Vol.2022-CG-185 No.2
2022/3/11

プライマリレイ走査高速化のための アフィン変換レイアライメント —Embreeを用いた実装—

西舘 祐樹[1,a)]　藤代 一成[1,b)]

概要：近年，リアルタイムレンダリングでもレイトレーシングが使われ始めているが，依然としてレイトレーシングの時間計算量は大きく，リアルタイムに出射できるレイの本数は限られている．本研究では，アフィン変換を用いて複数本のレイを 1 本にまとめることで，レイトレーシングの時間計算量を削減する新たな手法として，アフィン変換レイアライメントを提案する．提案手法をプライマリレイ走査に適用する実験の結果，レンダリング画像に誤差を発生させることなく，かつ空間計算量をほとんど増加させずに，時間計算量を削減できることが確認できた．

Affine-transformed ray alignment for accelerating primary ray traversal —Implementation with Embree—

Yuki Nishidate[1,a)]　Issei Fujishiro[1,b)]

Abstract: In recent years, ray tracing has begun to be used in real-time rendering, though the time complexity required for ray tracing is still large, and the number of rays that can be emitted in real-time is limited. In this study, we propose affine-transformed ray alignment as a novel method to reduce the time complexity of ray tracing by combining multiple rays into a single ray using affine transformation. As a result of experiments applying the proposed method to primary ray traversal, it was confirmed that the time complexity can be reduced with little increase in spacial complexity and without causing errors in the rendered images.

1. 序論

レイトレーシングは，写実的な画像をレンダリングするために用いられるアルゴリズムである．2018 年以降，GPU にレイトレーシング専用ハードウェアが搭載されたことにより，ゲームなどにおけるリアルタイムレンダリングでもレイトレーシングが使われ始めている．しかし，依然としてレイトレーシングの時間計算量は大きく，リアルタイムに出射できるレイの本数は限られている．

これまでにもレイの本数を削減する手法は数多く提案されているが，それらの手法は大幅に高速化できる反面，削減したレイによって得られるはずだった交差情報を損なうという本質的な欠点が存在する．これにより，最終的なレンダリング結果に誤差が生じる．

本研究では，アフィン変換を用いて複数本のレイを 1 本にまとめることで，レイトレーシングの時間計算量を削減する新たな手法として，アフィン変換レイアライメントを提案する．アフィン変換レイアライメント (Affine-Transformed Ray Alignment) は，1 本のレイから複数本分の交差情報を取得することで，レイ削減による交差情報の損失を解消する．そのため，レンダリング画像に一切誤差が生じない．先行報告 [1] に対して，本稿では Intel の高性能 CPU レイトレーシングライブラリである Embree を用いて高速化効果を向上させるとともに，多様なシーンを用いた評価実験によって本手法の実用性を示す．

提案手法をプライマリレイ走査に適用する実験の結果，空間計算量をほとんど増加させずに，時間計算量を削減できることが確認できた．

[1] 慶應義塾大学 理工学部
Faculty of Science and Technology, Keio University
[a)] Yuki.Nishidate@fj.ics.keio.ac.jp
[b)] fuji@ics.keio.ac.jp

2. 関連研究

レイトレーシングのコストの大部分は，シーン側の交差判定コストと，レイ側の本数によって変化する．本節ではレイトレーシングのコストを削減する手法を，交差判定コストを削減する手法と，レイの出射数を削減する手法に分けて紹介する．

2.1 交差判定コストを削減する手法

Bounding Volume Hierarchy (BVH) [2] はレイトレーシングを高速化するために利用される木構造である．標準的には，葉ノードにポリゴンへの参照を，内部ノードにはその子ノードを内包するバウンディングボックスをもつ．レイとの交差判定を行う際に BVH を走査することで，単純な線形探索と比較して，時間計算量を線形から対数に抑えることができる．

提案手法は Embree [5] を用いて実装されている．Embree では CPU の SIMD 機能を活用して交差判定を高速化するために，各ノードに子ノードを 4 つ，または 8 つもつ Wide BVH を利用する．

また，BVH には構築時間と品質のトレードオフが存在する．レイ走査を高速化するためには高品質な BVH を構築する必要があるが，それには構築に長い時間をかける必要がある．逆に，構築を高速化するためには，品質を犠牲にする必要がある．ひとつの BVH に対して多くのレイを飛ばすようなオフラインレンダリングでは品質が重視され，BVH を頻繁に更新する必要があるリアルタイムレンダリングでは構築時間が重視される．本手法において，BVH の構築時間と品質のバランスは自由に調節できるため，オフラインレンダリングとリアルタイム両方に応用可能である．

2.2 レイの出射数を削減する手法

レイを削減する手法には多様なアプローチがある．Lightcuts [6] は，多量の点光源を木構造として構築することでシャドウレイの本数を削減する．また，パストレーシングにおいては，レイのサンプル数を減らし，発生したノイズを画像空間でデノイズする手法 [3] が提案され，拡張手法が広く使われている．近年は，レンダリングする画像サイズを小さくしたうえで，深層学習によって超解像を行う手法 [7] も存在する．これらの手法は大幅に高速化できる反面，削減したレイによって得られるはずだった交差情報を損なうという本質的な欠点が存在する．そのため，最終的なレンダリング結果に誤差が生じることとなる．

提案手法は，レイを削減することでレイトレーシングを高速化するが，逆に交差判定コストの悪化を代償として，情報損失の問題を解決する．

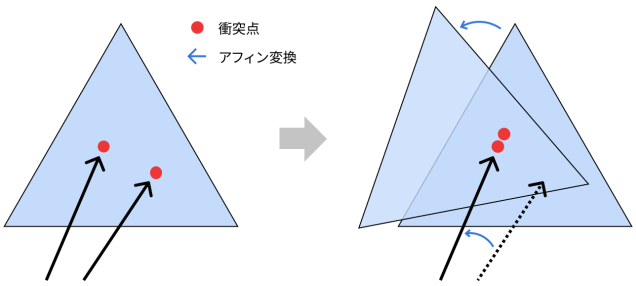


図 1: **アフィン変換レイアライメントの概要．レイとオブジェクトの両方に同一のアフィン変換を適用することで，レイの本数を減らしながら，変換前と同一の交差情報を取得する．**

3. 提案手法

本節ではアフィン変換レイアライメントを導入したあと，プライマリレイ走査への適用について詳説する．

3.1 アフィン変換レイアライメントの概要

アフィン変換レイアライメントは，アフィン変換を用いて複数本のレイを 1 本にまとめることでレイの本数を削減する手法である．通常はレイ削減に伴って交差情報も失われてしまうが，本手法では 1 本のレイから複数本分の交差情報を取得することで，損失した情報を補填する．その際，アフィン変換前のレイと同じ交差情報を取得する必要があるため，レイに適用したものと同一のアフィン変換をオブジェクトにも適用し，インスタンスを作成しておく．**図** 1 では，1 つのアフィン変換によって 2 本のレイを 1 本に束ね，それと同時に 1 枚のポリゴンを 2 枚に複製している．これにより，変換前と変換後でレイとポリゴンの相対位置を維持し，得られる 2 つの交点を一致させている．さらに，複製したインスタンスを含めて BVH を構築することで，インスタンシングによる計算時間の増加を対数時間に抑えた．

3.2 アフィン変換レイアライメントの高速化原理

BVH を利用した場合，レイトレーシングの計算時間は次式で表される:

$$\log(N_p) \times N_r \times C$$

ここで，N_p はポリゴンの枚数，N_r はレイの本数，C は環境に依存する係数を表す．この式から，レイの本数については線形で，ポリゴンの枚数については対数で計算時間に影響することが分かる．アフィン変換レイアライメントでは，レイを削減し，ポリゴン数を増加させる．例としてレイを 2 本から 1 本に，ポリゴンを 1 枚から 2 枚にした場合，計算時間は次式のように変化する:

$$\log(N_p \times 2) \times \frac{N_r}{2} \times C$$

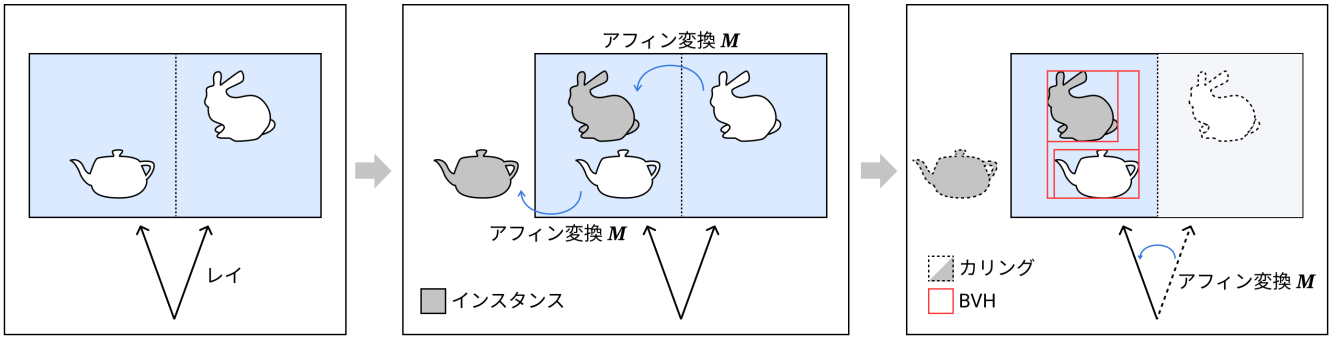


図 2: **アフィン変換レイアライメントを用いて，プライマリレイの本数を削減する例．レンダリング前に，オブジェクトを画像の半分だけ左に移動したインスタンスを作成したのち，BVH を構築しておく．レンダリング時には，画面左側半分にのみレイを出射し，1 本のレイで 2 本分の交差情報を取得することで，元のレイの本数と同一の情報が得られる．**

レイの削減によって線形に計算時間が小さくなると同時に，ポリゴンの増加によって対数で計算時間が大きくなる．これにより，式全体としては高速化が実現する．

3.3 アフィン変換レイアライメントの適用条件

アフィン変換レイアライメントを利用してレンダリングを高速化するには，次の 2 つの条件を満たす必要がある:

- ある 2 つのレイの集合が，ある 1 つのアフィン変換に対して全単射である．
- そのアフィン変換が，レンダリング前に既知である．

本手法を適用するには，あるレイ集合がアフィン変換によって別のレイ集合と完全に一致する必要がある．このため，2 つのレイ集合が 1 つのアフィン変換に対して全単射であるという条件が存在する．さらに，その 2 つのレイ集合が全体集合の直和分解となっている場合は，本手法による高速化が特に有効である．また，そのアフィン変換によってレンダリング前にポリゴンを複製する必要があることから，事前にアフィン変換が決定している必要がある．

3.4 プライマリレイへの適用

プライマリレイとは，視点から出射される第一のレイである．全てのレイの方向がレンダリング前に既知であるため，3.3 項の適用条件を満たす．

例として，**図** 2 に示すように画面を左右に 2 分割し，画面右側に出射するレイを左に移動すると，画面左側のレイに重なるため，本数を半減させることができる．また，レイを画面半分にのみ飛ばせばいいことが事前に決定するため，レイを飛ばさない領域のオブジェクトについては BVH 構築前にカリングすることも可能である．これにより，交差判定の時間計算コストを削減することができ，さらに高速化される．

画面は左右分割だけではなく，上下に分割することも可能である．さらに，左右と上下それぞれの分割数は，画像の縦横の画素数の約数であれば任意の値に設定できる．

本手法は，新しいデータ構造や，レイの新しいサンプリングアルゴリズムは必要なく，シーンを適切に複製するだけで実現できるため，実装もきわめて容易である．

4. 結果

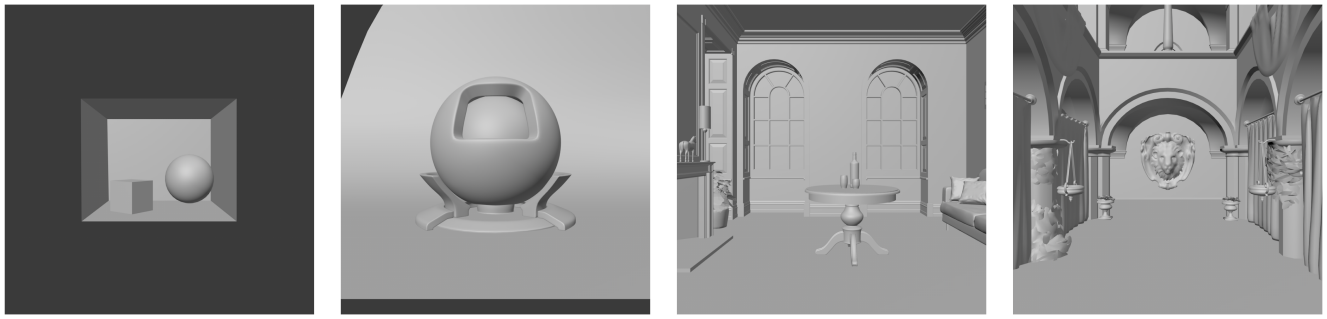
実験環境として，CPU: Intel Core i7-8700K 6 コア 3.70 GHz，RAM: 48.0 GB を用いた．また，シーンには 1,112 枚のポリゴンが含まれる Cornell box，61,600 枚のポリゴンが含まれる Mitsuba，143,173 枚のポリゴンが含まれる Fireplace room，262,267 枚のポリゴンが含まれる Sponza の 4 種類（**図** 3) を利用し，レンダリング画像は 1,024 × 1,024 画素とした．

4.1 縦横の分割数に対するレンダリング時間

縦横の分割数をそれぞれ 1 から 16 分割まで変化させ，そのレンダリング時間を計測した結果を**図** 4 に示す．ここで，縦横の分割数がどちらも 1 の場合は，提案手法を採用しない標準的なレンダリング手法となる．どのシーンにおいても，縦方向と横方向の両方を分割するよりも，どちらか一方だけを分割した方が高速化できることが分かった．また，縦方向と横方向の間に大きな差はないが，今回利用したシーンにおいては横方向に分割する方が若干高速であるため，これ以降は横方向にのみ分割を行うことにする．

4.2 横方向の分割数に対するレンダリング時間

横方向の分割数に対するレンダリング時間を，カリングを無効にした場合と有効にした場合で計測した結果を**図** 5 に示す．カリングを無効にした場合でも，すべてのシーンで提案手法によって高速化されたが，分割数を増やしすぎると高速化効果が低下し，シーンによっては逆に低速化する場合があった．これは，ポリゴンの枚数が増加しすぎるうえ，本手法によるポリゴンの移動量が小さすぎるために



(a) Cornell box (b) Mitsuba (c) Fireplace room (d) Sponza

図 3: **実験に使用したシーン**

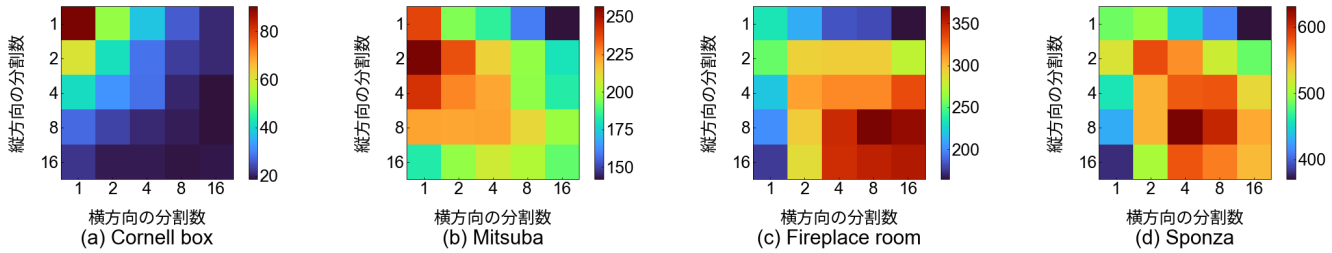


図 4: **縦横の分割数に対するレンダリング時間の変化**

大半のポリゴンが重なってしまい，BVH の品質が大きく低下することが原因であると考えられる．それに対して，不必要なポリゴンをカリングした場合は BVH の品質も保たれやすいため，分割数を増やすほど高速化効果が上がった．

カリングありの場合では，Cornell box で最大 7.50 倍速，Mitsuba で最大 2.94 倍速，Fireplace room で最大 1.85 倍速，Sponza で最大 1.71 倍速まで高速化された．

4.3 BVH 構築時間

レイトレーシングでは，レンダリング時間だけではなく BVH の構築時間についても考慮する必要がある．カリングを有効にした状態で，横方向の分割数に対する BVH 構築時間を計測した結果を**図** 6 に示す．本手法により分割数を増やすほど BVH 構築時間が増大するが，どのシーンにおいても 16 分割から 64 分割程度までであれば，影響は小さいといえる．

4.4 メモリ使用量

カリングを有効にした状態における，横方向の分割数に対するメモリ使用量を**図** 7 に示す．BVH 構築時間と同様に，分割数を増やすほどメモリ使用量も増大するが，16 分割から 64 分割程度までであれば，影響は小さかった．今回の実験では，単純にポリゴンをコピーしているが，インスタンシングを利用することで，さらにメモリ使用量を減らすことができる．

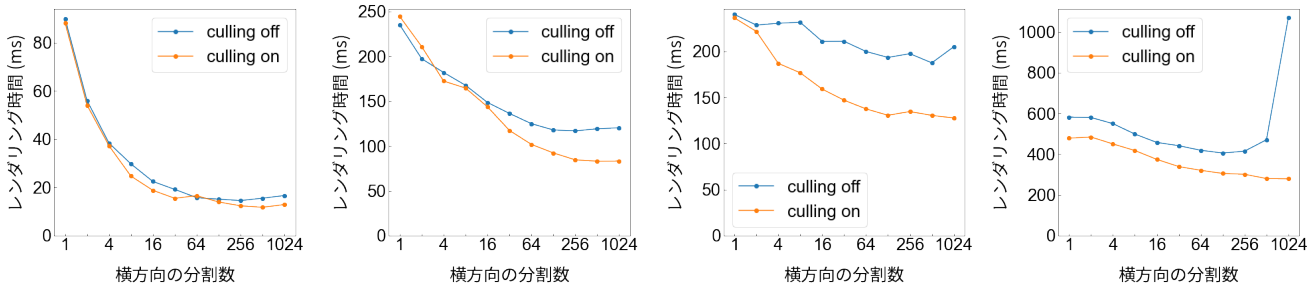
5. 制限

レイトレーシングでは，シーンに不透明物体しか含まれない場合は，レイの始点から最も近い交点を求めるだけで十分であることが多い．この場合，交差判定の際に BVH 全体を走査する必要がなくなり，早期に探索を終了することができる．しかし，本手法は 1 本のレイから複数本の交点を取得するという性質上，このような最適化を行うことはできない．そのため，最も近い交点のみを取得するような用途では，本手法を用いることで逆にレンダリングが低速化する場合がある．

6. 結論と今後の展望

本稿では，情報損失のないレイ走査の高速化手法として，アフィン変換レイアライメントを提案した．さらに，プライマリレイ走査への適用によって，レンダリングの高速化が確認され，空間計算量の増加も少なく，BVH の構築時間への影響も小さいことが実験的に確認できた．

現状では，特定のシーンに対してどのようなアフィン変換を行うのが最も高速化につながるのかを判断できない．よって，今後の課題として，本手法による高速化の効果を予測し，それによってシーンごとに最適なアフィン変換を推定することが考えられる．



(a) Cornell box (b) Mitsuba (c) Fireplace room (d) Sponza

図 5: **横方向の分割数に対するレンダリング時間**

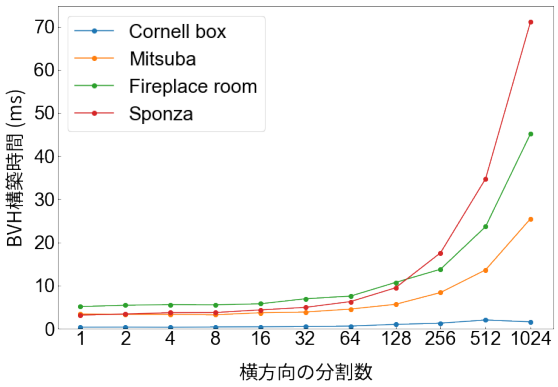


図 6: **横方向の分割数に対する BVH 構築時間**

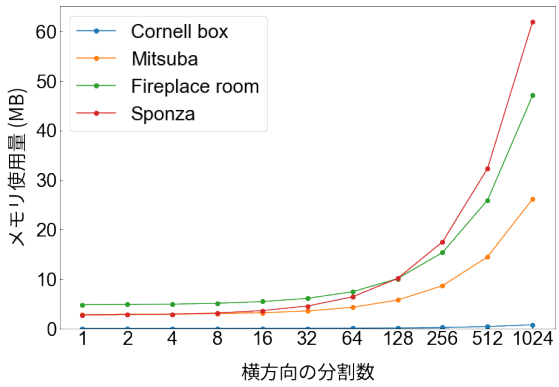
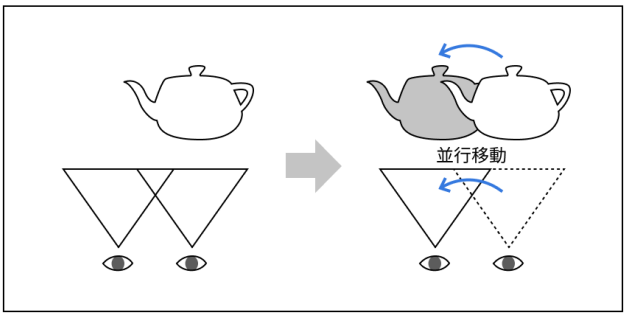


図 7: **横方向の分割数に対するメモリ使用量**

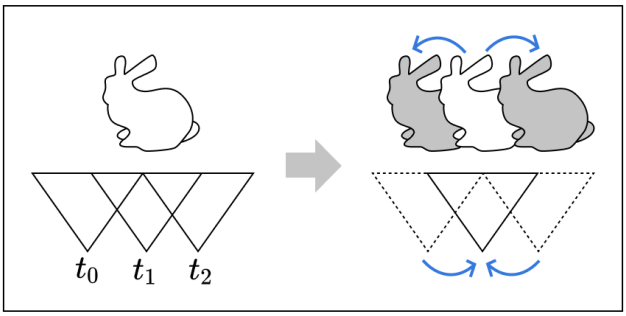
アフィン変換レイアライメントは，本稿で紹介した応用以外にも，3.3 項に示した適用条件を満たす問題に対しては応用可能である．現時点で考えられる応用例を**図** 8 に示す．

VR アプリケーションなどで必要とされるステレオレンダリングは，右左の視野用に 2 画面分をレンダリングする必要があるため時間計算量が大きい．ラスタライズであれば，右視野と左視野を 1 枚の画像にまとめてレンダリングすることでドローコールを削減する手法であるステレオインスタンシング [4] が使えるが，レイトレーシングでは画像をまとめてもレイの本数が減らないため，効果が得られない．そこで本手法を用いれば，右目から出射されるレイを左に並行移動することで，左目から出射されるレイに重ねることができる．これによってレンダリングを 1 画面分に抑えることができ，レンダリングの高速化が期待できる (**図** 8(a))．

さらに別の応用として，マルチフレームレンダリングが考えられる．本手法を用いれば，前後フレームで出射するレイを時間方向に重ねることも可能である．これにより複数フレームを一度にレンダリングすることができ，高速化できる可能性がある (**図** 8(b))．



(a) ステレオレンダリングへの応用



(b) マルチフレームレンダリングへの応用

図 8: **アフィン変換レイアライメントの応用例**

謝辞

本研究の一部は，令和 3 年度科研費挑戦的研究（開拓）20K20481 の支援により実施された.

参考文献

[1] 西舘 祐樹, 藤代 一成: 高速プライマリレイ走査のためのアフィン変換レイアライメント, 情報処理学会第 84 回全国大会講演論文集, 6ZF-01, 2022.
[2] J. H. Clark: “Hierarchical Geometric Models for Visible Surface Algorithms,” *Communications of the ACM*, Vol. 19, No. 10, pp. 547-554, 1976.
[3] H. E. Rushmeier and G. J. Ward: “Energy Preserving Non-Linear Filters,” in *Proceedings of ACM SIGGRAPH*, pp. 131–138, 1994.
[4] Unity: “Single Pass Instanced Rendering,” Unity Documentation, 2020, `https://docs.unity3d.com/2020.3/Documentation/Manual/SinglePassInstancing.html`（最終アクセス日: 2022-2-12）.
[5] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst: “Embree: A Kernel Framework for Efficient CPU Ray Tracing,” *ACM Transactions on Graphics*, Vol. 33, No. 4, Article 143, 2014.
[6] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg: “Lightcuts: A Scalable Approach to Illumination,” *ACM Transactions on Graphics*, Vol. 24, No. 3, pp. 1098-1107, 2005.
[7] L. Xiao, S. Nouri, M. Chapman, A. Fix, D. Lanman, and A. Kaplanyan: “Neural Supersampling for Real-Time Rendering,” *ACM Transactions on Graphics*, Vol. 39, No. 4, Article 142, 2020.