示す．

表 4　Position 1 での 4 条件の実験結果

	平均 fps	最高 fps	最低 fps
条件 A	82	186	38
条件 B	74	167	22
条件 C	114	265	30
条件 D	81	182	37

表 5　Position 2 での 4 条件の実験結果

	平均 fps	最高 fps	最低 fps
条件 A	73	179	35
条件 B	59	161	22
条件 C	60	168	27
条件 D	73	181	36

表 6　Position 3 での 4 条件の実験結果

	平均 fps	最高 fps	最低 fps
条件 A	58	179	6
条件 B	45	158	20
条件 C	59	165	33
条件 D	45	96	21

結果として，表 4 では，Foveated Rendering のみを適用した条件 C は, 通常のレンダリングである条件 A と比較して 3 割ほどの fps の上昇が見られた．また，被写界深度効果を適用した状態で Foveated Rendering を適用した条件 D では，被写界深度効果のみの条件 B より，1 割ほどの fps の上昇が見られた．Foveated Rendering を導入することで，通常のレンダリングに近い CG 再生がなされていることがわかる．これは，表 5 でも，同様である．

次に，表 6 では，被写界深度効果のみを適用しレンダリングした条件 B の fps と，被写界深度効果及び Foveated Rendering を適用してレンダリングした条件 D の fps を比較した際に，fps の差がないことがわかる．被写界深度効果のみ適用時は，コンピュータへの負荷が最も高くなるにも関わらず，Foveated Rendering を適用しても負荷が低減されていない．

これらの結果により，カメラ位置の違い，すなわち処理するポリゴン数が多すぎると，被写界深度効果に対する Foveated Rendering の改善効果がほとんど無いと考えられる．

5. おわりに

本研究では，リアルタイム性を維持しつつ動的被写界深度効果により没入感の向上を図ることを目的に，吉柳らの提案した動的被写界深度効果を付与しつつ，Foveated Rendering を導入した VR システムについての検討を行った．結果として，ポリゴン数が少ないときは，コンピュータへの負荷は低減され，fps の低下を抑えることができたが，ポリゴン数が多くなると Foveated Rendering による効果が弱くなることがわかった．

今後は，視線を動かした場合の fps の比較と，ポリゴン数と Foveated Rendering による負荷低減の程度との関係を明らかにすることが課題である．

参考文献

[1] 古柳俊佑, 蔡東生. 没入型 VR における動的被写界深度効果の実装と評価. 情報処理学会研究報告, 2012, vol. 2012-CG-146, no.26, p. 1-6.

[2] Sebastien Hillarie, Anatole Lecuyer, Remi Cocot and Gery Casiez., Using an Eye-Tracking System to Improve Camera Motions and Depth-of-field blur effects for first-person navigation in virtual environments. Proceedings of the ACM symposium on Virtual reality software and technology, 2007, p. 203-206.

[3] “フォービエイテッド・レンダリング（Foveated Rendering）“. https://www.moguravr.com/terms/index-h/terms-47447/, (参照 2020-2-4).

[4] “Vive Foveated Rendering”.

https://assetstore.unity.com/packages/tools/particles-effects/vive-foveated-rendering-145635, (参照 2020-1-16)
[5] “Tobii Introduces Tobii Spotlight Technology™ - Tobii.com”. https://www.tobii.com/group/news-media/press-releases/2019/7/tobii-introduces-tobii-spotlight- technology/, (参照 2020-2-4)
[6] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, John Snyder. Foveated 3D Graphics. ACM SIGGRAPH Asia, 2012.
[7] Guennadi Riguer, Natalya Tararchuk and John Isidroro. Real-Time Depth of Field Simulation ShaderX2: Shader Programming Tips and Tricks with DirectX 9. Wolfgang Engel, ed., Wordware, pp. 529-556, 2003.
[8] Robert Bridson. Fast Poisson Disk Sampling in Arbitrary Dimensions. ACM SIGGRAPH 2007 sketches, 2007, p. 22.