2023年度　卒業論文

曲線近傍を移動する視点から見えるステレオ自由視点画像の高速生成

Fast Synthesis of Stereo Novel Views Observed from a Moving Range near Curve

法政大学 情報科学部 ディジタルメディア学科

小池　優太郎
Yutaro Koike

指導教員　小池　崇文　教授

Abstract

This paper presents a system for fast synthesis of stereo novel views seen from a viewpoint moving near a curve. To achieve this, the author splits the scene into tube-like segments and distributed data to multiple neural networks for training and synthesis. Previous studies have presented two notable methods. One can handle a wide area but generates images at slower than 0.2 fps, and the other is faster but does not handle a wide area. This paper introduces three methods: scene decomposition, post-rendering scaling, and foveation to generate stereo free-viewpoint images faster and have performed a synthesis of those images moving on a 10 m path at the speed of nearly three fps. This paper also provides a method to allow automatic, determinable, and fast scene decomposition.

概要

本研究は曲線近傍を移動する視点から見えるステレオ自由視点画像を高速生成するシステムを提案する．それを実現するためにチューブ状の範囲でシーンを分割し，複数の MLP に分散させて学習させる．過去に自由視点画像を扱った研究が提案した手法として，広範囲を扱えるが 0.2 fps 未満で生成する手法，他方で高速であるが広範囲を扱わない手法がある．本研究は 3 つの手法: シーンデコンポジション，レンダリング後拡大，foveation を導入することで，10 m 以上移動できるステレオ自由視点画像を 3 fps 弱の速度で生成した．特にシーンデコンポジションについては自動的かつ確定的にシーンを分割する方法を提案する．

目次

Chapter 1

はじめに

1.1 背景

Research & Development Group, Hitachi, Ltd. が定める自由視点の定義は「三次元画像または映像において，画像または映像を撮影したカメラそのものが提供する視点ではなく，ユーザが自由に選択できる視点のこと」である [1]．自由視点画像は自由視点から見た画像である．自由視点画像を生成する技術は，ドローンが撮影するような映像を撮影現場に居なくとも生成できる技術である．

1.2 目的

広範囲なシーン内を自由視点画像で見ることができれば，広大な公園内や巨大な建物内などをスムーズに散策する体験をディスプレイを通じて提供できる．しかし問題として，データサイズの増加により自由視点画像生成システム速度が低下することが予測される．そのため本研究は視点移動可能範囲を広げつつも，より高速に自由視点画像を生成するシステムを開発することを目指す．

広範囲なシーン内を移動できる自由視点画像を目指した研究は多々ある．1 つの MLP が学習できる自由視点画像の移動範囲には情報量の観点から限界がある．また，十分な容量を持つ MLP であったとしても，広大なシーンを扱うと全く特徴の異なる風景があり，解の収束に時間が掛かる可能性がある．単一 MLP における課題点に着目した先行研究の中には広範囲の自由視点画像を生成することに特化したものがある．例えば Mega-NeRF，Block-NeRF，Switch-NeRF はいずれもシーンを分割し別々の MLP に学習させることで自由視点画像の移動範囲を広げることおよび解の収束を早めることを目指した．その 3 つの研究は同じイテレーションでより高い PSNR, SSIM 類似度の自由視点画像を生成した．

Mega-NeRF，Block-NeRF，Switch-NeRF は広範囲なシーンを扱うことを目指したが，生成速度は 1 fps 未満であった．このため著者は高速な自由視点画像生成を目指した別の研究に着目した．FoV-NeRF は高速なステレオ自由視点画像に特化している．この研究は foveation という高速化手法を適用し，1 枚生成する時間を 562 ミリ秒から 27 ミリ秒まで短縮した．

foveation とは人間の注視点から離れた箇所の解像度を抑えたりすることで高速化する手法である．

1.3 本論文の構成

第 2 章では本研究の基礎を支える前提知識・前提技術を説明する．第 3 章では自由視点画像に関する研究 NeRF とそれを応用した研究を取り挙げる．第 4 章では本研究で提案する手法の詳細を述べる．第 5 章では提案手法が動作するかを確認し，自由視点画像を生成する速度と品質を評価する．第 6 章では本研究の総括と今後の課題を述べる．

Chapter 2

前提知識

2.1 数式について

本ドキュメントにおける数式のルールを説明する．本ドキュメントではスカラー，ベクトル，行列 (2 次元行列)，関数を分けて表記している．表記分けは以下のようになっている．

- スカラー: 「太字ではない」かつ「直後に () が付いていない」を満たす「小文字」
- ベクトル: 「太字である」かつ「直後に () が付いていない」を満たす文字
- 行列: 「太字ではない」かつ「直後に () が付いていない」を満たす「大文字」
- 関数: 直後に () が付いた文字

ベクトルは特段指定が無い限り横ベクトルとする．ある行列 A の要素を参照する際の表記は $A[x, y]$ と $A_{i,j}$ 2 通りの方法がある．$A[x, y]$ と表記する場合は左から $x + 1$ 番目，上から $y + 1$ 番目の要素を指す．一方 $A_{i,j}$ と表記する場合は左から $j + 1$ 番目，上から $i + 1$ 番目の要素を指す．すなわち $A[x, y] = A_{y,x}, A_{i,j} = A[j, i]$ である．例えば $A[5, 0]$ と $A_{0,5}$ はいずれも最も上の行にある左から 6 番目の要素を指す．

実数全体の集合を $\mathbb{R}$，整数全体の集合を $\mathbb{Z}$，非負整数全体の集合を $\mathbb{Z}_{\geq 0}$ と表す．そのため $n \in \mathbb{Z}_{\geq 0}$ は n が非負整数であることを示す．

本ドキュメントに載っている演算子を説明する．入力を表す変数として実数 a，整数 m, n，ベクトル $\boldsymbol{v} \in \mathbb{R}^3$ を定義する．本ドキュメントに載っている演算子は以下のように定義される．

- $\lfloor a \rfloor$: a の整数部分
- $\lfloor \lfloor a \rfloor \rfloor$: a の小数部分 $= a - \lfloor a \rfloor$
- $\mathrm{mod}(m, n)$: m を n で割った余り
- $||\boldsymbol{v}||$: $\boldsymbol{v}$ のノルム $= \sqrt{\boldsymbol{v} \cdot \boldsymbol{v}}$

2.2 画像類似度指標について

本研究では 2 画像間の類似度を測るために画像類似度指標 PSNR, SSIM, LPIPS を用いた．PSNR と SSIM はそれぞれ式 2.1 と式 2.2 で定義される [2]．PSNR と SSIM は値が高いほど類似度が高いことを示す．A, B はグレースケール画像を表す 2 次元行列である．A と B は同じ幅と高さを持ち，U は画素座標の集合，n は A の画素数とする．A, B の画素値は 0 以上 255 以下の整数とする．

$$\mu = \frac{1}{n}\sum_{(x,y)\in U}(A[x,y]-B[x,y])^2, \mathrm{PSNR}(A,B) = 10 * \log_{10}\left(\frac{255^2}{\mu}\right). \tag{2.1}$$

$$\begin{aligned}
c_1 &= (0.01\times 255)^2, c_2 = (0.03\times 255)^2 \\
\mu_A &= \frac{1}{n}\sum_{(x,y)\in U} A[x,y], \mu_B = \frac{1}{n}\sum_{(x,y)\in U} B[x,y] \\
\sigma_A &= \sqrt{\frac{1}{n}\sum_{(x,y)\in U}(A[x,y]-\mu_A)^2}, \sigma_B = \sqrt{\frac{1}{n}\sum_{(x,y)\in U}(B[x,y]-\mu_B)^2} \\
\sigma_{AB} &= \frac{1}{n}\sum_{(x,y)\in U}(A[x,y]-\mu_A)(B[x,y]-\mu_B) \\
\mathrm{SSIM}(A,B) &= \frac{(2\mu_A\mu_B+c_1)(2\sigma_{AB}+c_2)}{(\mu_A^2+\mu_B^2+c_1)(\sigma_A^2+\sigma_B^2+c_2)}.
\end{aligned} \tag{2.2}$$

LPIPS は学習ニューラルネットワークを用いて画像類似度を測る指標である [3]．LPIPS は値が低いほど画像類似度が高いことを示す．

2.3 ボリュームレンダリング

ボリュームレンダリングは物体を密度で表現することで，ポリゴンでは表されない雲や粒子システム等をレンダリングする技術である [4]．ボリュームレンダリングはボリューム密度という概念を導入している．ボリューム密度とは 3 次元空間上のある 1 点においてどれほど物体が密集しているかを表す．ボリュームレンダリングにおいてのボリューム密度はいわゆる画像の不透明度に近い概念である．ボリューム密度が高い地点はより多くの光量を反射し，低い地点はほぼ光を反射せず真空のような振る舞いをする．シーンの表現には 3 次元点座標を入力するとその地点の色とボリューム密度を返す関数を用いる．

ある 3 次元座標を $\boldsymbol{x} \in \mathbb{R}^3$ とする．ボリューム密度 $\sigma(\boldsymbol{x}) \in$ と RGB 値 $\boldsymbol{c}(\boldsymbol{x})$ が既知の時，レイ $\boldsymbol{r}(t) = \boldsymbol{O} + t\boldsymbol{d}$ に対応する色 $\boldsymbol{C}(\boldsymbol{r})$ は式 (2.4) で求められる．なお $t_n, t_f (t_n \leq t_f)$ は限界を表す実数であり，本研究では $t_n = 0, t_f = 1$ とした．コンピュータ上では式 (2.4) を離散的に求めており，その計算に用いるサンプリング点の個数 (式 (2.3) の n) をレイサンプリング数と本研究では呼ぶ．一般にレイサンプリング数が高いほど PSNR 値が高くなる．なお式 (2.4) の $T(\boldsymbol{r}, t)$ はレイが $[t_n, t]$ 区間内で通過するボリューム密度が総じて低いという確率である．レイが $[t_n, t]$ 区間内で高ボリューム密度点を通過すると，$T(\boldsymbol{r}, t)$ が減少し，$\sigma(\boldsymbol{r}(t_i))\boldsymbol{c}(\boldsymbol{r}(t_i))$

が薄くなるという仕組みである．$T(\boldsymbol{r}, t)$ を用いる理由は，レイ原点に近いボリューム密度と RGB 値を大きな配分で $\boldsymbol{C}(\boldsymbol{r})$ に反映させるためである．

$$
\begin{aligned}
t_i &= \frac{i}{n-1}(t_f - t_n) + t_n, \\
\boldsymbol{C}(\boldsymbol{r}) &= \frac{1}{n}\sum_{i=0}^{n-1} T(\boldsymbol{r}, i)\sigma(\boldsymbol{r}(t_i))\boldsymbol{c}(\boldsymbol{r}(t_i)), \\
T(\boldsymbol{r}, i) &= \exp\left(-\frac{1}{i+1}\sum_{j=0}^{i}\sigma(\boldsymbol{r}(t_j))\right).
\end{aligned}
\tag{2.3}
$$

$$
\begin{aligned}
\boldsymbol{C}(\boldsymbol{r}) &= \int_{t_n}^{t_f} T(\boldsymbol{r}, t)\sigma(\boldsymbol{r}(t))\boldsymbol{c}(\boldsymbol{r}(t))dt, \\
T(\boldsymbol{r}, t) &= \exp\left(-\int_{t_n}^{t}\sigma(\boldsymbol{r}(s))ds\right).
\end{aligned}
\tag{2.4}
$$

2.4 Neural Radiance Fields

Mildenhall らは少ない入力画像から複雑かつ連続的なシーンを合成し，高品質な自由視点画像を得る手法 Neural Radiance Fields (以下「NeRF」という) を提案している [5]．彼らはラディアンスを持つ点が無数に存在する空間，Radiance Fields を関数 (2.5) で表現している．関数 (2.5) は，点の 3 次元座標 $\boldsymbol{x} = (x, y, z)$ と 2 次元視点方向 $\boldsymbol{d} = (\theta, \phi)$ を入力すると，その点におけるラディアンスの RGB 色 $\boldsymbol{c} = (r, g, b)$ とボリューム密度 σ を出力する．彼らは関数 (2.5) を実現するために多層パーセプトロン (以下「MLP」という) を用いている．MLP はパーセプトロンが幾層にも重なって構成されるニューラルネットワークである [6]．関数 (2.5) から自由視点画像を生成するには $(\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{d}), \sigma(\boldsymbol{x})) = F(\boldsymbol{x}, \boldsymbol{d})$ と定義し，ボリュームレンダリングをおこなう．ボリュームレンダリングをおこなって生成された自由視点 RGB 画像を $\hat{R}_i, \hat{G}_i, \hat{B}_i$ と定義する．

$$
F : (\boldsymbol{x}, \boldsymbol{d}) \rightarrow (\boldsymbol{c}, \sigma). \tag{2.5}
$$

NeRF が自由視点画像を生成するまでの過程には学習フェーズと生成フェーズがある．学習フェーズで必要なデータは学習用 RGB 画像 R_i, G_i, B_i と画像各画素に対応するレイ $\boldsymbol{o}_i[x, y], \boldsymbol{d}_i[x, y]$ である．R_i, G_i, B_i は画像の赤・緑・青チャンネルを表す行列である．学習フェーズは損失 l(式 (2.6)) を最小化するフェーズである．式 (2.6) は集合 U は画素座標の集合である．学習フェーズは以下 4 ステップで構成される．

1. 全ての入力画像を見て RF を実装する全結合な多層パーセプトロンに $(\boldsymbol{c}, \sigma)$ の初期値を学習させる．

2. 入力画像と同じカメラ位置・向きからの視点から見える箇所に対応する $(\boldsymbol{c}, \sigma)$ を MLP から得る．

3. 得た $(\boldsymbol{c}, \sigma)$ を用いてボリュームレンダリングをおこない入力画像に対応する自由視点画像を得る．

4. 自由視点画像と入力画像の差を比較し MLP に学習させる．

学習フェーズを終えた後の生成フェーズでは $\hat{R}_i, \hat{G}_i, \hat{B}_i$ を求めて出力する．

$$l = \sum_{(x,y)\in U} (R_i[x,y] - \hat{R}_i[x,y])^2 + (G_i[x,y] - \hat{G}_i[x,y])^2 + (B_i[x,y] - \hat{B}_i[x,y])^2 \qquad (2.6)$$

Chapter 3

関連研究

Mildenhall らはニューラルネットワークを用いてラディエンスフィールドを実装する手法 NeRF を提案した (詳細は節 2.4 を参照されたし)．Mildenhall が提示したデモ画像は小物や室内などいずれも狭い範囲を写したものである．他方で Tancik らは町通りの風景を自由視点画像で閲覧可能にする研究をおこなっている [7]．シーンを範囲ごとに分割し各範囲を別々の MLP に学習させることで，各 MLP が保持する平均データサイズを軽減している．しかしそこで提案された手法が自由視点画像を生成する速度は 0.2 fps 未満である．

Mildenhall らが提案した手法をステレオ画像に拡張し，ステレオ画像の性質を利用して高速化した研究もある．Deng らは NeRF を改良する形で，ステレオ自由視点画像の見え方を利用して高速化を試みている [8]．Deng らは解像度 1,440 [px] × 1,600 [px], 視野角 110 度の HMD 上に自由視点画像を 50 fps で生成する手法『FoV-NeRF』を提案した．両眼視差のずれを軽減したり，注視点から離れた部分の解像度を抑えたりなど，HMD 等のステレオディスプレイに特化した品質向上の工夫が施されている．一方でこの手法は固定された点 1 つを原点とする座標系を用いており，その点から離れた視点からの画像生成に非対応，すなわち歩くほどの距離移動ができない問題がある．

先行研究にはシーンデコンポジションの方法にも課題点がある．Block-NeRF は街中のシーンを交差点を中心とした球状の範囲に分割している．これは人間が恣意的に定めるため精確性が保証されないことや大量にデータがある場合に時間がかかることが課題である．また Switch-NeRF は訓練可能なゲートを用いて複数の MLP を選択する機能を実装している [9]．しかし訓練可能なゲートを用いることは非確定的な選択や学習時間が掛かることに繋がる．本研究では自動的かつ確定的シーンデコンポジションをおこなう方法を提案する．

本研究は Tancik らの広範囲自由視点画像技術と FoV-NeRF の高速なステレオ自由視点画像技術を組み合わせ，実用的な速度で自由視点画像を生成するシステムを開発することを目指す．具体的には 10 m 以上伸びた曲線上を移動する視点から見たステレオ自由視点画像を 1 fps 以上で生成する．

Chapter 4

提案手法

本研究は 2 つの関連研究を参考に，曲線上の視点から見たステレオ自由視点画像を，高速に生成する手法を提案する．1 つ目の関連研究である Tancik らの研究は節 4.2 にて活用し，もう 1 つの関連研究 FoV-NeRF は節 4.7 にて活用する．本研究の特徴は視点移動範囲を「曲線上」に制限することである．視点位置を空間上に存在する任意の 3 次元点ではなく曲線上に制限する理由は 2 つある．

1 つ目は自由視点画像生成用の素材を撮影することが容易になるためである．視点位置を空間上に存在する任意の 3 次元点にすると，様々な角度から撮った写真が必要である．一方で視点位置を曲線上に制限すると，その曲線付近からの撮影で十分であり，より少ない写真からでも高品質な画像を生成できると考える．

2 つ目はより自然な使用用途に合っているためである．人間は歩行や自動車などで移動するため，多くのツアーやテーマパークライドなどのように，一定のコースに沿って視点が変化することが自然である．したがって，視点移動を曲線上の移動に制限しても自然な使用用途には問題ないと考えるためである．

4.1 ニューラルネットワーク設定

本研究で用いる MLP は Tiny CUDA Neural Network Framework (以下「TinyCudaNN」という) である．TinyCudaNN は CUDA GPU 内のレジスタや Shared Memory などの高速なメモリを活用することで，Tensorflow より高速に動作する Neural Network を提供する [10]．本研究ではボリューム密度を返すネットワークと RGB 値を返すネットワークを実装した．ボリューム密度を返すネットワークは隠れ層 1 層，幅 64 の全結合多層パーセプトロンである．入力値は 3 次元座標の 3 値 (x, y, z) である．RGB 値を返すネットワークは隠れ層 2 層，幅 64 の全結合多層パーセプトロンである．入力値は 3 次元座標と方向の 6 値 (x, y, z, d_x, d_y, d_z) である．両ネットワークで活性化関数 ReLU $= \max(x, 0)$ を用いた．最適化アルゴリズムには Adam[11] を用いる．

解の収束を速めるため，両ネットワークにはエンコーディングも施した．エンコーディングとは入力値の個数を特定のアルゴリズムにより増やすことで，ネットワークが学習に用い

るデータを増大させる工夫である．ボリューム密度を返すネットワークには Multiresolution hash エンコーディングを用いた [12]．レベル数は 16，ハッシュテーブルのエントリーサイズは 2^{19}，ハッシュテーブルエントリー内の要素数は 2 である．

RGB 値を返すネットワークには Frequency エンコーディングを用いた．これはサイン関数とコサイン関数を用いたエンコーディングであり，NeRF でも用いられた．RGB 値ネットワークに用いた Frequency エンコーディングは 1 つの実数 x を 24 個の実数 $\sin \pi x, \cos \pi x, \sin 2\pi x, \cos 2\pi x, \sin 4\pi x, \cos 4\pi x, \sin 8\pi x, \cos 8\pi x, ..., \sin 2^{11}\pi x, \cos 2^{11}\pi x$ に拡張するエンコーディングである．RGB 値を返すネットワークの入力値は 3 値であるため，72 値まで拡張することとなる．

4.2 シーンデコンポジション

提案手法はシーンを図 4.1 に示されるようなチューブ状の範囲に区切る．この工夫をシーンデコンポジションと呼ぶ．チューブ状の範囲をセグメントと呼び，セグメントを定める軸となる曲線をセグメント基軸と呼ぶ．1 セグメントごとに 1 つの多層パーセプトロンを対応させることで，並列学習を可能にする．u を 0 以上 1 未満の実数，曲線を $\boldsymbol{f}(u)$ としたとき，ある 3 次元座標 $\boldsymbol{x}$ がどのセグメントに属しているかは $||\boldsymbol{x} - \boldsymbol{f}(u)||$ を最小にする u を求めることで判定可能である．本研究では軸となる曲線は非一様 2 次の B スプライン曲線を採用する．

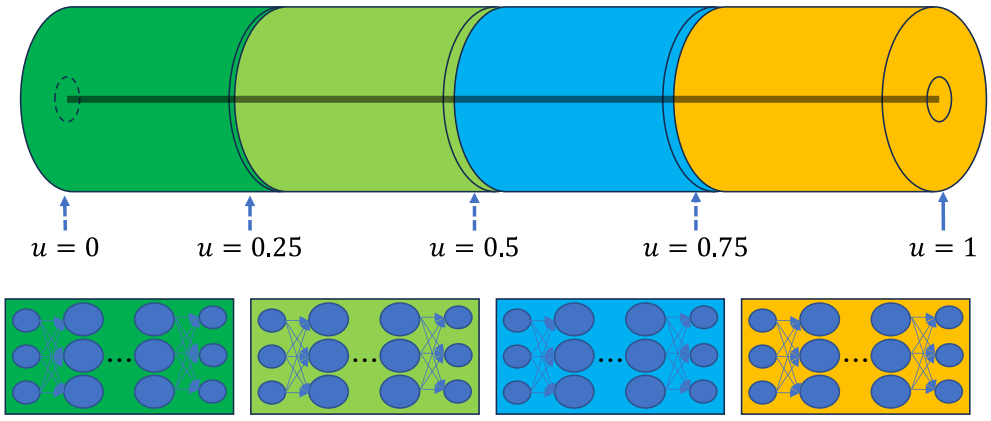

図 4.1: セグメント基軸 (中央黒い線) と 4 つのセグメント (緑・黄緑・水色・オレンジ)．セグメント基軸は 3 次元空間上を通り，その周りをシーン分割範囲 (セグメント) とする．1 セグメントにつき MLP が割り当てられる．

4.3 セグメント基軸の決定

m 個の実視点位置の 3 次元直交座標を $\boldsymbol{V}_0, \boldsymbol{V}_1, ..., \boldsymbol{V}_{m-1}$ と置く．これらの点の間を通過する曲線を一意に求め，セグメント基軸として用いる．この曲線は n 個の 3 次元制御点 $\boldsymbol{C}_0, \boldsymbol{C}_1, ..., \boldsymbol{C}_{n-1}$ とノット列 $k_0, k_1, ..., k_{n+2}$ を元に式 (4.1) で定義される．d 次の B スプライン曲線のノット数は $n + d + 1$ で計算できるため，今回は $n + 3$ である．ノット列は式

(4.2) で定義される．このノット列は最初の 3 値が 0，最後の 3 値が 1 と揃っているため，$\boldsymbol{C}_0 = \boldsymbol{V}_0, \boldsymbol{C}_{n-1} = \boldsymbol{V}_{m-1}$ となる．ただし式 (4.1) に含まれる $B_i^n(t)$ は B スプライン基底関数である．

$$\boldsymbol{f}(t) = \sum_{i=0}^{n-1} B_i^2(t)\boldsymbol{C}_i. \tag{4.1}$$

$$k_i = \begin{cases} 0 & (i = 0, 1, 2) \\ \frac{i-2}{n-2} & (3 \leq i < n) \\ 1 & (i = n, n+1, n+2) \end{cases} . \tag{4.2}$$

$$\begin{aligned} B_i^0(t) &= \begin{cases} 1 : t \in [k_i, k_{i+1}) \\ 0 : t \notin [k_i, k_{i+1}) \end{cases} , \\ B_i^n(t) &= \frac{t - k_i}{k_{i+n} - k_i} B_i^{n-1}(t) \\ &\quad + \frac{k_{i+n+1} - t}{k_{i+n+1} - k_{i+1}} B_{i+1}^{n-1}(t). \end{aligned} \tag{4.3}$$

$\boldsymbol{V}_0, \boldsymbol{V}_1, ..., \boldsymbol{V}_{m-1}$ から 3 次元制御点 $\boldsymbol{C}_0, \boldsymbol{C}_1, ..., \boldsymbol{C}_{n-1}$ を得るために Jing らの論文に掲載された「最小二乗法を用いた B スプライン曲線フィッティング」の方法を導入する [13]．数式の記述範囲を節約するため $\boldsymbol{V}_i, \boldsymbol{C}_i$ は横ベクトルとする．この方法は $\sum_{i=0}^{m-1} ||\boldsymbol{V}_i - \boldsymbol{f}(u_i)||$ が最小になるような $\boldsymbol{V}_i$ を得る方法である．u_i は $\boldsymbol{V}_i$ の地点を示す値で，当アルゴリズムは $\boldsymbol{f}(u_i)$ が $\boldsymbol{V}_i$ に近づくようフィッティングする．このとき制御点は方程式 (4.6) を解くことで得られる．

$\boldsymbol{V}_i$ に対応する u_i は主成分分析を用いて求める．この方法は初期値としてある 3 次元座標 $\boldsymbol{O}$ が必要である．最初に m 個の 3 次元座標 $\boldsymbol{V}_i$ の分布に関する第一主成分 $\boldsymbol{w}$ を求める．$\boldsymbol{w}$ は式 (4.4) を満たす 3 次元ベクトルである．$\boldsymbol{w}$ を求める問題は行列 S に対する固有値問題に等しい．$\boldsymbol{w}$ に複数の候補がある場合は実数 λ が最も大きい場合の候補を採用する．次に $||\boldsymbol{V}_i - \boldsymbol{O}||$ が最小となる座標 $\boldsymbol{V}_o$ を求める．続いて各座標 $\boldsymbol{V}_i$ に対して $d_i = \boldsymbol{w}^T(\boldsymbol{V}_i - \boldsymbol{V}_o)$ を求める．最後に d_i が n 番目に小さい時に $u_i = \frac{n-1}{m-1}$ と定める．

$$
\begin{aligned}
&(x_i, y_i, z_i) = \boldsymbol{V}_i, \\
&\qquad \mu_x = \frac{1}{m}\sum_{i=0}^{m-1} x_i, \mu_y = \frac{1}{m}\sum_{i=0}^{m-1} y_i, \mu_z = \frac{1}{m}\sum_{i=0}^{m-1} z_i, \\
&\qquad S = \frac{1}{m}\sum_{i=0}^{m-1} \left\{ \begin{pmatrix} (x_i-\mu_x)^2 & (x_i-\mu_x)(y_i-\mu_y) & (x_i-\mu_x)(z_i-\mu_z) \\ (x_i-\mu_x)(y_i-\mu_y) & (y_i-\mu_y)^2 & (y_i-\mu_y)(z_i-\mu_z) \\ (x_i-\mu_x)(z_i-\mu_z) & (y_i-\mu_x)(z_i-\mu_y) & (z_i-\mu_z)^2 \end{pmatrix} \right\}, \\
&\qquad S\boldsymbol{w} = \lambda \boldsymbol{w}.
\end{aligned}
\tag{4.4}
$$

方程式 (4.6) にある行列 A は式 (4.5) により定義される．なおノット列を始める最初の 3 値が 0，最後の 3 値が 1 と揃っているため，$\boldsymbol{C}_0 = \boldsymbol{V}_0, \boldsymbol{C}_{n-1} = \boldsymbol{V}_{m-1}$ である．$\boldsymbol{q}_i = \boldsymbol{V}_i - \boldsymbol{V}_0 B_0^2(u_i) - \boldsymbol{V}_{n-1} B_{n-1}^2(u_i)$ と定義する．

$$
A = \begin{pmatrix} B_1^2(u_1) & B_2^2(u_1) & \dots & B_{n-2}^2(u_1) \\ B_1^2(u_2) & B_2^2(u_2) & \dots & B_{n-2}^2(u_2) \\ \vdots & & & \\ B_1^2(u_{m-1}) & B_2^2(u_{m-1}) & \dots & B_{n-2}^2(u_{m-1}) \end{pmatrix}. \tag{4.5}
$$

$$
A^T A \begin{pmatrix} \boldsymbol{C}_1 \\ \boldsymbol{C}_2 \\ \vdots \\ \boldsymbol{C}_{n-1} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n-2} B_1^2(u_1)\boldsymbol{q}_i \\ \sum_{i=1}^{n-2} B_2^2(u_2)\boldsymbol{q}_i \\ \vdots \\ \sum_{i=1}^{n-2} B_{n-1}^2(u_{m-1})\boldsymbol{q}_i \end{pmatrix}. \tag{4.6}
$$

4.4 高速なセグメント特定

ある 3 次元座標の点がどのセグメントに属するかを特定することをセグメント特定と呼ぶ．本研究では 1 枚の自由視点画像を生成するために新規視点原点 (x, y, z) に関してセグメント特定する．高速なセグメント特定を実現するためにキャッシングと kd 木を用いたアルゴリズムを導入し，(x, y, z) から $\hat{t}$ を求める処理を高速化する．

前準備として $\boldsymbol{P}_i = \boldsymbol{f}(\frac{i}{n-1})(0 \le i \le n-1)$ を求め，i を与えれば $\boldsymbol{P}_i$ を返す kd 木を用意する．さらに $\boldsymbol{f}(u)$ を近似する関数 $\hat{\boldsymbol{f}}(u)$ を 式 (4.7) より定義する．プログラミング的には $\boldsymbol{P}_i$ を配列にキャッシュしておくことで $\hat{\boldsymbol{f}}(u)$ は実装可能である．

$$
\begin{aligned}
i &= \lfloor u(n-2) \rfloor, \\
\alpha &= u(n-2) - i, \\
\hat{\boldsymbol{f}}(u) &= (1-\alpha)\boldsymbol{P}_i + \alpha \boldsymbol{P}_{i+1}.
\end{aligned} \tag{4.7}
$$

セグメント特定をおこなうアルゴリズムは以下のステップに沿って $\boldsymbol{x} = (x, y, z)$ から $\hat{t}$ を求める．このアルゴリズムは並列化可能であるため，並列分散処理による更なる高速化も可能である．

1. kd 木に $\boldsymbol{x}$ を提示し，$\boldsymbol{P}_i$ 中から $||\boldsymbol{x} - \boldsymbol{P}_i||$ が最も小さくなる 5 点 $\boldsymbol{P}_a, \boldsymbol{P}_b, ..., \boldsymbol{P}_e$ を求める．なお，a, b, c, d, e は互いに異なる．

2. $u_a = a + \frac{1}{n} \frac{(\boldsymbol{P_{a+1}} - \boldsymbol{P_a}) \cdot \boldsymbol{x}}{||\boldsymbol{P_{a+1}} - \boldsymbol{P_a}||^2}$ とする．

3. 定義されている $\hat{\boldsymbol{f}}(u_a), \hat{\boldsymbol{f}}(u_b), \hat{\boldsymbol{f}}(u_c), \hat{\boldsymbol{f}}(u_d), \hat{\boldsymbol{f}}(u_e)$ の内，$\boldsymbol{x}$ に最も近い点の引数を u' と置く．例えば $\hat{\boldsymbol{f}}(u_a)$ が定義されていて $(\boldsymbol{x}, \hat{\boldsymbol{f}}(u_a))$ 間の距離が最も近い場合は $\hat{u} = u_a$ とする．

4. $\boldsymbol{x}$ とユークリッド距離的に最も近い $\boldsymbol{f}(u)$ 上にある点の座標は $\boldsymbol{f}(\hat{u})$ である．

4.5 画像分割

大きな学習用画像を扱う際に GPU メモリ不足が発生した．この問題に対処するため，ピクセル毎分割を施す．画像を分割する方法はピクセル毎分割とグリッド毎分割がある．

ピクセル毎分割とグリッド毎分割を説明するために必要な変数を定義する．最初に幅 w [px]，高さ h [px] のグレースケール画像 G を定義する．G の画素値への参照は行列と同じ表記を用いる．したがって $G[25, 46]$ は左から 26，上から 47 番目の画素値を指す．画像分割後にできる小さな画像をパネルと呼ぶ．横方向への分割枚数を d_x，縦方向への分割枚数を d_y とすると，出来上がるパネルの個数は $d_x d_y$ である．パネルのインデックスを $0 \leq i < d_x \times d_y$ と定義した時，各パネルを F_i と表記する．

ピクセル毎分割は $F_i(x, y) = G[\mathrm{mod}(i, d_x) + x d_x, \lfloor i/d_x \rfloor + y d_y]$ と画素値をパネルに配置する．一方グリッド毎分割は $F_i(x, y) = G[x + \mathrm{mod}(i, d_x) \times w/d_x, y + \lfloor i/d_x \rfloor \times h/d_y]$ と画素値をパネルに配置する．

学習画像群からランダムに選択し学習する方式を採っているため，グリッド毎分割は画像の左上ばかり学習してしまうことがあった. これでは画像の右下に写っている部分が全く学習されない．ゆえに，ピクセル毎分割法を採用した．

4.6 レンダリング後拡大

MLP は膨大な時間計算量を必要とする可能性があるため，MLP への依存度を軽減する高速化手法「レンダリング後拡大」を導入する．レンダリング後拡大は目標となる画像を縮小し生成する．次にバイリニア補間付き拡大をおこない生成画像を得る手法である．本研究では縦に 1/2 倍，横に 1/2 倍まで縮小することで，ニューラルネットワークへの入力を 1/4 倍まで縮小する．

バイリニア補間付き拡大を説明するため，節 4.5 で定義したグレースケール画像 G を流用する．加えて幅が $w'(>w)$ [px]，高さが $h'(>h)$ [px] であるグレースケール画像 G' を用意する．G から G' へ拡大する時，バイリニア補間付き拡大は式 4.8 に示されるように画素値を配置する．

$$\alpha = \frac{x'w}{w'}, \beta = \frac{y'h}{h'},$$
$$G'(x', y') = \begin{pmatrix} 1 - \lfloor\lfloor\beta\rfloor\rfloor & \lfloor\lfloor\beta\rfloor\rfloor \end{pmatrix} \begin{pmatrix} G(\lfloor\alpha\rfloor, \lfloor\beta\rfloor) & G(\lfloor\alpha\rfloor + 1, \lfloor\beta\rfloor) \\ G(\lfloor\alpha\rfloor, \lfloor\beta\rfloor + 1) & G(\lfloor\alpha\rfloor + 1, \lfloor\beta\rfloor + 1) \end{pmatrix} \begin{pmatrix} 1 - \lfloor\lfloor\alpha\rfloor\rfloor \\ \lfloor\lfloor\alpha\rfloor\rfloor \end{pmatrix}. \tag{4.8}$$

例えば幅 1,024 [px]，高さ 512 [px] の新視点画像を生成したいとする．このとき MLP から 幅 512 [px]，高さ 256 [px] の新視点画像を生成し，新視点画像を縦横それぞれ 2 倍に拡大する．このレンダリング後拡大により MLP への負担を 1/4 倍まで軽減できる．

4.7 ステレオ映像の生成

自由視点画像を生成するために用いている画像データには，撮影したカメラのポーズ (原点の位置情報と回転情報) が含まれている．一つの画像には一つのポーズが付随しており，ポーズは原点を表す 3 次元縦ベクトル $\boldsymbol{O}$ と回転を表す 3 次正方行列 R で表される．ここでカメラの原点からその画像の画素 (i, j) へ延びるレイ $\boldsymbol{r}$ は式 (4.9) で表される．

$$\boldsymbol{r} = \boldsymbol{O} + t\left(j - \tfrac{W}{2},\ -\left(i - \tfrac{H}{2}\right),\ -1\right) R. \tag{4.9}$$

なお W, H は画像の幅 [px] と高さ [px] を表す．例えば 3 次元極座標 (r, θ, ϕ) の位置から原点に向く時，R は式 (4.10) で求められる R' の左上 3×3 を切り出した行列である．

$$R' = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ 0 & 0 & 0 & 1 \end{pmatrix} . \tag{4.10}$$

生成プログラムは $\boldsymbol{O}$ と R を与えればその視点からの画像を生成する．R が求められればステレオ画像を生成するためのカメラ配置も定まる．例えばある点 $\boldsymbol{A}$ から左に d だけ離れた所に一つ，右に d だけ離れた所に一つカメラを配置する場合を考える．この時左のカメラのポーズは原点 $A - dR^T(1,0,0)^T$, 回転は R で表される．一方で右のカメラのポーズは原点 $A + dR^T(1,0,0)^T$, 回転は R で表される．

FoV-NeRF を参考に注視点から離れた部分のレイサンプリング数を抑えて，高速化を図る．FoV-NeRF では画像中心を注視点とし，視野角 20 度以内，45 度以内とその他の範囲で区切っていた．画像中心から画像端までの最短距離を r [px] とおく．本研究では生成画像の中心から半径 $r\tan 20^\circ = 0.364r$ [px] の円の中に入る画素に高いレイサンプリング数を設定する．同様に生成画像の中心から半径 $r\tan 45^\circ = 0.5r$ [px] の円の中に入る画素に中程度のを，その他の画素には低いレイサンプリング数を設定する．この注視点から離れた部分のレイサンプリング数を抑える手法を foveation と呼ぶ.

Chapter 5

実験

5.1 実験環境

実験環境は以下のスペックを持つ PC である．

- CPU: Intel(R) Core(TM) i9-13900
- GPU: NVIDIA GeForce RTX 3070 Ti
- VRAM: 8,192 メビバイト
- メモリ容量: 62 ギビバイト
- OS: Ubuntu 22.04.3 LTS

5.2 実験内容と結果

本手法が有効であることを示すために実験を 3 つおこなった．1 つ目の実験ではセグメント配置が動作することを確認した．2 つ目の実験では 3 つの手法 (セグメンテーション，レンダリング後拡大，foveation) が生成速度と品質に与える影響を測定した．3 つ目はステレオ自由視点画像を生成できることを確認した．用いたデータは Tancik らの研究にて使われた Mission Bay データセットである．このデータセットにはカリフォルニア州ミッションベイで撮影された約 12,000 枚の写真とカメラ情報 (位置・向き・内部パラメータなど) が含まれている．

実験 1 では Mission Bay データセットに含まれていた撮影地点を全てプロットし，全点に沿うようなセグメント基軸を描画した．セグメント基軸は節 4.2 にて述べたアルゴリズムで作成した．制御点数は 50 点である．結果を図 5.1 と図 5.2 に示した．図 5.1 は Block-NeRF の撮影車が走ったルートを上空から見たものである．図 5.1 の座標系は図左下を原点とする相対座標であり，グラフ軸は撮影車が走った XY 平面を構成している．なお Z 軸は高さを表す．図 5.1 ではセグメント基軸が全体的にカメラ位置に沿っていることが確認できる．図 5.1 の一部を拡大した図 5.2 では曲がりうねった箇所にも沿っていることが確認できる．実

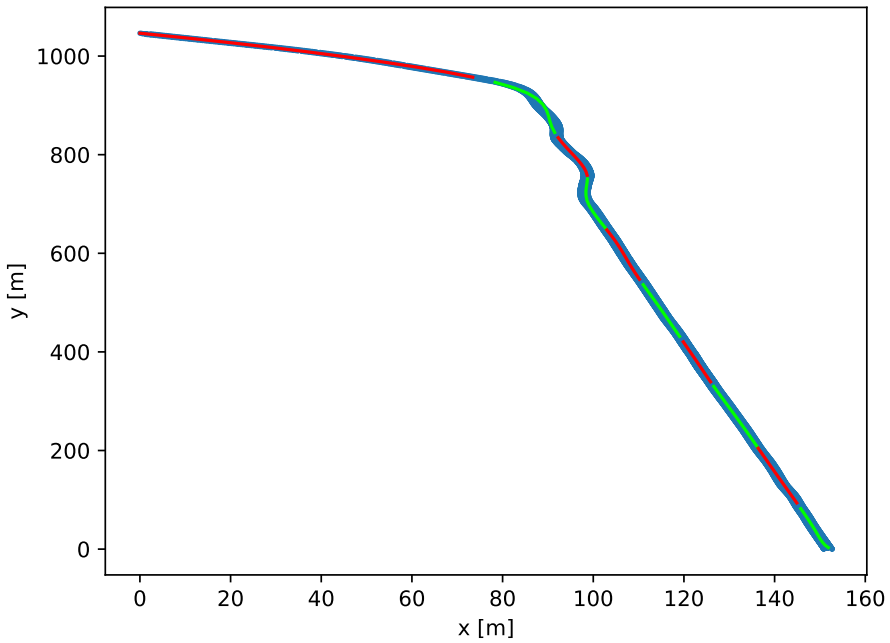

図 5.1: Block-NeRF 画像群が撮られた位置 (青点) とそれらに最小二乗法で近似した曲線 (赤線・緑線). セグメント間を示すため赤・緑のまだら模様で描画した

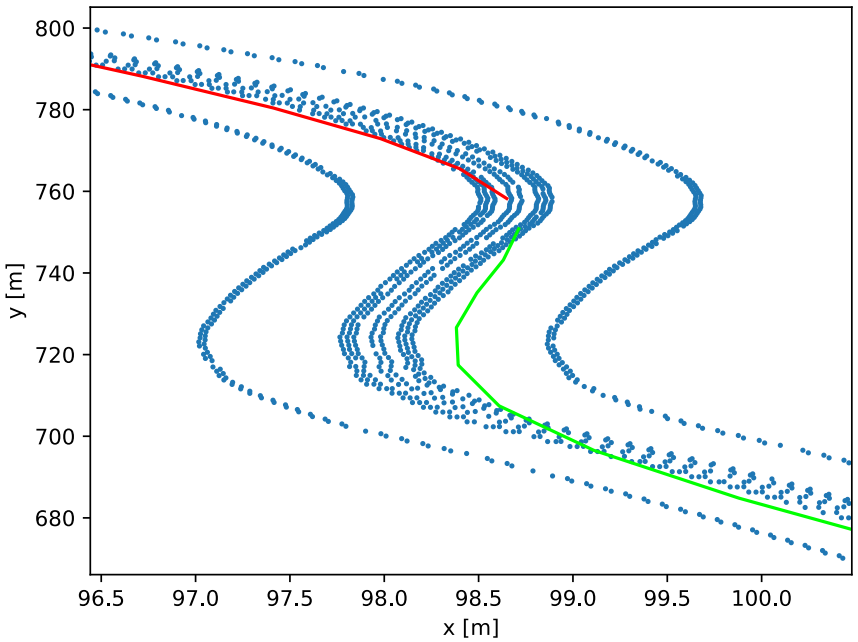

図 5.2: 図 5.1 の一部を拡大した図

表 5.1: 生成速度と結果画像品質

適応手法	None(図 5.4)	Se(図 5.5)	Se+Sc(図 5.6)	Se+Fov(図 5.7)	All(図 5.8)
fps	0.422	0.422	1.608	0.981	3.247
PSNR↑	18.92	21.07	21.04	20.94	20.91
SSIM↑	0.9296	0.9577	0.9573	0.9565	0.9563
LPIPS↓	0.131	0.126	0.127	0.127	0.127
PSNR 毎秒	7.984	8.891	33.832	20.542	67.895

験 2,3 で用いるデータはこのうち 0.4 から 0.5 まで延びるセグメント内 (図 5.1 における上から 3 番目の赤線周辺) にある実視点画像・位置・向きである．

実験 2 では 3 つの手法 (セグメンテーション，レンダリング後拡大，foveation) の効果を測定した．3 つの手法を組み合わせて，生成速度と結果画像品質がどのように変化するかを計測した (表 5.1)．簡単のため，表 5.1 ではそれぞれの手法を Se, Sc, Fov と略す．表 5.1 にある All はセグメンテーション，レンダリング後拡大，foveation の 3 つを全て用いた結果である．結果画像品質は PSNR, SSIM, LPIPS で正解画像との画像類似度を求めることで計測した [3]．また図 5.8 を生成した際の処理を細かく分けそれぞれに掛かった時間を計測した (表 5.2).

セグメンテーション無し実験では 1,125 枚分の実視点画像・位置・向きのデータを 1 つの MLP に与えた．学習イテレーション数は 10,000 である．対してセグメンテーション付き実験では 1,125 枚分のデータを 10 つの セグメントに分割し各セグメントに対応した MLP に与えた．すなわち 10 つの MLP に分散して学習させた．学習イテレーション数は 1 つの MLP につき 1,000 である．

Foveation 付き生成結果画像 (図 5.7) では生成画像の部分ごとにレイサンプリング数を変えた．画像中心から半径 226 [px] 内は 512，半径 311 [px] 内は 256，その他の箇所は 128 にレイサンプリング数を設定した．

表 5.2: 図 5.8 の生成に関する時間分布

処理内容	処理時間 [ms]	全体に対する割合
MLP でボリューム密度を求める	136	44.2 %
MLP で RGB 値を求める	108	35.1 %
ボリューム密度と RGB 値からレンダリング	32	10.4 %
その他	32	10.4 %
合計	308	100.0 %

実験 3 ではステレオ自由視点画像を生成した．図 5.9 はインタラクティブなステレオ自由視点画像生成システムの動作風景である．左右間隔 5 [m] のステレオカメラを配置した．両画像とも解像度は 512 [px] 四方に，生成手法は All に設定した．結果，両画像の生成には合計で 0.336 秒かかった．またセグメント間を移動する際に，セグメントごとでシーンの美麗さが変化することがあった．すなわちセグメント間で自由視点画像の品質が不安定であった．

実験 4 では学習イテレーション数を増やすことが品質にどう影響するかを測定した．セグメンテーション付き実験の学習イテレーション数を 1 つの MLP あたり 5,000 と 10,000 ま

図 5.3: 正解画像 (1096 [px] x 622 [px])

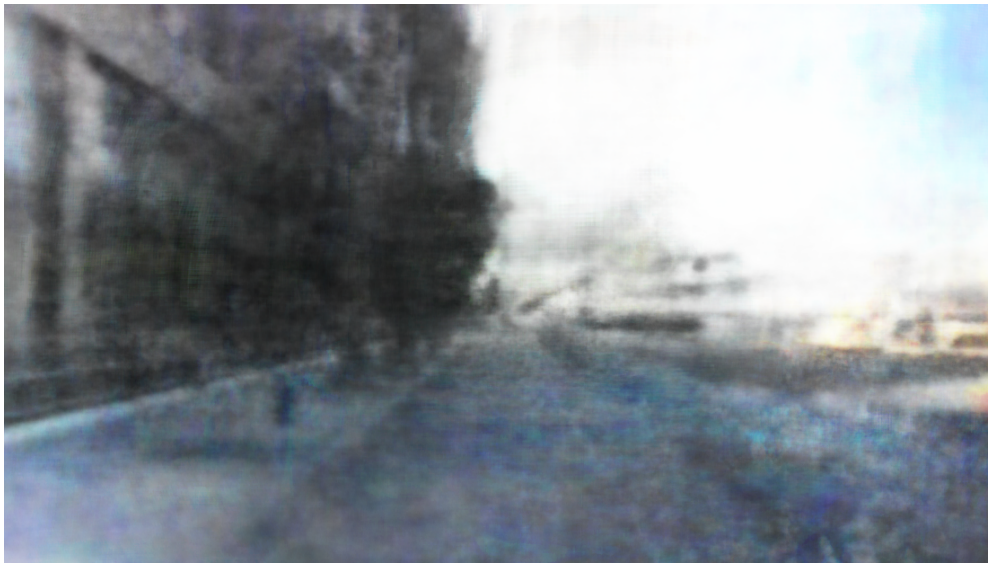

図 5.4: 生成結果画像 (セグメンテーションなし)

で増やし学習させた．レンダリング手法は Se + Sc である．その結果図 5.10 と図 5.11，表 5.3 を得た．

表 5.3: 学習イテレーション数と品質

イテレーション数	1,000(図 5.6)	5,000(図 5.10)	10,000(図 5.11)
PSNR↑	21.04	21.79	20.02
SSIM↑	0.9573	0.9642	0.9462
LPIPS↓	0.127	0.113	0.109

図 5.5: 生成結果画像 (セグメンテーションのみ)

図 5.6: 生成結果画像 (セグメンテーション+レンダリング後拡大)

5.3 考察

実験 2 ではシーンデコンポジションと高速化手法 (レンダリング後拡大と foveation) が生成速度と品質にどう影響するかを計測した．レンダリング後拡大は fps を 4 倍近く向上させた上に， PSNR の減少も 0.003 以内に収まった．速度が約 4 倍向上した原因は， MLP が処理するレイの個数が 1/4 倍になったためと考える．品質がさほど落ちなかった原因は，生成画像に元々高周波成分が入っていないぼやけた風景画であるためだと考える．

レンダリング後拡大が効果を発揮した理由は表 5.2 より示唆される．表 5.2 によると MLP が費やす時間は生成過程全体にかかる時間の 8 割弱に上る．このため MLP への負担を軽減するレンダリング後拡大が効果を発揮したと考える．加えて今後さらなる高速化のためには MLP 自体を高速化する，または MLP への依存を減らす工夫が必要であると考える．

図 5.7: 生成結果画像 (セグメンテーション + foveation)

図 5.8: 生成結果画像 (セグメンテーション+レンダリング後拡大 + foveation)

TinyCudaNN は NVIDIA GPU 向けに最適化されているため，MLP 自体を高速化するにはより良いハードウェアが必要になると考える．そのためキャッシングなどを用いて MLP への依存を減らす工夫を施す方が比較的汎用的な使い勝手ができる上に安価な環境で済むため良いと考える．キャッシングを用いて高速化した例としては， 800 [px] 四方の画像を 200 fps で生成する FastNeRF がある [14].

実験 4 では学習イテレーション数と品質の関係を評価した．表 5.3 からイテレーション数が 5,000 から 10,000 に増えた際に， PSNR と SSIM が悪化していることが分かる．一方で LPIPS は改善している．これが発生した原因は図左右端の明るさが図中央の明るさに比べて異なるためだと考える. 明るさが異なる原因は学習用データの露出にムラがあるためであると考える．

ステレオ自由視点画像生成システムを動作確認中に，セグメント間を移動すると品質が

図 5.9: ステレオ自由視点画像の生成結果

図 5.10: イテレーション数を 5,000 まで増やした生成結果画像 (元は図 5.6)

不安定になることがあった．考えられる理由の 1 つはセグメントごとに異なる実画像を学習しているためである．品質を安定させるためにはセグメント間に重なる地点を設ける，すなわちオーバーラップを設けることが有効だと考える．実際に Tancik らもオーバーラップはアーティファクトを防ぐためには重要であると述べている [7].

図 5.11: イテレーション数を 10,000 まで増やした生成結果画像 (元は図 5.6)

Chapter 6

まとめ

6.1 結論

本研究ではシーンデコンポジションをおこなうためにセグメント基軸による区分をおこなった．セグメント基軸による区分は自動的・確定的にシーンを分割した．加えて kd 木やレンダリング後拡大， TinyCudaNN などの高速化手法を適応した．その結果，1096 [px] x 622 [px] の大きさを持つ自由視点画像を 1 fps を上回る速度で生成した．加えて 512 [px] 四方のステレオ画像を 0.336 秒で生成した．

6.2 今後の課題

今後の課題は 3 つ挙げられる．それは高速化，学習用データをバランス良く分配すること，分散学習の実装の 3 つである．

1 つ目の課題は高速化である．これは生成速度の向上はもちろん，学習速度の向上も含まれる．生成速度の向上には表 5.2 より MLP がおこなう処理を高速することが不可欠だと考察する．学習速度を向上させるためには Python から C 言語に移植することや分散学習を実装することが考えられる．Garbin らが提案した FastNeRF はキャッシングを用いることで 200 fps 出力を達成したという [14]．Garbin らは RGB 値ネットワーク $\boldsymbol{c} : (\boldsymbol{x}, \boldsymbol{d}) \rightarrow \mathbb{R}^3$ のアーキテクチャを見直し，$\boldsymbol{c'} : (\boldsymbol{d}) \rightarrow \mathbb{R}^8$ まで入力値を減らした．そうすることでキャッシングに必要な空間的計算量を 5,400 テラバイトから 54 ギガバイトまで削減した．本手法に対してもネットワークアーキテクチャを見直し，キャッシングを実装することで高速化が見込めると考える．

2 つ目の課題は学習用データをバランス良く分配することである．学習用データをバランス良く分配するためには，セグメント基軸を引くアルゴリズムと適切な範囲設定をすることが重要である．例えば Zheng らの方法 [15] や Wang らの方法 [16] が役に立つと考える．また範囲設定については二分法を用いることで全セグメントが同数の画像を学習するよう分割することができると考える．

3 つ目の課題は分散学習の実装である．本研究ではチューブ型シーンデコンポジション

を実装したため，セグメントごとの学習が可能となった．そのため複数台のコンピュータを用いて各セグメントの機械学習を分散して進めることができれば学習速度の向上に繋がると考える．

謝辞

本論文は, 筆者が法政大学情報科学部ディジタルメディア学科在学中におこなった研究について執筆したものです．本研究を進めるにあたり，研究の進め方や論文のまとめ方をご指導いただいた小池崇文教授と院生の先輩方に心より感謝申し上げます．また, 日々の議論を通じて多くの意見を提供してくださった研究室の同期に感謝いたします. 最後に, 学生生活を支えてくれた家族に深く感謝いたします.

参考文献

[1] “Research & Development Group, Hitachi, Ltd. 用語集: 自由視点,” https://www.hitachi.co.jp/rd/glossary/jp_shi/ziyuushiten.html, (2024 年 01 月 24 日確認).

[2] A. Horé and D. Ziou, “Image Quality Metrics: PSNR vs. SSIM,” *2010 20th International Conference on Pattern Recognition*, pp. 2366–2369, 2010.

[3] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The Unreasonable Effectiveness of Deep Features as a Perceptual Metric,” *IEEE/CVF Conference CVPR*, pp. 586–595, 2018.

[4] J. T. Kajiya and B. P. Von Herzen, “RAY TRACING VOLUME DENCITIES,” *ACM SIGGRAPH Computer Graphics*, vol. 18, no. 3, pp. 165—-174, 1984. [Online]. Available: https://doi.org/10.1145/964965.808594

[5] B. Mildenhall, P. P. Srinivasan, and M. Tancik, “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis,” *Communications of the ACM*, pp. 99–106, 2021.

[6] K. Hornik, M. Stinchcombe, and H. White, “Multilayer Feedforward Networks are Universal Approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[7] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar, “Block-NeRF: Scalable Large Scene Neural View Synthesis,” *Proceedings of the IEEE/CVF CCVPR*, pp. 8248–8258, 2022.

[8] N. Deng, Z. He, J. Ye, B. Duinkharjav, P. Chakravarthula, X. Yang, and Q. Sun, “FoV-NeRF: Foveated Neural Radiance Fields for Virtual Reality,” *IEEE Transactions on Visualization and Computer Graphics*, pp. 3854–3864, 2022.

[9] Z. Mi and D. Xu, “Switch-NeRF: Learning Scene Decomposition with Mixture of Experts for Large-scale Neural Radiance Fields,” *Poster on International Conference on Learning Representations (ICLR)*, 2023. [Online]. Available: https://openreview.net/forum?id=PQ2zoIZqvm

[10] T. Müller, F. Rousselle, Novák, and A. Keller, "Real-time Neural Radiance Caching for Path Tracing," *ACM Transactions on Graphics*, pp. 1–16, 2021.

[11] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *Poster on International Conference on Learning Representations (ICLR)*, 2015.

[12] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant Neural Graphics Primitives with a Multiresolution Hash Encoding," *ACM Transactions on Graphics*, pp. 102:1–102:15, 2022. [Online]. Available: https://doi.org/10.1145/3528223.3530127

[13] L. Jing and L. Sun, "Fitting B-Spline Curves by Least Squares Support Vector Machines," *2005 International Conference on Neural Networks and Brain*, pp. 905–909, 2005.

[14] S. J. Garbin, M. Kowalski, M. Johnson, J. Shotton, and J. Valentin, "FastNeRF: High-Fidelity Neural Rendering at 200FPS," *arXiv preprint arXiv:2103.10380*, 2021.

[15] W. Zheng, P. Bo, Y. Liu, and W. Wang, "Fast B-Spline Curve Fitting by L-BFGS," *Computer Aided Geometric Design*, pp. 448–462, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167839612000301

[16] W. Wang, H. Pottmann, and Y. Liu, "Fitting B-Spline Curves to Point Clouds by Curvature-Based Squared Distance Minimization," *ACM Trans. Graph.*, p. 214–238, apr 2006. [Online]. Available: https://doi.org/10.1145/1138450.1138453