

# Bfloat16形式の活用及び時空間的最適化に基づく Ryzen AI 上での 3D Gaussian Splatting の高速化手法

奥野 聡文<sup>1,a)</sup> 高前田 伸也<sup>1,2,b)</sup>

**概要：**Ryzen AI Engine は、高性能かつ高エネルギー効率を実現する Neural Processing Unit (NPU) を統合した、AI ワークロード向けの低消費電力エッジデバイスである。このデバイスは高密度な一般の行列乗算に高い適正を示すが、幅広いワークロードへの適用はアーキテクチャの独自性により依然として限定的である。このようなワークロードの一つとして、3D Gaussian Splatting (3DGS) があげられる。特に、3DGS は最先端の新規視点合成手法ではあるものの計算負荷が高いため、リソース制約のあるエッジデバイスへの効率的な展開が求められている。

本論文では、数値的な量子化と時間・空間的最適化を組み合わせることで、Ryzen AI 上での 3DGS の効率的な実装を提案する。数値的最適化については、メモリ使用量の削減と演算効率の向上のために、NPU 上でネイティブにサポートされている bfloat16 を採用する。しかし、bfloat16 の単純な採用は現実的ではない。bfloat16 の限られた仮数部の精度は数値的不安定性を生じさせ、レンダリング品質を著しく低下させるという課題がある。この問題に対処するため、我々は 3 つの誤差低減戦略を提案する：(1) 共分散行列の選択的な再計算、(2) ソートに用いるインデックスの最適化、(3) 画面座標の高精度での計算である。それらに加え、時空間的な最適化として SIMD ベースのカーネル設計、カーネルからタイルへのマッピング、デバイス間のスケジューリングなど、NPU, GPU, CPU にわたる多岐な並列性を活用する最適化を導入する。

カーネルごとの評価では、本研究は既存の手法に対して最大で 5.47 倍のサイクル数の削減を達成した。またエンドツーエンドの評価では、3 つのベンチマークデータセットを用いた実験の結果、提案手法は、GPU のみを用いた実装と比較して 1.07 倍から 1.10 倍の高速化を達成し、かつレンダリング品質の低下も十分に抑えられることが示された。

## 1. 序論

GPU は、長年にわたり高性能計算の基盤アーキテクチャとして、大規模並列処理と高い演算スループットのもと、大規模言語モデル (LLM) や大規模数値計算といったワークロードを支えてきた。しかし、特にリソースが制約されたエッジ環境において、性能とエネルギー効率を両立したデバイスの需要が急速に高まっていることから、従来の GPU を代替する専用アクセラレータの開発が進んでいる。その一例である Ryzen AI Engine [1] は、CPU, GPU, およびニューラルプロセッシングユニット (NPU) を単一のシステム内に統合したプラットフォームである。この Ryzen AI の特徴としては、NPU が XDNA2 アーキテクチャ [2] と

呼ばれる、AI Engine (AIE) タイルと呼ばれる軽量なコアとメモリからなる演算タイルを 2 次元メッシュ状に配置した、空間的なアーキテクチャを採用している点である。このアーキテクチャは、処理の並列性とデータの局所性を活用することで、行列演算などの構造化された計算を効率的に計算できるよう設計されている。その結果、このアーキテクチャは行列計算やマルチヘッドアテンション [3] など主要な AI 関連処理において高い性能とエネルギー効率を実現しており、FPGA ベースのシステム [4] から消費者向けプロセッサ [1] に至るまで、幅広いプラットフォームでの導入が進んでいる。しかし、その適用範囲は依然として一部のワークロードにほぼ限定されており、このアーキテクチャをより複雑で不規則な演算を要するワークロードに活用することは、依然として未解決の課題である。そのようなワークロードの一例が、3D Gaussian Splatting (3DGS) である [5]。

3DGS は、最先端の新規視点合成手法である。近年の技術

<sup>1</sup> 東京大学  
University of Tokyo, Bunkyo, Tokyo 113-8656, Japan

<sup>2</sup> 理化学研究所  
RIKEN

a) okkundori@is.s.u-tokyo.ac.jp

b) shinya@is.s.u-tokyo.ac.jp

の進歩により、コンピューターグラフィックス(CG)技術はデジタルツイン、拡張現実(AR)／仮想現実(VR)、シミュレーションなどの幅広い分野に応用されている [6], [7], [8], [9]. そして CG 技術の適用範囲が架空のシーンから現実世界の再構築へと拡大するのに合わせて、よりリアルな視点合成技術の需要が高まっている。

これに応えるために、3DGS は新規視点合成手法の一つとして開発された [5]. 3DGS は、ガウシアンと呼ばれる概念を用いてシーンを表現する手法であり、このガウシアンの持つ高い表現力により、高解像度な画像においても、他の手法に比べて高速かつ高品質なレンダリングを行うことができる。しかし、3DGS は比較的高速かつ高性能なレンダリング手法ではあるものの、その計算コストは依然として大きく、計算性能が限られたエッジデバイスでの運用には課題がある [10]. したがって、これらのデバイス上で効率的に 3DGS を動かすことは重要な課題である。

先行研究はこの XDNA2 アーキテクチャの 3DGS への利用可能性を示したものの、実際の速度向上は未だ達成されていない [11]. そのため本研究では、XDNA2 アーキテクチャを活用することによって、3DGS 処理を高速化できることを実証する。

これを実現するために、我々はアーキテクチャとアルゴリズムの両方のレベルにおいて、いくつかの最適化手法を提案する。アーキテクチャレベルでは、NPU に対する空間的な最適化と、演算コア間の時間的な最適化を行う。具体的な最適化内容は以下の通りである：

- 並列性を最大化するため、計算カーネルとその NPU 内での配置を明示的に設計する。
- CPU と NPU を並列にスケジューリングするためダブルバッファリング戦略を採用し、さらにレイテンシ隠蔽のために GPU 上で行われる次のフレームのレンダリングと NPU で行われる前処理を並行して行う。

アルゴリズムレベルでは、前処理に bfloat16 形式を活用する。bfloat16 数値形式は、単精度浮動小数点の仮数部を量子化したものであり、もともとは Google Brain チームによって AI タスク向けに開発された形式である [12]. Ryzen AI Engine では bfloat16 演算がネイティブにサポートされており、bfloat16 を活用することで効率的に実行時間とメモリの転送量を削減できるという利点がある。しかし、我々は 3DGS への bfloat16 の単純な適用はレンダリング品質を著しく低下させることを示した。この劣化を低減するため、我々は効果的に画質を維持する 3 つの誤差低減戦略を提案する：

- 共分散行列における重大な数値誤差を検出し、それらを選択的に高精度で再計算する。
- ソートをし、ガウシアンの  $z$  座標で行うのではなく、代わりにヒューリスティックに計算されたインデックスを用いて行う。

- 画面座標の計算を bfloat16 精度で行うことを避け、代わりに通常の単精度浮動小数点で計算する。

また本研究では bfloat16 の使用に加え、単一命令複数データ (SIMD) 演算を活用してさらなる高速化を実現する。具体的には複数のガウシアンからのデータを単一のベクトルにパックすることで、並列に計算を行う。

最終的に、本研究の貢献は次のようにまとめられる：

- (1) スケジューリングおよびタイル配置戦略を通じて Ryzen AI Engine の並列性を活用し、高いリソース利用率を実現する。
- (2) 前処理に bfloat16 形式を採用することで高速化を実現しつつ、3 つの緩和策を用いてレンダリング画像の品質を維持する。
- (3) SIMD 演算を用いて、複数ガウシアンの計算を並列に実行する。

我々の実験では、既存手法に比べて提案手法がガウシアン当たりに必要なカーネルのレイテンシを 1.33 から 5.47 倍減少出来ることを確認した。また、3 つのベンチマークで同一デバイス上で単純な GPU 実装と比較して、提案手法が最大 1.10 倍のエンドツーエンドの速度向上を達成することを示した。さらに、SSIM および PSNR の低下幅はそれぞれ 0.014~0.037dB と 0.605~1.318dB に抑えられ、提案手法が画質を維持できることを実証した。

## 2. 背景

### 2.1 3D Gaussian Splatting (3DGS)

新規視点合成とは、事前に撮影された限られた数の画像を元にシーンを構築し、該当シーンの任意の視点からの画像を生成するというタスクである。これに対する代表的な手法である Neural Radiance Field (NeRF) は、深層ニューラルネットワークを用いてシーンをパラメータとして符号化する手法である [13]. しかし、NeRF はニューラルネットワークの計算コストが高く、リアルタイムレンダリングが困難であるという課題がある。

そのため、NeRF に変わる最先端の手法として 3DGS が提案された [5]. 3DGS は、シーンをニューラルネットワークでエンコードする代わりに、ガウシアンと呼ばれる概念を用いて明示的にシーンを表現し、高解像度でのリアルタイムレンダリングを可能にする。各ガウシアンは、位置  $\mu$ 、大きさ  $s$ 、クォータニオン  $q$  で表される回転、不透明度  $\alpha$ 、および色を表現するために用いられる球面調和関数の係数  $sh$  といった属性を持つ。これらの属性はそのままではレンダリングに適さないため、まず前処理を通じてレンダリングしやすい形式に変換され、その後レンダリングフェーズに渡される。前処理の過程では、以下の計算が行われる。ガウシアン分布の位置情報  $\mu$  はワールド座標系にあるため、アフィン変換で 3 次元のカメラ座標系および 2 次元のスクリーン座標系に変換される。スクリーン座標系にお

るガウシアン分布を表す 2 次元共分散行列  $\Sigma'$  は, 3 次元共分散行列  $\Sigma$ , 変換行列  $R_{cam}$ , およびアフィン変換のヤコビ行列  $J$  から導出される. なお,  $R$  および  $S$  は, 共分散を定義する回転行列およびスケール行列であり, それぞれ属性  $q$  および  $s$  から計算される.

$$\Sigma' = JR_{cam}\Sigma R_{cam}^T J^T \quad (1)$$

$$\Sigma = RSS^T R^T \quad (2)$$

また, 視線方向からのガウシアンの色  $c$  は, 球面調和関数の係数を評価することで計算される.

前処理後, すべてのガウシアンは深度順に手前から奥へとソートされ,  $\alpha$  ブレンディングを用いてレンダリングされる. 画面の手前から  $i$  番目のガウシアンの属性を添え字  $i$  で表し, ターゲットのピクセルから  $i$  番目のガウシアンの中心までの咆哮ベクトルを  $X_i$  と表すと, ピクセル  $C$  の最終的な色は, 次のように計算される.

$$C = \sum_{i=1}^N \alpha'_i c_i T_i \quad (3)$$

$$T_i = \prod_{j=1}^{i-1} (1 - \alpha'_j) \quad (4)$$

$$\alpha'_i = o_i \exp\left(-\frac{1}{2} X_i^T \Sigma_i'^{-1} X_i\right) \quad (5)$$

## 2.2 Bfloat16

Brain Floating Point (bfloat16) は, 機械学習タスクのために開発された 16 ビットの浮動小数点形式である [14]. 16 ビットの浮動小数点として標準的な半精度浮動小数点形式 (fp16) は, 符号部 1 ビット, 指数部 5 ビット, 仮数部 10 ビットで構成されているのに対し, この bfloat16 は符号部 1 ビット, 指数部 8 ビット, 仮数部 7 ビットで構成される点の特徴である. そのため bfloat16 は, 仮数部を減らして数値精度を犠牲にする代わりに, fp16 よりも広い数値幅を表現することができる. これは機械学習タスクにおいて, 小さな数値精度よりもより広い値の範囲を表現する能力の方が有益であるという観察に基づいている. Bfloat16 は Google Brain チームによって Tensor Processing Units (TPU) 向けに導入され, タスク効率の向上に大きく寄与した [12].

## 2.3 XDNA2 アーキテクチャ

XDNA2 は, AMD が計算負荷の高いワークロード, 特に機械学習ワークロードの高速化を目的として開発したアーキテクチャである [2]. Ryzen AI NPU は, このアーキテクチャを採用した代表的なプラットフォームの一つである. このアーキテクチャは, AI Engine (AIE) タイルと呼ばれる複数の小型の処理ユニットを 2 次元メッシュ状に

配置するという構造を取っている.

XDNA2 は, 演算タイル, メモリタイル, インターフェースタイルの 3 種類のタイルで構成されている.

演算タイルはこのアーキテクチャの主要な処理ユニットである. このタイルの中には SIMD 演算をサポートする処理コアと, I/O オーバーヘッド低減のための小さなローカルキャッシュが備えられている.

メモリタイルは, メインのキャッシュとして働くユニットである. このタイルは計算をする代わりに比較的大きな, L2 キャッシュとして機能するメモリを提供する.

インターフェースタイルは, 外部ホストメモリとの間の I/O を担当するユニットである. このタイルには主に, 外部ホストメモリ間とのデータ転送を管理するためのコントローラが搭載されており, 外部との I/O に特化したユニットとなっている.

これらの 3 種のタイルを組み合わせることで, XDNA2 は高い演算スループットとスケーラビリティを実現している.

## 3. 3D Gaussian Splatting での Bfloat16 活用のためのアルゴリズム

### 3.1 Bfloat16 による量子化

前述の通り, bfloat16 は IEEE の単精度浮動小数点数形式 (fp32) の仮数部を切り落とし, 精度を減らした浮動小数点形式である. これは 16 ビット形式であるため, fp32 に比べて計算レイテンシやデータ移動のオーバーヘッドが小さく, ML ワークロードで高いパフォーマンスを実現する [12]. 本研究では ML ワークロードにおける bfloat16 の有効性に着想を得て, 我々は 3DGS の前処理において量子化として, ハードウェアレベルでサポートのもと bfloat16 を採用した. なお, 本研究のレンダリングの前処理には bfloat16 が使用されるものの, そのレンダリングで用いるデータの学習ステップには一切の変更を加えていない. 本論文で使用されるガウシアンのパラメータは, すべて fp32 精度でのオリジナルの 3DGS 手法を用いて学習されている.

しかし, 3DGS に bfloat16 を適用することには課題がある. これは, bfloat16 の精度の限界や数値誤差 (桁落ちなど) の蓄積が, レンダリング品質を著しく低下させる場合があることに起因する. この問題に対処するため, 我々は 3 つの緩和策を提案する.

### 3.2 共分散行列の選択的な再計算

2 次元共分散行列は数値誤差の影響を特に受けやすく, レンダリング品質に重大な影響を及ぼす. これは, 2 次元共分散行列がレンダリングにおいてどのように使用されるかに起因する. 2 次元共分散行列は画面上に投影されたガウシアンのガウス分布を表すため, 画面上の特定の点におけるガウス分布の寄与を計算する時, この行列の逆行列と

方向ベクトルによる演算を行うことになる。ここで2次元共分散行列  $\Sigma'$  は対称行列であるため、逆行列は次のように表すことができる。

$$\Sigma' = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \quad (6)$$

$$\Sigma'^{-1} = \frac{1}{ac - b^2} \Sigma' \quad (7)$$

この計算をするには  $\Sigma'$  の行列式、すなわち  $ac - b^2$  を計算する必要がある。しかし、 $ac$  と  $b^2$  の2つの値が十分近い場合、それまでの計算ステップで蓄積された  $a, b, c$  の誤差や減算時の桁落ち誤差によって行列式に大きな差が生じることがある。その結果、行列の逆行列が真の値と大きく異なるものになる場合がある。

そのため、我々はこれらの致命的な数値誤差を効率的に検出して、後に通常の fp32 精度での共分散行列の再計算を CPU 上で行うことで、誤差による影響を低減する方法を提案する。誤差評価のため、我々はガウシアンを安全なガウシアン、不安定なガウシアン、および誤計算されたガウシアン の3つのグループに分類する。

誤計算されたガウシアン とは、行列式  $ac - b^2$  が負となるガウシアンのことである。2次元共分散行列は2次元ガウス分布の分散共分散行列であるため、その行列式は必ず0または正の値になる。そのため、行列式が負と計算された場合、これは計算エラーであると判断でき、誤計算として分類される。

不安定なガウシアンとは、行列式が0に近いガウシアンの事である。行列式が十分小さい場合、bfloat16 精度では計算結果での桁落ちの影響が大きくなる。そのため行列式の値が0に近い場合その数値を信頼せず、fp32 精度で再計算された値を用いるべきである。より正確には以下の条件が成立する場合に、そのガウシアンを不安定と呼ぶ。

$$\frac{ac - b^2}{ac} < \tau \quad (8)$$

ここで、 $\tau$  は小さな閾値である（本研究では、これを 0.01 に設定する）。

安全なガウシアン とは、それ以外の、行列式が正でかつ小さくないガウシアンのことである。このグループのガウシアンにも依然数値誤差が含まれている可能性があるものの、これらの誤差は値そのものに比べて無視できる程度であるため、再計算を行う必要はない。

この分類の後、不安定または誤計算とマークされたすべてのガウシアンは、再計算の候補として扱われる。

我々の観察から、不透明度の低いガウシアンは画像に及ぼす影響が限定的であり、たとえ顕著な数値誤差を示した場合でも、再計算の必要性が低いと分かった。したがって、

本手法では不透明度の高いガウシアンのみを再計算し、計算コストと数値的安定性のバランスを図っている。

また上述した行列式による分類に加えて、本研究ではサイズが大きなガウシアンを支配的なガウシアンとしてマークする。これらのガウシアンはシーンの大きな部分を占めており、仮に行列式による分類で安全とされていたとしても、そのまま描画に用いると品質に影響が出る。そのためすべての支配的なガウシアンは、その不透明度にかかわらず、必ず FP32 精度で再計算される。

### 3.3 最適化されたソート用インデックス

オリジナルの 3DGS では、 $\alpha$  ブレンディングのために、レンダリング前にすべてのガウシアンが深度（すなわち、カメラ座標系における  $z$  座標）順にソートされる。しかし bfloat16 精度の場合、同じ方法でソートを行うとうまく行かない場合がある。これは互いに近い2つの値が同じ bfloat16 表現に量子化されて、順序に曖昧が生じることがあるためである。

もしガウシアンが正しくソートされていないと、累積透過率に誤りが生じ、レンダリング画像の品質低下が発生する。そしてこの影響は、不透明度が高い（特に1に近い）ガウシアンにおいて顕著になる。例えば、ガウシアン  $G_1$  が高不透明度のガウシアン  $G_2$  の手前にあるにもかかわらず、bfloat16 精度では逆順にソートされてしまうと、ガウシアン  $G_1$  の寄与が大幅に減衰してしまう。

この影響を軽減するため、我々は二つの要素を用いてソートを行う手法を採用した。具体的には、二つのガウシアンはまずその  $z$  座標で比較され、もしそれが同値の場合は不透明度で比較される。この戦略は、たとえガウシアンが誤った順序でソートされていたとしても、不透明度の低いガウシアンが前面にある場合、最終的な画像への影響が限定的であるという洞察に基づいている。

実際にはこれは1回の整数比較でこの二段階の比較を行うために、ハイブリッドなインデックスとして実装される。このインデックスは符号なし32ビット整数として表され、上位16ビットはガウシアン関数のカメラ空間  $z$  座標の bfloat16 のビット表現、下位16ビットは不透明度の bfloat16 のビット表現が割り当てられる。図1はこの構成を示す。ここで、2つの正の正規化浮動小数点値（例：bfloat16 や fp32）は、符号なし整数として比較した場合もその数値的な順序が保持されることに留意されたい。この特性に基づき、最上位16ビットを  $z$  座標（最初のインデックス）として、最下位16ビットを不透明度（タイブレーカー）として利用することで、一度の整数比較により二つの指標を用いたソートを実現することが出来る。なお、不透明度と  $z$  座標は常に正の値となることが保証される。これは、 $z$  座標が負のガウシアンはレンダリングに寄与しないため、前の段階で除去されるためである。

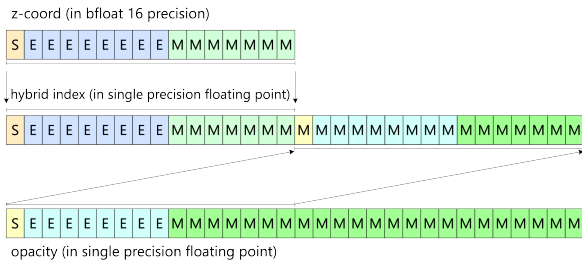


図 1 ハイブリッドソートインデックスの概念図

### 3.4 高精度での画面座標の計算

前述したように、前処理の大部分は十分な配慮のもと bfloat16 精度で実行可能である。しかし、ガウシアン画面座標の計算は例外である。これらの座標は通常数百から数千のオーダーであり、そのうえ小数点以下の細かい精度が求められる。しかし bfloat16 では仮数部の精度が限られているため、小数点以下が切り捨てられた大雑把な値に丸められてしまう。したがって、画面座標の計算に bfloat16 を採用すると、ガウシアン位置が粗い離散的なグリッドに量子化され、縞状のひずみが画像に現れてしまう。これを防ぐため、我々は画面座標の計算に fp32 精度を使用する。ただし、二種類以上の数値精度を NPU に転送することには制限があるため、実装上この計算は NPU ではなくホスト CPU 上で実行する。

## 4. アーキテクチャ親和的な最適化

### 4.1 カーネルの設計とマッピング

Ryzen AI NPU では複数のタイルを 2 次元メッシュ状に配置して、並列化・パイプライン化して動作させることを通じて、複雑なタスクを分割処理する最適化を行うことができる。そのため本研究では、先行研究 [11] のアイデアをベースに、前処理を小さな計算タスクに分割し処理の複雑さを低減しつつパイプライン実行を可能にすることで、レイテンシを向上させる戦略を取ることにした。これによりプロセス全体でレイテンシの低減が実現できるが、依然として一部の計算カーネルは負荷が他のカーネルよりも著しく高くなり、パイプライン全体のボトルネックになってしまうという課題が存在する。この問題に対処するため、我々は特定のカーネルのみを複数のタイルに分散させる、負荷注目型のタイルマッピング戦略を提案する。この戦略では、負荷が重い一部のカーネルを複数の AIE タイルに分割して並列に実行することで、1 タイルあたりの計算負荷を大幅に軽減し、スループットの向上によるカーネル遅延の低減を行う。例えば、軽量のカーネルが単一の AIE タイルで 128 個のガウシアンを処理している間に、4 つのタイルに分散された計算負荷の高いカーネルは、1 タイルあたり 32 個のガウシアンを計算すれば良いようにすることで、総計算時間を均すことができる。図 2 は、具体的な計算の

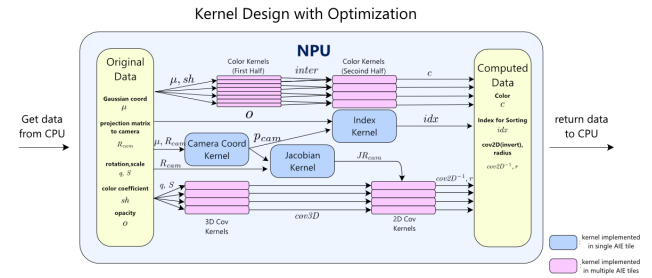


図 2 Ryzen AI Engine における計算フローとカーネル設計の概要。

フローとカーネル設計の概要を示している。ピンクの長方形は並列性を利用するために複数の AIE タイルに複製された高負荷なカーネルを表し、青色の長方形は単一の AIE タイルに実装された軽量のカーネルを示している。本研究では測定された各カーネルの実行サイクル数に基づいて、3 次元共分散カーネル、2 次元共分散カーネル、およびカラーカーネルの後半部分を 4 つのタイルに複製し、カラーカーネルの前半部分を 8 つのタイルに複製している。

### 4.2 ユニット間のスケジューリング

Ryzen AI には、CPU、GPU、NPU という 3 種類の計算ユニットが搭載されている。そしてこれらの性能を最大限に引き出すためには、適切にタスクを分解してスケジューリングを行い、並列に活用することが重要である。また、これらのユニットはそれぞれ異なる種類の性質を持った演算に特化しているため、効率良い利用のためには特性を活用した分割を考える必要がある。例えば、GPU は非常に高い並列性を有するのが特徴であるのに対し、NPU は並列性が中程度であり、代わりにタスクのパイプライン処理に適しているという差がある。また、CPU は高い並列性を欠くものの、それよりはるかに複雑なタスクを柔軟に実行することができるという強みがある。

これらの特性を活用するため、本研究ではパイプライン実行が有効的な 3DGS の前処理を NPU に、画面座標の計算および高精度での 2D 共分散行列の再計算を CPU に、そして高い並列性が求められる 3DGS のレンダリングを GPU に割り当てる。

また、これらのユニットは独立して動作させることができるため、レイテンシ隠蔽のため並列なスケジューリングを行う。図 3 は、このスケジューリング戦略を視覚化したものである。図の上半分は、前処理中の CPU-NPU 間のスケジューリングを示している。本研究においては、3DGS の前処理ではオーバーヘッド低減のためガウシアンを一定個数（本研究では 32768 個）まとめてバッチ化して NPU 上で処理する。そして NPU がこのバッチを処理している間、CPU は一つ前のバッチに含まれるガウシアンに対する再計算と画面座標の計算を実行し、そのレイテンシを NPU の実行と重ねて隠蔽する。なお NPU はデータ転送に常に

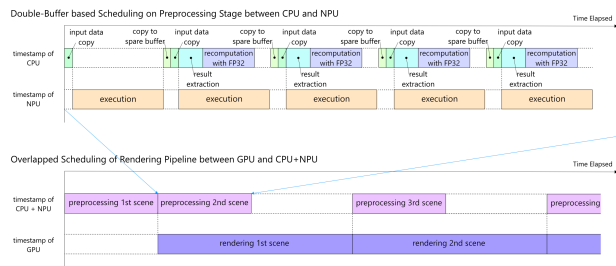


図 3 CPU, GPU, および NPU 間のスケジューリングの可視化.

同じメモリ領域を使用するという制約があるため、ダブルバッファリングを行うため、NPU によって計算されたデータを CPU 計算用の補助バッファにコピーしている。また図の下半分は、各描画シーン間でのレンダリングと前処理の、GPU と NPU の間のスケジューリングを示している。シーン同士は独立しているため、あるシーンの前処理計算を行うためにその前のレンダリングが完了している必要はない。そのためシーン  $i+1$  の前処理を NPU と CPU 上で実行しつつ、シーン  $i$  のレンダリングを GPU 上で実行することができる。これにより前処理とレンダリングをパイプライン化し、それらの実行をオーバーラップさせることで、高いリソース利用率を実現している。

この戦略を採用する際、考慮すべき点の一つがガウシアンの数である。NPU の計算時間はガウシアン関数の数に対して急な線形に比例するが、GPU はそれよりもゆるやかなスケーリングを示す。そのためガウシアンの数が大きい場合（通常 100 万～150 万以上）、前処理にレンダリングよりも時間がかかり、この戦略の全体的なメリットが低下してしまう。この場合、我々は元々の 3DGS と同じように前処理を GPU にフォールバックすることで、高負荷なシーンでも NPU を利用したことによる性能劣化を考えずに実行できることを保証している。

また、先行研究によると、3DGS では学習済みのガウシアンをレンダリング品質の低下を最小限に抑えつつ 80～90 %削減できることが示されている [15], [16]。そのためこのような剪定を事前に適用することで、数百万個のガウシアンを含むシーンであっても、基本的には NPU ベースの前処理が依然として有益な規模にまで削減できる。ただ、GPU への処理のフォールバックを行うことで、この剪定が適用されていない場合や、品質上の制約により望ましくない場合でも、処理時間が必要以上に増加することを防ぐ保証を行うことが出来る。

#### 4.3 SIMD 最適化

Ryzen AI Engine は SIMD ベクトル演算をサポートしたコアを搭載しており、スループットを最大化するには、これらの演算を活用することが不可欠である。現在の先行研究では、XDNA2 上での 3DGS への SIMD 演算の活用は、

3 次元共分散行列の計算という特定の計算に限定されている [11]。そこで我々は SIMD 演算を最大限に活用するため、3DGS 前処理における個々のガウシアン計算が独立であることに着目する。この独立性から、16 個のガウシアンの情報を単一のベクトルにパックし、それらのベクトルに対して SIMD 演算を用いることで並列に前処理を行うことが出来る。その後、計算された値はベクトルから抽出されて利用される。しかし、この並列化は常に出来るとは限らないという問題がある。実際にはレジスタ負荷などの制約により、計算負荷の高い一部のカーネルは効率的にベクトル化できないのである。したがって、本研究ではアーキテクチャとコンパイラの制約が満たされる限り SIMD 演算を採用するが、それらが満たされない場合は残りの計算は通常の単一データ処理で行うこととする。

## 5. 実験

### 5.1 実験設定

提案手法の評価は、サイクルレベルとエンドツーエンドの 2 つの観点から行う。Ryzen AI Engine に関するすべての実験は、IRON フレームワーク [17] および OpenCL を用いて実装され、AMD Ryzen™ AI 9 HX 370 チップ上で行われた。

#### 5.1.1 サイクルレベル評価の実験設定

本実験では、各計算カーネルにおけるガウシアン 1 つあたりのサイクルレイテンシを評価する。具体的にはガウシアンの属性をランダムに生成し、IRON フレームワークが提供するトレースシステムを用いて計算に掛かったサイクル数を実機計測する。安定性のため、20 回カーネルを反復実行してサイクル数をサンプリングして平均化した後、それを 1 反復あたりのガウシアン数で割ることで、ガウシアン 1 つあたりのサイクルレイテンシを算出する。なお、反復ごとのガウシアン数は、そのカーネルに割り当てられた AIE タイルの数によって異なる（詳細はセクション 4.1 を参照）：詳細には 1 タイルの場合は 128 ガウシアン、4 タイルの場合は 32 ガウシアン、8 タイルの場合は 16 ガウシアンである。本実験のベースラインとしては、Shimamura ら (2025) [11] で報告されている、Window 法によるサイクル数を採用した。なお、彼らはシミュレーションを用いてサイクル数を評価しているのに対し、我々は実デバイス上のトレースシステムを使用しているという差があることに留意されたい。両研究は NPU が搭載されているデバイス自体は異なるものの、同じ XDNA2 アーキテクチャの NPU を使用している。

#### 5.1.2 エンドツーエンド評価の実験設定

エンドツーエンド評価のために、Mip-Nerf 360 データセット [18] の全シーン、synthetic Blender データセット [13] の全シーン、および Tanks&Temples データセット [19] からの 3 つのシーンを採用した。選択したデータセット

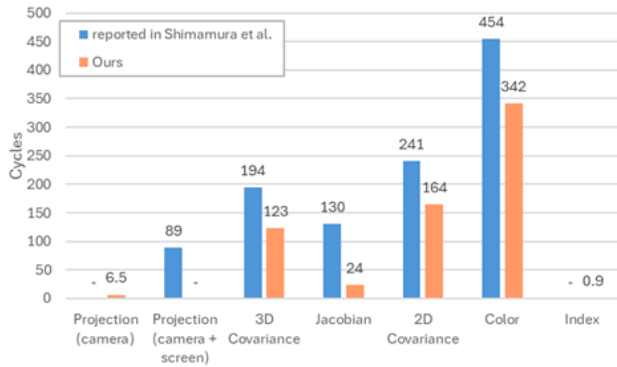


図 4 ガウシアンごとのサイクルレイテンシの比較

には、自然環境から合成されたシーンに至るまで、屋内および屋外の環境を含む様々なシーンが含まれている。これらすべてのデータセットは A100 GPU 上でオリジナルの 3DGS 実装を用いて学習させた後、PUP 3D-GS [16] を用いて剪定を行うことでレンダリング用のデータを生成した。

レンダリングについては、オリジナルの 3DGS は CUDA で実装されており Ryzen の GPU とは互換性がないため、本実験では OpenCL を用いて Ryzen の GPU 上で 3DGS のレンダリングパイプラインを再実装しベースラインとした。性能指標として、データセットごとの計算時間を比較する。また、提案手法およびオリジナルの 3DGS で生成された画像について、Peak Signal-to-Noise Ratio (PSNR) および Structural Similarity (SSIM) の二種類の指標を用いて、正解画像との誤差も比較する。

## 5.2 サイクルレベル評価の結果

図 4 は、各カーネルにおけるガウシアンごとのサイクルレイテンシを示している。なお、それぞれの研究に存在しないカーネル (Shimamura ら [11] の研究におけるインデックスカーネルやカメラ座標カーネル、および提案手法におけるカメラ座標カーネルと画面座標カーネルを組み合わせた投影カーネル) についてはサイクル数がハイフンで表示され、比較対象から除外されている。さらに両研究ともカラーカーネルを 2 つの部分に分解しているものの、その分割方法は異なるため、比較のためにそれらのサイクルレイテンシを合計した。

実験結果から、両研究で実装された比較可能なすべてのカーネルにおいて、提案手法はカーネルレイテンシを減少させることが示された。特に最も改善されたカーネルであるヤコビアンカーネルではレイテンシが 5.47 倍低減し、最も改善が少なかったカラーカーネルでも 1.33 倍の低減が見られた。すべてのカーネルは bfloat16 精度で計算されているため、このカーネルごとの削減率の差は主に SIMD の活用度合いに起因する。特にヤコビアンカーネルのようなレジスタ負荷の低いカーネルは SIMD 演算を用いて積極的に最適化できるが、カラーカーネルのような計算負荷の高

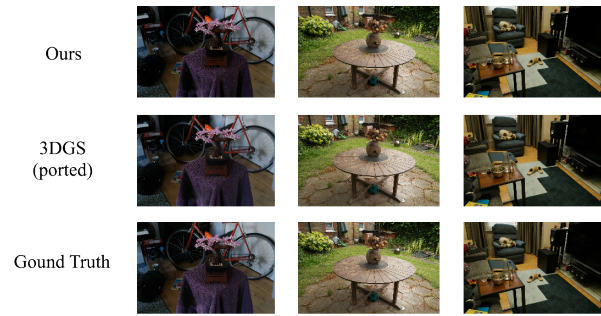


図 5 各手法によるレンダリング画像の比較

いカーネルはそのような最適化を行うと計算結果が誤ったものになってしまうため、SIMD 最適化を行うことが難しい。この誤計算はレジスタの溢れに起因する可能性が高いが、正確な原因についてはさらなる調査が必要である。

表 1 は、ガウシアン 1 つあたりおよび 1 反復あたりのサイクルレイテンシを示している。一部のカーネルは並列処理のために複数の AIE タイルにマッピングされるため、1 回の反復処理中に各 AIE タイルが処理するガウシアン数は  $128/N$  と計算できる。ここで、 $N$  は当該カーネルに割り当てられた AIE タイルの数を表す。総レイテンシは、ガウシアン 1 つあたりのサイクルレイテンシと、タイルあたりに処理されるガウシアン数の積によって与えられる。

表 1 に示された数値から、計算負荷の高いカーネルのほとんどは 1 回の反復につき約 3000~5000 サイクルを必要としており、大きな差が無いことが見受けられる。これは、傾斜のあるカーネルのタイル割り当てがレイテンシを効果的に均等化し、イテレーションごとのレイテンシ差を小さくさせ、効率的なパイプライン化を可能にしていることを示している。

## 5.3 エンドツーエンド評価の結果

表 2 は、データセット全体の総レンダリング時間、および平均 PSNR と SSIM の比較である。これによるとすべてのデータセットにおいて、提案手法はレンダリング時間を 1.07~1.10 倍短縮していることが分かる。しかし、SSIM が 0.014~0.037、PSNR が 0.605~1.318 dB 低下しているように、レンダリング品質の指標上の低下も見られる。

ただし、移植された 3DGS 実装と提案手法によるレンダリング画像の比較図 5 を見ると、指標である SSIM と PSNR の低下に対し、レンダリングにおける知覚的な劣化はほとんど見られない。これは、指標の低下が主にその低下分の大部分は色の小さな違いや画像のちょっとした座標のずれ等によるものであるからであると考えられる。

また追加で前処理時間についても測定すると、前処理時間とレンダリング時間の比率は、Blender データセットで 23.38 %、Mip-Nerf データセットで 67.56 %、Tanks&Temples データセットで 38.18 %であった。これ

表 1 提案手法におけるガウシアンごとのサイクルカウント数および反復回数.

Kernels	cycles per Gaussian Gaussian	# of allocated AIE tiles	# of Gaussians computed in one tile	total latency per iteration
Camera Coord	6.5	1	128	827.0
Jacobian	24	1	128	3043.5
Index	0.9	1	128	121
3D Covariance	123	4	32	3939.6
2D Covariance	164	4	32	5251.8
Color (second half)	95	4	32	3050.0
Color (first half)	246	8	16	3940.4

表 2 総レンダリング時間と画質の比較. レンダリング時間とは、各データセットのシーンをレンダリングするのに必要な合計時間を指す.

Method	Blender			Mip-Nerf			Tanks&Temples		
	time (s)	PSNR	SSIM	time (s)	PSNR	SSIM	time (s)	PSNR	SSIM
GPU only (baseline)	39.79	26.132	0.921	102.66	28.578	0.853	55.71	23.343	0.757
Ours	36.48	25.129	0.907	96.20	27.260	0.816	50.45	22.738	0.728

表 3 平均ガウシアン数と、前処理時間とレンダリング時間の比率

Dataset	# of Gaussian	ratio (%)
Blender	18649	23.38
Mip-Nerf	282435	67.56
Tanks&Temples	106889	38.13

らの結果は表 3 にまとめられている. すべてのデータセットにおけるガウシアンの平均数はシーンの複雑さに応じて約 10,000 から 300,000 の範囲にあり, このガウシアン数の範囲では前処理時間はレンダリング時間よりも短く, 並列実行が効果的であることが分かる. しかし, ガウシアンの数を増やすと前処理時間の割合が増加するため, ガウシアンの数が多くなるにつれて本手法のスケーラビリティに制限が生じることも示唆されている.

## 5.4 アブレーション実験

### 5.4.1 アブレーション実験の実験設定

本節では, 誤差の緩和戦略の効果をそれぞれ分離して評価する. アブレーションとして, 以下の 3 つの状況下で, 正しい画像と比較した PSNR および SSIM を算出して比較することで, 各緩和策の効果を測定する. なお, ここで使用するデータセットやデバイスなどの実験設定はエンドツーエンド実験と同等である.

- (1) 誤計算されたガウシアン, 不安定なガウシアン, および支配的なガウシアンに対する再計算を無効化する (詳細はセクション 3.2 を参照).
- (2) 画面座標の計算を bfloat16 精度で行う (NPU 上での実装がないため, CPU 上で実行する).
- (3) ソートインデックスの最適化を無効化し, 代わりにもとの実装と同じくガウシアンの深度をインデックスとして使用する.

各アブレーション実験では, 提案手法の単一の構成要素のみを変更し, 他の要素は同一に保つことで, 各構成要素の



図 6 提案手法と微分再計算を行わない場合の比較. 再計算を行わないバージョンでは, 誤計算された, 不安定な, および支配的なガウシアンの再計算が省略される.

効果を個別に評価できるようにしている. この結果は表 4 にまとめられている.

### 5.4.2 選択的再計算による影響

図 6 は, 提案手法とその再計算をスキップした版から出力された画像を示している. この図を見ると, 再計算を行っていないほうの画像には不自然な細長い楕円形の物体が映り込んでいることが分かる. これらは, 数値誤差の著しいガウシアンが画像として現れたものである.

また, Mip-Nerf および Tanks&Temples データセットの PSNR の低下は, Blender データセットでの低下 (0.60 dB) よりも大きい値を示している (最大で 1.69 dB 低下) ことが確認された. これは主にこれら現実のシーンに基づくデータセットには背景領域が存在することに起因する. 背景構築に使用されるガウシアンはサイズが大きい傾向があり, その誤差は画像の広い領域に伝播しやすい. そのためこれらの誤差は, たとえその誤差が微小でも, シーンを著しく汚染する可能性が高いと考えられる.

また追実験として, 再計算されたガウシアンの割合と, 前処理ステップ全体に対する再計算時間の割合についても測定した. その結果は表 5 に示す通りである. すべての

表 4 各種アブレーションにおける平均 PSNR および SSIM. 評価にあたっては、すべてのレンダリング画像を正解画像と比較している.

Method	Blender		Mip-Nerf		Tanks&Temples	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Proposed Method (Original)	25.22	0.907	27.26	0.816	22.74	0.728
Without Recomputation	24.63	0.897	25.57	0.771	21.63	0.681
Screen Coordinates (bfloat16)	20.45	0.809	18.99	0.419	18.54	0.545
No Index Optimization	25.19	0.907	26.97	0.812	22.71	0.727

表 5 ガウシアン再計算率と、前処理全体にかかる時間に対する再計算時間の比率. ガウシアン再計算率とは、全体のガウシアン数に対する再計算されたガウシアンの割合を指す. 前処理は NPU 上で実行される一方、これらの再計算は CPU 上で実行されるため、この比率が高ければ両者を効果的に並行して実行することができる.

Datasets	gaussian ratio (%)	time ratio (%)
Blender	3.33	2.87
Mip-Nerf	5.76	7.31
Tanks&Temples	8.28	8.94



図 7 提案手法と、画面座標を bfloat16 精度で計算するアブレーションの比較.

データセットにおいて、再計算されたガウシアンの割合とその計算時間は約 3~9% である. したがってこの再計算戦略は、レンダリング品質と計算時間の程よいトレードオフを実現できる.

#### 5.4.3 画面座標の計算による影響

図 7 は、提案手法と、画面座標を bfloat16 精度で計算した実験との比較を示している. 定量的指標の低下が示すように、このアブレーションで生成された画像には不自然な格子模様がみられる. この模様は bfloat16 の表現可能な値が離散的であるために生じる (詳細はセクション 3.4 を参照).

#### 5.4.4 インデックス最適化による効果

図 8 は、提案手法とソートインデックスの最適化を行わない手法で生成された画像の比較である. これを見ると生成された画像間に視覚的に大きな違いは見られないものの、定量的な指標では、同じか若干の低下がみられる. ただ、このインデックスの計算は極めて軽量である (詳細は 5.2 節を参照) ことを考慮すると、この最適化は計算コストとレンダリング品質の間に良好なトレードオフをもたら



図 8 提案手法とインデックス最適化を行わないバージョンの比較.

すと言える.

以上のすべてのアブレーション実験から、選択的再計算の無効化と、画面座標の bfloat16 精度での計算時がレンダリング品質の低下が顕著であり、これらが特に品質の維持において重要であることを示している.

## 6. 考察と結論

本論文では、Ryzen AI Engine における 3DGS 向けのアーキテクチャ親和的な高速化手法を提案した. 具体的には、bfloat16 の活用およびユニット内およびユニット間の並列化戦略のもと、ベースライン実装と比較してエンドツーエンドのレンダリング時間で 1.07~1.10 倍の高速化を達成しつつ、画質の低下を SSIM では 0.014~0.037, PSNR では 0.605~1.318 dB に抑えた. 提案手法の高速化には、NPU 上での並列実行による前処理時間そのものの短縮と、GPU 上でのレンダリングで前処理のレイテンシを隠蔽することが大きく寄与している. 本研究ではオリジナルの 3DGS を採用したが、提案手法は同様の計算特性を持つ他の 3DGS のバリエーションにも適用可能である.

また、本研究は 3DGS における bfloat16 の利用が及ぼす影響についても分析した. 3DGS の前処理のうち色味の計算などの一部の計算は bfloat16 を使用して容易に実装できるのに対し、2 次元共分散行列の計算などの演算は bfloat16 の仮数部の制約されたビット数の影響を受けやすい. そのため本手法は分散共分散行列の特性に着目し、選択的な再計算によって、計算負荷とレンダリング品質との間で良好なトレードオフを実現している.

bfloat16 は、Ryzen AI Engine に限らず、最新の GPU や NPU など bfloat16 をネイティブにサポートするアーキ

テクチャにおいて、効率的な数値形式として普及しつつある。したがって、本研究が示した 3DGS の bfloat16 活用は、Ryzen AI Engine 上だけでなく、他のデバイスへの展開においても十分活用可能である。そのため本研究がもたらす知見は、今後の 3DGS 上で bfloat16 を利用する研究においても誤差の低減に役立つであろう。

結論としては、我々は Ryzen AI Engine 上で、アーキテクチャを意識した最適化と bfloat16 による量子化を組み合わせ、品質の低下を最小限に抑えつつレンダリングを高速化する効率的な 3DGS 実装を提示した。

**謝辞** 本研究の一部は JST CREST JPMJCR21D2 の支援による。

## 参考文献

- [1] AMD: AMD Ryzen™ AI - Windows PCs with AI Built In (2026). <https://www.amd.com/en/products/processors/consumer/ryzen-ai.html> [14th, January, 2026].
- [2] Rico, A., Pareek, S., Cabezas, J., Clarke, D., Ozgul, B., Barat, F., Fu, Y., Munz, S., Stuart, D., Schlangen, P., Duarte, P., Date, S., Paul, I., Weng, J., Santan, S., Kathail, V., Sirasao, A. and Noguera, J.: AMD XDNA NPU in Ryzen AI Processors, *IEEE Micro*, Vol. 44, No. 6, pp. 73–82 (online), DOI: 10.1109/MM.2024.3423692 (2024).
- [3] Wang, E., Bayliss, S., Bisca, A., Blair, Z., Chowdhary, S., Denolf, K., Fifield, J., Freiburger, B., Hunhoff, E., James-Roxby, P., Lo, J., Melber, J., Neuendorffer, S., Richter, E., Rosti, A., Setoain, J., Singh, G., Taka, E., Vasireddy, P., Yu, Z., Zhang, N. and Zhuang, J.: From Loop Nests to Silicon: Mapping AI Workloads onto AMD NPUs with MLIR-AIR, *ACM Trans. Reconfigurable Technol. Syst.*, (online), DOI: 10.1145/3785670 (2026). Just Accepted.
- [4] AMD: AMD Versal™ AI Core Series VCK190 Evaluation Kit, <https://www.amd.com/en/products/adaptive-socs-and-fpgas/evaluation-boards/vck190.html> [17th, January, 2026] (2026).
- [5] Kerbl, B., Kopanas, G., Leimkuehler, T. and Dretakis, G.: 3D Gaussian Splatting for Real-Time Radiance Field Rendering, *ACM Transactions on Graphics*, Vol. 42, No. 4 (online), available from (<https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/>) (2023).
- [6] Tewari, A., Fried, O., Thies, J., Sitzmann, V., Lombardi, S., Sunkavalli, K., Martin-Brualla, R., Simon, T., Saragih, J., Nießner, M., Pandey, R., Fanello, S., Wetzstein, G., Zhu, J.-Y., Theobalt, C., Agrawala, M., Shechtman, E., Goldman, D. B. and Zollhöfer, M.: *Computer Graphics Forum*, Vol. 39, No. 2, pp. 701–727 (online), DOI: 10.1111/cgf.14022 (2020).
- [7] Wang, L., Shi, X. and Liu, Y.: Foveated rendering: A state-of-the-art survey, *Comp. Visual Media*, Vol. 9, p. 195 – 228 (online), DOI: 10.1007/s41095-022-0306-4 (2023).
- [8] Luigi, L., Bolognini, D., Domeniconi, F., Gregorio, D., Poggi, M. and Di Stefano, L.: ScanNeRF: A Scalable Benchmark for Neural Radiance Fields, *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 816–825 (2023).
- [9] Engel, K., Hadwiger, M., Kniss, J. M., Lefohn, A. E., Salama, C. R. and Weiskopf, D.: Real-time volume graphics, *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, New York, NY, USA, Association for Computing Machinery, p. 29 – es (online), DOI: 10.1145/1103900.1103929 (2004).
- [10] Lee, J., Lee, S., Lee, J., Park, J. and Sim, J.: GScore: Efficient Radiance Field Rendering via Architectural Support for 3D Gaussian Splatting, *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, New York, NY, USA, Association for Computing Machinery, p. 497 – 511 (online), DOI: 10.1145/3620666.3651385 (2024).
- [11] Shimamura, K., Ohno, A. and Takamaeda-Yamazaki, S.: Exploring the Versal AI Engine for 3D Gaussian Splatting, *IEEE Symposium on Low-Power and High-Speed Chips and Systems (COOL Chips)*, Vol. 28 (2025).
- [12] Wang, S. and Kanwar, P.: BFloat16: The secret to high performance on Cloud TPUs (2019). <https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus?hl=en>.
- [13] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R. and Ng, R.: Mildenhall, Ben and Srinivasan, Pratul P. and Tancik, Matthew and Barron, Jonathan T. and Ramamoorthi, Ravi and Ng, Ren, *Commun. ACM*, Vol. 65, No. 1, p. 99 – 106 (online), DOI: 10.1145/3503250 (2021).
- [14] Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., Yang, J., Park, J., Heinecke, A., Georganas, E., Srinivasan, S., Kundu, A., Smelyanskiy, M., Kaul, B. and Dubey, P.: A Study of BFLOAT16 for Deep Learning Training (2019). <https://arxiv.org/abs/1905.12322>.
- [15] Fan, Z., Wang, K., Wen, K., Zhu, Z., Xu, D. and Wang, Z.: LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS (2023).
- [16] Hanson, A., Tu, A., Singla, V., Jayawardhana, M., Zwicker, M. and Goldstein, T.: PUP 3D-GS: Principled Uncertainty Pruning for 3D Gaussian Splatting, *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, pp. 5949–5958 (online), available from (<https://pup3dgs.github.io/>) (2025).
- [17] Xilinx: Xilinx/mlir-ai: An MLIR-based toolchain for AMD AI Engine-enabled devices. (2025). [https://github.com/Xilinx/mlir-ai/tree/main\[3rd, January, 2026\]](https://github.com/Xilinx/mlir-ai/tree/main[3rd, January, 2026]).
- [18] Barron, J. T., Mildenhall, B., Verbin, D., Srinivasan, P. P. and Hedman, P.: author = Barron, Jonathan T. and Mildenhall, Ben and Verbin, Dor and Srinivasan, Pratul P. and Hedman, Peter, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5470–5479 (2022).
- [19] Knapitsch, A., Park, J., Zhou, Q.-Y. and Koltun, V.: Tanks and temples: benchmarking large-scale scene reconstruction, *ACM Trans. Graph.*, Vol. 36, No. 4 (online), DOI: 10.1145/3072959.3073599 (2017).