H-031

自由視点画像を高速に生成するためのオンラインレンダリング

Online Rendering for Fast Generation of Free Viewpoint Images

荒川 雄登　置田 真生　伊野 文彦
Yuto Arakawa　Masao Okita　Fumihiko Ino

大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

1　はじめに

GRAF（Generative Radiance Fields）[1] とは，2 次元画像から物体の 3 次元構造（ボリューム）を推定する NeRF（Neural Radiance Fields）[2] を基にした機械学習モデルである．GRAF および NeRF は，推定により得られる 3 次元構造を基に，自由視点からの 2 次元画像を生成できる．GRAF の特長の一つとして，学習データセットに存在しない色や形状を持つ物体の 3 次元構造を推定できる点が挙げられる．例えば，ある患者の医用データが学習データセットになかったとしても，GRAF を使えば，その患者の 3 次元 CT 像を他の患者の X 線投影像から生成でき，患者の被爆量を削減できる [3]．

GRAF は，以下に示す 2 つのステップで自由視点画像を生成する．以降では，画像およびボリュームのサイズをそれぞれ $N \times N$ 画素および $V \times V \times D$ ボクセルとする（D は奥行き方向）．

1. ボクセル値の推論：ボリュームを構成するすべて（V^2D 個）のボクセルの値を得る．
2. ボリュームレンダリング：自由視点からのレンダリングにより $N \times N$ 画素の 2 次元画像を生成する．

これらのステップは計算量が多く，GPU（Graphics Processing Unit）を用いて加速したとしても，視点 1 つにつき数秒の生成時間を要している．したがって，実時間の画像生成のために，これらのステップの高速化が求められている．

そこで本研究では，GRAF における自由視点画像生成の高速化を目的として，ボクセル値の推論回数を削減するオンラインレンダリング手法を提案する．提案手法は，すべてのボクセル値を推論するのではなく，自由視点画像の生成に必要なボクセル値のみを推論する．さらに，古典的な高速化手法である ERT（Early Ray Termination）[4] をレンダラに組み込むことで，レンダリング結果に対して寄与の少ないボクセルに対する累積計算を打ち切り，処理全体の高速化を図る．

ダミーデータを用いた実験の結果，提案手法の生成時間は推論回数に比例し，提案手法は GRAF と比較して最大で 3.1 倍ほど高速であった．一方，実データに対しては，提案手法は人の顔のモデルで 9%高速であったが，車のモデルで 3%低速であった．

2　関連研究

NeRF の高速化を目的とする関連研究 [5, 6, 7] は存在する．KiloNeRF [5] は，2 次元画像を実時間で生成するために，レンダリングの高速化技術および機械学習のデータ削減技術を併用している．前者は Empty Space Skipping [8] であり，物体が存在しない空白領域における累積計算を省く．後者は蒸留 [9] であり，学習モデルのサイズを削減する．さらに，推定のためのニューラルネットワーク（NN：Neural Network）を複数の小さな NN に分割することにより，推論を並列処理し，高速化を図る．

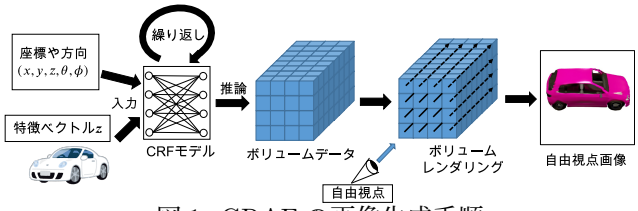

図 1: GRAF の画像生成手順

PlenOctrees [6] は，NeRF が推定した 3 次元構造を 8 分木で表現することにより，空白領域を粗い粒度で保持する．この不均一なデータ構造は，NeRF における推論の一部を省略し，画素生成を高速化できる．

SNeRG [7] は，物体の色情報を事前にボクセルに格納し，視点依存の効果のみを NN を用いて推論する．この手法により，NN による推論時間を短縮し，ボリュームレンダリングを高速化する．

これらの高速化手法は，自由視点画像を生成する前に物体の位置が確定することを前提としている．しかし，GRAF の生成対象は学習データセットにない物体であり，画像生成前に物体の位置を確定できない．したがって，これらの手法はそのまま GRAF に適用できない．

3　準備：GRAF および ERT

GRAF [1] は，条件付き敵対的生成ネットワーク [10] の 1 種であり，NN がボリュームを構成するボクセルの値 $(\mathbf{c}, \sigma)$ を一つずつ推論する（図 1）．ここで，c はボクセルの色を表し，σ はボクセルの密度を表す．NN として，NeRF を拡張した CRF（Conditional Radiance Fields）を採用していて，CRF への入力は以下の 3 点である．

- ボクセルの座標 (x, y, z)
- ボクセルを貫く視線の方向 (θ, ϕ)
- 生成する物体の形状や色を表す特徴ベクトル $\mathbf{z}$

GRAF は，以下に挙げる手順で自由視点画像を生成する（アルゴリズム 1）．

1. ボクセル値の推論（1〜5 行目）．ボリュームを構成するすべて（V^2D 個）のボクセル値を得るために，ボクセルの座標や視線の方向 (x, y, z, θ, ϕ) を変えなが

アルゴリズム 1 既存手法による自由視点画像の生成

Input: 特徴ベクトル $\mathbf{z}$ および学習済 CRF モデル M
Output: 自由視点画像
1: **for all** ボリューム上の座標 (x, y, z) **do**
2: **for all** 画素ごとの視線 (θ, ϕ) **do**
3: $M(x, y, z, \theta, \phi, \mathbf{z})$ によるボクセル値の推論;
4: **end for**
5: **end for**
6: **for all** 画素ごとの視線 (θ, ϕ) **do**
7: 視線が貫くボクセルを手前から奥の順に特定;
8: **for** 手前から奥のボクセルの座標 (x, y, z) **do**
9: ボクセル値を用いた累積計算;
10: **end for**
11: **end for**

ら学習済 CRF モデル M に与え，ボクセル値 $(\mathbf{c}, \sigma)$ の推論を反復する．

2. ボリュームレンダリング（6〜11 行目）．2 次元画像を構成するすべて（N^2 個）の画素値を得るために，画素ごとに視線 $r = (\theta, \phi)$ を飛ばし，r が貫くボクセルの値を手前から順番に累積計算する．

したがって，GRAF における推論の回数はボリューム内のボクセルの個数 V^2D と一致する．

ERT [4] は，ボリュームレンダリングの古典的な高速化手法である．この手法は，手前から奥へのボクセル処理において，不透明度が閾値 ϵ に達した場合に，以降のボクセルは画素値への影響が小さいと考えて累積計算を省く．アルゴリズム 1 に対しては，9 行目の直後に ERT の処理を追加すれば，8 行目における for 文の反復を停止できる．

4 提案手法

ERT を既存手法に単純に適用した場合，省略できる計算はレンダリング処理（6〜11 行目）に限定されてしまい，1〜5 行目の推論回数は削減できない．そこで，提案手法はボクセル値の推論および累積計算を交互に実行することで，ERT による計算の打ち切り時に推論をも打ち切ることを狙う．すなわち，必要に応じてボクセル値を推論するオンラインレンダリングを ERT と併用する．

アルゴリズム 2 に提案手法の疑似コードを示す．提案手法は，先に挙げた交互の実行を実現するために，推論のためのループ（アルゴリズム 1 の 1〜5 行目）を，レンダリングのためのループ（アルゴリズム 1 の 6〜11 行目）に統合している．結果，前者はアルゴリズム 2 には存在せず，4 行目にボクセル値を得るための推論が追加されている．提案手法は，得られたボクセル値を累積計算したのち，ERT を適用するための if 文を処理する．その時点で不透明度が閾値 ϵ に達していれば，現在の視線に関する累積計算を省略し，次の視線に処理を進める．

なお，簡単のため，アルゴリズム 2 ではボクセル 1 個に対する推論および累積計算を交互に実行している．実際の実装では，累積計算の直前に CPU・GPU 間のデータ転送が生じるため，このオーバヘッドが全体性能を低

アルゴリズム 2 提案手法による自由視点画像の生成

Input: 特徴ベクトル $\mathbf{z}$ および学習済 CRF モデル M
Output: 自由視点画像
1: **for all** 画素ごとの視線 (θ, ϕ) **do**
2: 視線が貫くボクセルを手前から奥の順に特定;
3: **for** 手前から奥のボクセルの座標 (x, y, z) **do**
4: $M(x, y, z, \theta, \phi, \mathbf{z})$ によるボクセル値の推論;
5: ボクセル値を用いた累積計算;
6: **if** 不透明度が閾値 ϵ に達したか **then**
7: break;
8: **end if**
9: **end for**
10: **end for**

表 1: 実験環境

項目	仕様
CPU	AMD EPYC 7232P
GPU	NVIDIA RTX A6000
GPU メモリ	48GB
OS	CentOS 7.9
Python	3.8.15
PyTorch [11]	1.9.0
CUDA [12]	11.1.1

下させる恐れがある．つまり，ERT により打ち切りできるボクセル数および CPU・GPU 間のデータ転送量はトレードオフの関係にある．そこで，性能を高めるために，提案手法は g 個のボクセルに対する推論および累積計算を交互に実行する．このタスク粒度 g の適切な値は，トレードオフの関係をもとに実験的に決める．g が小さければ，ERT による早期の打ち切りを実現できるが，CPU・GPU 間のデータ転送量が増えてしまう．

5 評価実験

実験では，オンラインレンダリングならびに ERT による高速化効果を評価することを目的として，1 つの自由視点に要する画像生成時間に関して，提案手法および既存手法を比較した．比較対象は，オンラインおよび ERT の有無に関して組み合わせた手法であり，(1) 提案手法（オンラインかつ ERT），(2) オンラインのみ，(3) ERT のみ，(4) いずれもなし（GRAF）である．なお，CRF に与える特徴ベクトル $\mathbf{z}$ は，すべての手法において同一のものを与えた．データサイズは $N = 512$，$V = 512$ および $D = 64$ であり，閾値は $\epsilon = 0.99$ である．表 1 に，実験に用いた環境を示す．

5.1 タスク粒度の選定

適切なタスク粒度 g の値を実験的に決めるために，$1 \leq g \leq 16$ の範囲で値を変えながら画像生成時間を計測した（図 2）．なお，実験にはダミーデータを用い，ボクセルの不透明度を調節することにより，累積計算時の不透明度が閾値 ϵ に達するまでのボクセルの数 d $(1 \leq d \leq D)$

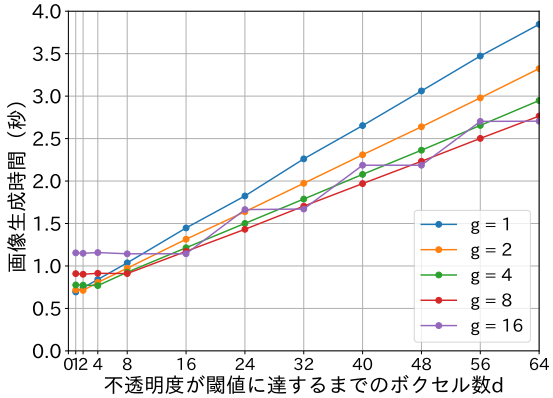

図 2: タスク粒度 g と画像生成時間の関係

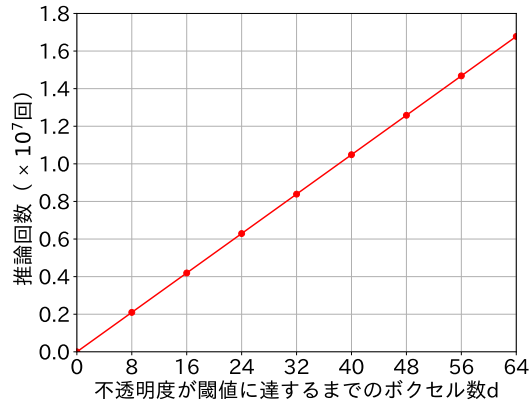

図 3: 不透明度が閾値に達するまでのボクセル数 d と最小の推論回数の関係

を制御した．例えば，$d = 12$ であれば，視線ごとに手前から 12 個のボクセルを参照すれば，ERT による累積計算の打ち切りが可能であることを意味する．累積計算の打ち切りが d 個のボクセルで生じた場合，提案手法における全体の推論回数は N^2d となる（図 3）．

一般に，g の値が小さければ，より早期に累積計算を打ち切ることができるが，ERT に起因するオーバヘッドは大きい．このトレードオフの関係を図 2 に確認できる．例えば，$g = 1$ の場合，$d = 1$ において画像生成時間が $g > 1$ の場合よりも短いが，$d \geq 16$ においては最長である．つまり，早期に累積計算を打ち切れるデータに対して ERT は効果的であるが，打ち切りが遅いデータに対しては，CPU・GPU 間のデータ転送を含め，不透明度を確認するためのオーバヘッドが大きい．

一方，$g = 16$ の場合，$d = 16$，32，48，64 において画像生成時間が最短であるが，$d < 16$ において最長である．つまり，累積計算の打ち切りが遅いデータに対しては，不透明度を頻繁に確認することなく大きなタスク粒度で処理できるが，早期に累積計算を打ち切れるデータに対しては，大きなタスク粒度が原因で打ち切りできることを直ちに検出できない．なお，$d = 24$，40，56 において，$g = 16$ の画像生成時間は最短ではない．この理由は，タスク粒度の大きさ g にあり，g 個ごとにしか

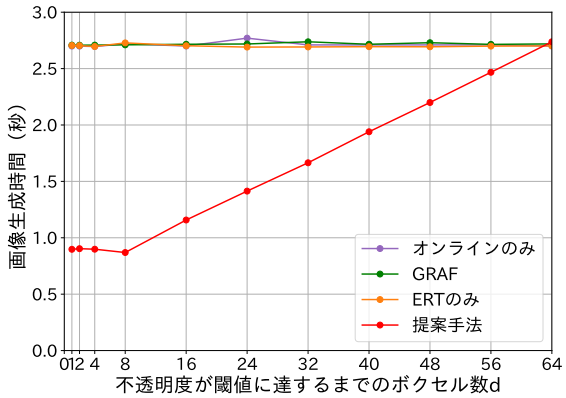

図 4: ダミーデータに対する画像生成時間（$g = 8$）

ERT による打ち切りを実現できないからである．結果，$d = 24$，40，56 の生成時間はそれぞれ $d = 32$，48，64 の場合と同等となる．

上記で挙げたトレードオフの関係をもとに，以降では $g = 8$ として性能を計測した．具体的には，$d \geq 8$ において $g = 8$ もしくは $g = 16$ の画像生成時間が最短であること，また，$d < 16$ において $g = 8$ は $g = 16$ よりも 22%高速であることから，$g = 8$ を選択した．

5.2 推論回数の削減による高速化効果

ボクセル値の推論回数とともに画像生成時間を削減できることを確認するために，引き続きダミーデータを用い，提案手法および既存手法の画像生成時間を計測した（図 4）．$d \geq 8$ において，提案手法のみが d に比例した画像生成時間を実現している．つまり，ボクセルの推論回数を削減することにより，画像生成時間を短縮できている．なお，$d < 8$ $(= g)$ において提案手法の画像生成時間が一定である理由は先に述べた通りである．

一方，残り 3 つの手法に関しては，画像生成時間は d に依存することなく，2.7〜2.9 秒の範囲に収まっている．特に，ERT のみの手法は，累積計算を打ち切ったとしても画像生成時間を短縮できていない．したがって，ERT だけでなくオンラインレンダリングとの併用が画像生成時間の短縮には不可欠である．結果，$d = 8$ において提案手法は既存手法と比べて 3.1 倍の高速化を実現した．

なお，$d = 64$ において提案手法は他の手法よりも 1.4% 低速である．このとき，いずれの既存手法においてもボクセルの推論回数は同一（N^2D 回）である．しかし，CPU・GPU 間のデータ転送回数は，既存手法が 1 回であるのに対し，提案手法では D/g 回に増える．したがって，CPU・GPU 間における転送量の増大が全体の性能低下を引き起こした．

5.3 実モデルに対する有用性

実データに対する有用性を確認するために，学習済モデルとして人の顔ならびに自動車を生成するモデルを用意した（図 5）．人の顔のモデルに対しては（表 2），提案

表 2: 実モデルを用いた比較

モデル	比較項目	提案手法	オンラインのみ	ERT のみ	GRAF [1]
人の顔	推論回数（回）	13,631,488	16,777,216	16,777,216	16,777,216
	生成時間（秒）	3.38	3.78	3.70	3.69
自動車	推論回数（回）	16,777,216	16,777,216	16,777,216	16,777,216
	生成時間（秒）	2.90	2.90	2.83	2.83

(a) 人の顔モデル

(b) 自動車モデル

図 5: 実モデルを用いて生成した自由視点画像

手法は他の手法と比較してボクセル値の推論回数を 19%削減できたが，画像生成時間は 9%の短縮にとどまった．この際の削減時間は 0.4 秒であり，ダミーデータに対して推論回数を同程度削減できたときの削減時間 0.38 秒と同様の結果である．したがって，提案手法は削減した推論回数に応じて画像生成時間を短縮できたが，初期化処理など，推論回数に依存しない部分の処理が改善効果を低下させていた．

一方，自動車のモデルに対しては，提案手法はボクセル値の推論回数を削減できなかった．結果，画像生成時間は既存手法より最大で 0.1 秒増加した．この理由を調べるために，ボクセルの持つ不透明度 α $(0 \leq \alpha \leq 1)$ の分布をモデルごとに調べたところ（表 3），自動車のモデルでは，空白のボクセルが 94.6%を占めていた．ボクセルの大半が空白であったことから，すべての視線で累積計算の打ち切りが発生しなかった．これら空白ボクセルに対する累積計算を省くために，8 分木のようなデータ構造を組み込むことが必要である．

6 まとめと今後の課題

本報告では，GRAF における自由視点画像生成の高速化を目的として，ボクセル値の推論回数を削減する手法を提案した．提案手法は，オンラインレンダリングおよび ERT を併用してボクセル値の推論回数を削減する．具体的には，ボクセル値の推論とその累積計算を交互に処理することで視線の打ち切り後の推論を回避する．

実験の結果，提案手法はボクセル値の推論回数とともに，画像生成時間を削減でき，GRAF に対して最大で 3.1 倍の高速化を実現した．ただし，空白領域を含む実データに対しては，ERT による削減効果を確認できなかった．今後の課題として，空白領域に着目した高速化が挙げられる．

謝辞

本研究の一部は，JSPS 科研費 JP23H03371 の補助による．

表 3: 実モデルにおける不透明度 α の分布（%）

α の値	人の顔	自動車
0	52.3	94.6
(0, 0.9]	6.0	1.1
(0.9, 1]	41.7	4.3

参考文献

[1] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. In *Proc. 33th Neural Information Processing Systems (NeurIPS 2020)*, December 2020. 13 pages.

[2] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Proc. 16th European Conference on Computer Vision (ECCV 2020)*, August 2020. 25 pages.

[3] Abril Corona-Figueroa, Jonathan Frawley, Sam Bond Taylor, Sarath Bethapudi, Hubert P. H. Shum, and Chris G. Willcocks. MedNeRF: Medical Neural Radiance Fields for Reconstructing 3D-aware CT-Projections from a Single X-ray. In *Proc. 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC 2022)*, pp. 3843–3848, July 2022.

[4] Marc Levoy. Efficient Ray Tracing of Volume Data. *ACM Trans. Graph.*, Vol. 9, No. 3, p. 245–261, July 1990.

[5] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV 2021)*, pp. 14335–14345, October 2021.

[6] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV 2021)*, pp. 5752–5761, October 2021.

[7] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking Neural Radiance Fields for Real-Time View Synthesis. In *Proc. IEEE/CVF International Conference on Computer Vision (ICCV 2021)*, pp. 5875–5884, October 2021.

[8] Wei Li, Klaus Mueller, and Arie Kaufman. Empty Space Skipping and Occlusion Clipping for Texture-based Volume Rendering. In *Proc. IEEE Visualization (VIS 2003)*, pp. 317–324, October 2003.

[9] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. arXiv, 2015. abs/2006.05525.

[10] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. arXiv, 2014. abs/1411.1784.

[11] The Linux Foundation. PyTorch. `https://pytorch.org`, accessed February 1, 2023.

[12] NVIDIA Corporation. CUDA C Programming Guide Version 12.0. `{http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf}`, December 2022. accessed February 1, 2023.