修士論文
令和 6 年度

ライトフィールドディスプレイのための多視点画像生成に関する研究
— Research on multi-view image generation for a light field display—

京都産業大学大学院
先端情報学研究科
博士前期課程 2 年
386049
大野　陽基

要旨

本論文では, Japan Display 社が提供しているライトフィールドディスプレイに現実世界の物体を投影するための多視点画像生成に関する研究について説明する. ライトフィールドディスプレイは表示される画像が立体的に見えることや, 見る角度によって物体の見える方向が変わることが特徴である. これまで, ライトフィールドディスプレイに表示する画像は, 事前に 3 次元 CG を用いて 3D モデリングを行い, その 3D モデルを Unity の中で 69 視点分の画像として撮影し, ディスプレイに投影する画像を生成していた. これに対して, 2 台のカメラで撮影した実写画像から, 2 視点間の 67 視点分の画像を生成することで表示用の画像を生成する手法を提案する.

本研究では, 中間画像を生成する基本的な手法としてステレオマッチングを用いた. 通常のステレオマッチングは, 同じ性能の 2 台のカメラを水平方向に並べて設置し, エピポーラ線が水平走査線に一致する状態で行うのだが, 2 台のカメラを被写体に向けた状態で撮影した画像はカメラの光軸が平行ではないためエピポーラ線が水平走査線に一致しなくなる. そこで, ステレオ画像の平行化を行うことによって画像のエピポーラ線を水平走査線に揃え, ステレオマッチングを適用できるようにした. また, もう一つの手法として, 複数画像から自由視点画像を生成する 3D Gaussian Splatting から発展した Splatt 3R を用いて二つの画像からその中間視点を得ることができた. それを用いることで 67 視点分の画像を生成することを試みた.

結果として, 2 視点から輻輳角度をつけて撮影した画像を用いて, 平行化処理を行なったのちステレオマッチングを行うことで 67 枚の画像を生成することができた. また, Unity 上で行なっていた画像を合成する処理を Python で書きかえ, 表示画像の生成を行った. 生成した画像をライトフィールドディスプレイに投影した結果, 綺麗に投影することができたが, 基になる 2 視点の画像の輻輳角度を大きくとることが難しいため, 立体感が薄れてしまう結果となった. Splatt3R の利用についてはその動作確認を行った.

目次

第 1 章 序論

1.1 背景

医療や教育，エンターテインメントなどの分野で立体視を可能にするディスプレイ技術が利用されている．実用化されている立体ディスプレイの多くは，3 次元の世界を両目で観察した時に生じる両眼立体視に基づくものである．この方式のものにも，いくつかのバリエーションがあり，大別すると，3D メガネやヘッドマウントディスプレイ（HMD）などの器具を頭部に装着するタイプのものと，それらを必要とせず裸眼で立体視が可能なものに分けることができる．

装着型の立体視は，器具を装着する煩わしさが指摘されており，特に長時間の使用には適していない[1].

裸眼両眼立体視の原理は，画像面の直上にシート状の特殊なレンズを配置することで，空間中の特定の位置で左目画像と右目画像が分離して見えるようにすることである．例えば，左目画像と右目画像を縦にごく細い短冊状に切り分け，細く分割した左目画像と右目画像を交互に横に並べた画像を準備する．それぞれの短冊状画像の上に短冊幅のレンチキュラレンズを配置したシート状レンチキュラレンズを貼り付ける．この時，左目画像と右目画像が，レンチキュラレンズの働きによって，空間中で，両目距離で分離した位置で焦点を結ぶように設計する．すると，左目画像だけが左目に入射し，右目画像だけが右目に入射するようにできるため，両眼立体視が可能になる．これを「固定視点の両眼立体視」と呼ぶことにする．固定視点の両眼立体視で立体視できる位置は，原理的には空間中の一点だけである．その位置から前後左右のある程度の範囲では立体視可能であるが，頭部を大きく動かすと，画像が二重に見えるなどの現象が発生する．

ライトフィールドディスプレイは，固定視点の両眼立体視技術を多数の視点からでも両眼立体視が可能な技術に発展させてものである．図 1.1 は水平方向の 7 視点で立体視可能なライトフィールドディスプレイの説明図である．この場合，ディスプレイ面には 7 枚の画像を短冊状に切り分けて配置する．ディスプレイ面の上に貼り付けたレンチキュラレンズの働きによって 7 枚の画像は図のような 8 箇所で焦点を結ぶ．隣り合う画像を左目と右目で観察した時に立体視できるように，適切な視差を画像に与えておく．さらに視点位置を移動させると，その視点位置に応じて見えているシーンの方向が適切に変化するように画像を準備する．なお，多数の左目・右目画像を一枚に合成した画像を表示する装置として通常の液晶ディスプレイを用いることで，立体動画像の表示が可能になる．

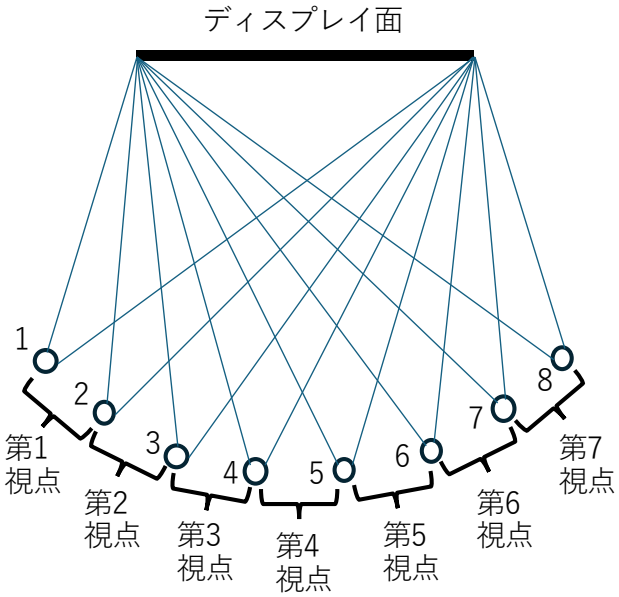

図 1.1 水平方向の 7 視点で立体視可能なライトフィールドディスプレイ

ライトフィールドディスプレイは，3D メガネなどの装着デバイスを用いることなく，自然な視覚体験を提供する立体視デバイスである．ライトフィールドディスプレイは，複数の視点からの立体画像を提供し，視差や奥行き感をリアルに再現することで，観察者が自然に視点を変えることが可能である[2].

ライトフィールドディスプレイは，主にコンピュータで生成された 3D モデルの表示に用いられており，現実世界の物体を投影することには制約があった．その理由として，実シーンの多視点画像を得るためには，多数のカメラでの撮影が必要となり，撮影の煩雑さが課題であった[3]．一方で，現実の物体を少ないカメラ視点で撮影し，そのデータから多視点画像を生成する技術が発展している．これにより，リアルタイムに近い形で現実の物体をライトフィールドディスプレイに投影し，遠隔地にいるユーザーに自然な立体体験を提供することが可能になると期待される．

本研究では，現実の物体を 2 台のカメラで撮影し，その 2 枚の画像データを基にライトフィールドディスプレイ向けの多視点画像を生成する方法を検討する．これにより，現実世界の物体を容易に立体化し，遠隔地からでも臨場感のある立体視体験を提供することを目指す．この技術は，医療や教育，リモートワークなどの分野における遠隔体験の質を向上させ，特に直感的な操作が求められる場面での有用性が期待される．

1.2 目的

本研究の目的は，ライトフィールドディスプレイ上に現実世界の物体を自然な立体視で表現するため，2 台のカメラで撮影した画像データから多視点画像を生成し，リアルタイ

ムに近い形で表示する手法を開発することである．

従来のライトフィールドディスプレイでは 3D モデルを用いた視覚表現が主流であったが，本手法により現実世界の物体を多視点で立体表示可能にする．これにより，遠隔地にいるユーザがリアルな立体映像を通して医療や教育，リモートワークなどの場面で高度な臨場感を得られることを目的としている．

1.3 論文構成

本論文では，第 2 章で，関連研究として，本研究に用いるライトフィールドディスプレイ，ステレオマッチングやオプティカルフロー，複数の視点から三次元シーンの復元を行う 3D Gaussian Splatting の説明を行う．第 3 章で実際に 2 視点の画像を用いてライトフィールドディスプレイに投影する画像の作成を行う．第 4 章ではそれらを用いてライトフィールドディスプレイに投影し，評価を行う． 第 5 章で結論を述べる。

第 2 章 Light Field Display と多視点画像の生成方法

２章では関連研究と本研究で用いる技術について説明する．まず，本研究に用いる Japan Display 社(JDI)のライトフィールドディスプレイの仕組みについて説明する．次に複数の画像から，それらの画像の中間視点からの画像を生成する方法について概要を説明する．最後に最新の任意視点画像生成技術として 3D Gaussian Splatting の紹介を行う．

2.1 ライトフィールドディスプレイの仕組み

本研究で用いるライトフィールドディスプレイは JDI が開発，販売しているものを用いる(図 2.1)．このディスプレイの画面サイズは 5. 5 型であり，画面の横幅 6.85cm，縦幅 12.15cm，画素数は 1440×2560 となっており，HDMI で画像の入力を行う．このディスプレイの特徴は裸眼立体視が可能で，かつ，図 2.2 に示すようにディスプレイを見る角度によって対象物体の見え方が変わることである．

このディスプレイに対して，JDI が SDK を提供しており，それを基に Unity 環境で像を投影することが可能である．Unity 環境での表示プロセスは以下のようになる．

まず，表示対象となる物体の 3D モデルを作成する．次に 69 台のカメラを直線上に設置し，69 視点から画像データを作成する．最後にそれら 69 個の画像をディスプレイに適した形に編集し，表示を行う．

3D モデルは，Blender や 3D スキャナなどで作成する．3D モデルを Unity に読み込み，69 視点分の仮想カメラで撮影を行う．カメラは対象の物体に対して垂直に一直線に並べる(図 2.3)．配置されているカメラは全て同じ方向を向いており，Lens Shift の機能を用いて，各カメラの光軸に角度がつけられている．Lens Shift というのは，カメラレンズの位置を水平に移動させて撮影する機能のことである．この機能により，カメラの方向を被写体に向けることなく，被写体像を画像の中央で捉えることができる．Lens Shift の機能として図 2. 4 の画像のような機能がある．上段が対象のオブジェクトに向けてカメラの光軸を傾けた場合である．この場合赤い線で表される水平線が右に向かって収束しているのが分かる．それに対して下段は Lens Shift を行った場合の様子である．この場合を見るとレンズを水平方向に移動してオブジェクトをフレームに収めることによって赤い線は収束せず平行に保たれる[4]．最も左のカメラの Unity での設定は図 2.5 の通りとなっており，Position で位置調整を行い，Lens Shift を用いてカメラ内のレンズを調整することでカメラの向きを仮想的に調整する．

最後に各カメラで撮影された 69 個の画像を用いてディスプレイに適した画像の作成を行う．これは Unity 上で，JDI が提供した compute などのスクリプトを用いて行われている．JDI のライトフィールドディスプレイに送る最終画像はハードウェアに依存しており，69 個の 512×1024 画素のカメラ画像から，　1440×2880 画素の RGB の値を生成し，ディスプレ

イに合うように各ピクセルに対応するカメラ ID と座標から画素値を得て，最終画像を生成する．実際に表示される両端の画像を図 2.6 の(a)と(b)に示す．

図 2.1 本研究で用いるライトフィールドディスプレイの外観

図 2.2 視点によるライトフィールドディスプレイの見え方の違いの例

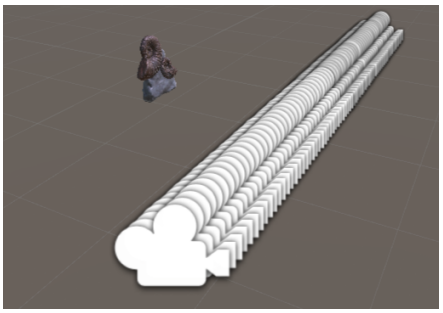

図 2.3 Unity でのオブジェクトとカメラ配置の様子

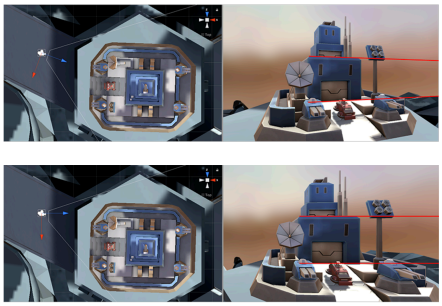

図 2.4 Lens Shift の様子

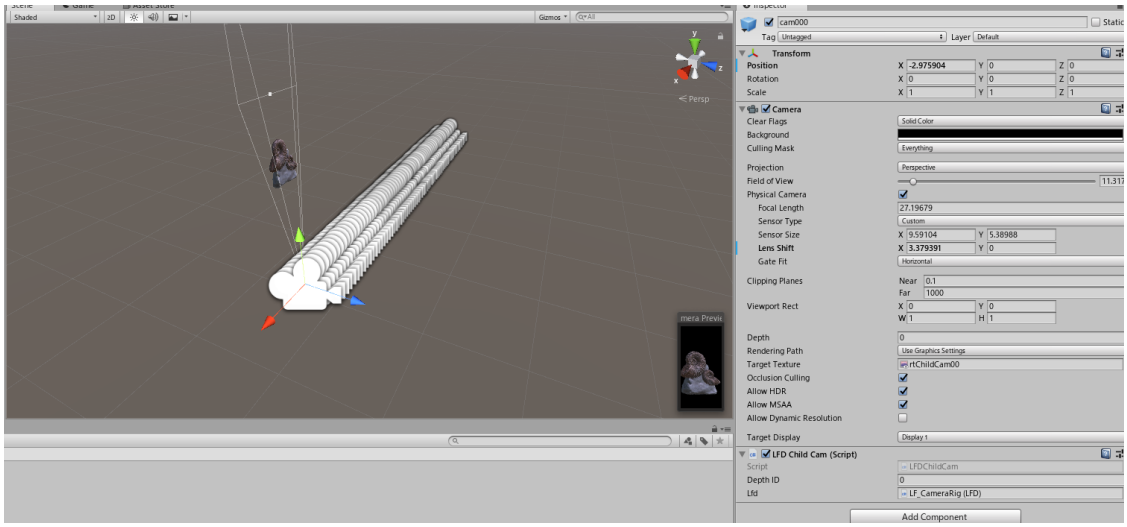

図 2.5　一番左のカメラの設定値の様子

図 2.6 (a)左端からの画像

図 2.6 (b)右端からの画像

2.2 2 画像の中間画像を生成する方法の概要

異なる視点で撮影した 2 画像からその中間視点の画像を生成する方法には，さまざまなものがある[5][6][7]．本論文では，2 画像からそれらの視差マップを推定し，視差マップをもとにして，中間画像を推定する方法を用いる．2 画像を，光軸がお互いに平行で画像面が共面になっているカメラで撮影した場合，それらの視差を計算する問題は平行ステレオでのステレオマッチングに帰着される．平行ステレオは図 2.7 のように向きが揃えられた 2 台の同じカメラを水平方向に並べて撮影するステレオカメラである．ステレオマッチングにおける対応点探索はエピポーラ線とよぶ直線上に拘束される．平行ステレオの場合，エピポーラ線が画像の水平走査線に一致する．視点間の距離はベースラインと呼ぶ．視差とは二枚の画像間で対応する画素の位置の差を表示する．

ステレオマッチングを用いた中間画像生成の例として図 2.8 の(a)(b)を用いて二つの画像の中間視点を求める場合を説明する．ここで用いる画像は筑波大学によって用意された画像であり，ステレオビジョンシステムを評価するための画像としてよく利用される．

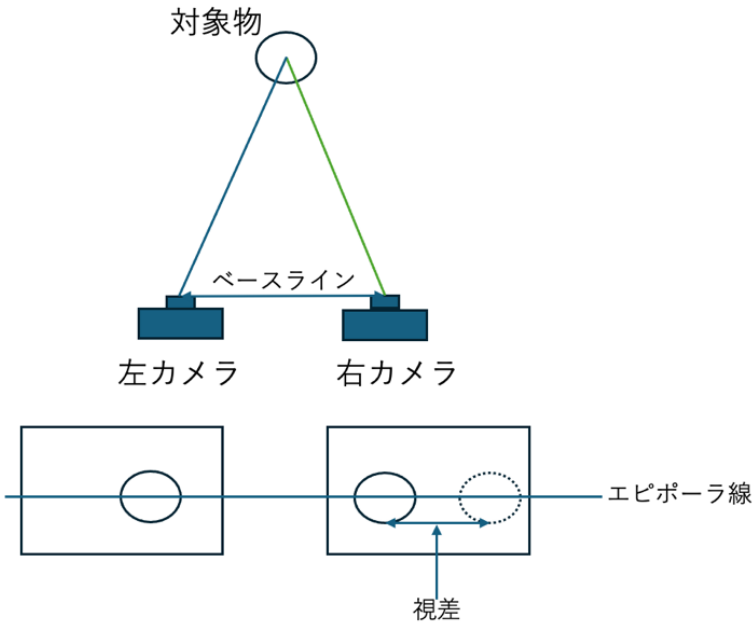

図 2.7 平行に設置されたカメラでのステレオマッチングの様子

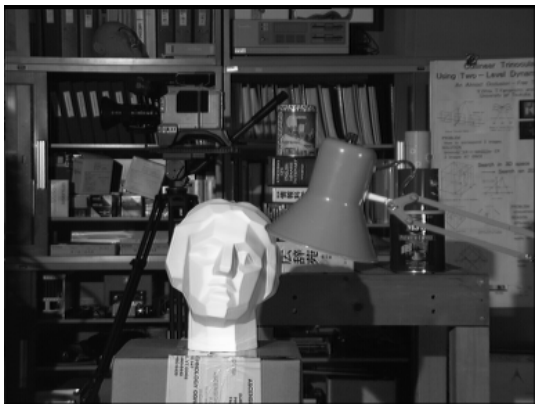

図 2.8 (a)ステレオ画像(左側)

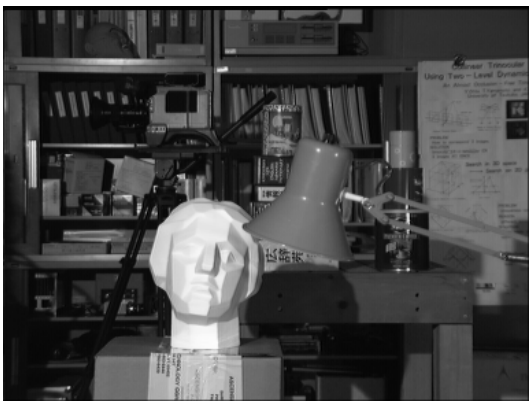

図 2.8 (b)ステレオ画像(右側)

ステレオマッチングを行う場合，OpenCV が提供している StereoSGBM クラスを用いてたステレオマッチングを利用することが多い[8]． SGBM（Semi-Global Block Matching）は，ステレオマッチング手法の一つであり，視差マップを生成する際にローカル手法とグローバル手法の特長を組み合わせたアプローチを採用している．ローカルなブロックマッチングでは計算コストが低いものの，精度が限定的である．一方，グローバル手法はエネル

ギー最小化を通じて精度を高めるが，計算コストが高い．SGBM は，この二者の中間的な手法として，複数方向（通常は 5 方向または 8 方向）の視差計算を統合することで，計算効率と精度のバランスを取る．これにより，不連続部分や細部の表現が改善され，3D 復元や物体検出など高精度が求められる応用分野で用いられている．SGBM のステレオマッチングではブロック単位で類似度を計算し対応点検出を行う．ブロックサイズを 1 に選択することでピクセルごとの類似度の計算となる．StereoSGBM に適切に行うために以下の表 1 にまとめた入力パラメータがある．

表 1　StereoSGBM における入力パラメータ

パラメータ	**説明**
minDisparity	ステレオマッチングを用いるにあたってとり得る最小の視差値．通常は 0
numDisparities	とり得る最大の視差値．常に 0 よりも大きくなる必要がある．2024 年 12 月の段階ではこの値は 16 の倍数になる必要がある．（メモリアクセスの関係上）
blocksize	マッチングされるブロックのサイズ．1 以上の奇数にする必要があり，多くは 3~11 が適用される．1 を指定するとピクセルごとのマッチングとなる．
P1	視差の滑らかさを制御するためのペナルティで，隣接するピクセル間が+1 または-1 した場合のペナルティ
P2	隣接するピクセル間の視差が 1 以上変化した場合のペナルティ (P2 > P1)
disp12MaxDiff	ステレオマッチングでは通常左右両方の視差マップを形成するため，それらの左右の視差マップにおいての差分を計算し，各ピクセルについてその差が指定された閾値以下の場合のみ視差を有効とする．値を小さくする(1, 2)と精度が高くなるが対応点が見つかりにくい場所などは欠損してしまう．(0, -1 にすると無効)
preFfilterCap	視差の計算を行う前に画像の輝度を事前にクリップするための上限値を決定する．この値を決定することで画像の輝度が [-preFilterCap, +preFilterCap]に制限される．値を 100 などの大きな値にすると輝度情報が保持されるため詳細な視差マップを得やすいが，ノイズが増えやすい恐れがある．
uniquenessRatio	視差マップを生成する際，視差値の候補を複数作成し，そこからコスト(マッチング誤差)が最小のものを選択するが，他の視差値の候補と比べてコストの差が小さい場合，誤った視差値が選ばれる恐れがある．そのため，最小のコストの視差と 2 番目のコストの差

	が最小に対して設定した割合(%)以上である場合のみにその視差が採用される．値を 1, 5 などの小さい値にした場合，詳細な視差マップが得られるが，誤マッチングが起こる可能性が生ずる．
speckleWindowSize	視差マップ上の斑点ノイズを除去するためのウィンドウサイズを設定するパラメータである．設定した数値以下の領域の独立した視差をノイズとして除去する．
specleRange	視差マップ上の隣接するピクセル間で視差値の差が設定された範囲以下に収まっていない場合ノイズとして除去される．
mode	視差マップを計算する際のアルゴリズムモードの設定

ここで mode は以下の表 2 に記載する 4 つのモードから選択する．モード名は cv2.StereoSGBM_MODE_までは共通の記載になる．

表 2　StereoSGBM におけるモードの比較

モード名	説明
SGBM	標準なステレオマッチング．速度と精度のバランスが良いことが特徴となっている．
HH	8 方向のスキャンライン(上下左右斜め)を考慮して正確な結果を生成する．高精度になるが，計算コストが増加される．
HH4	HH を基に GPU を活用して高速に計算を行うことができる
SGBM_3WAY	3 方向のスキャンラインに制限することで処理時間の高速化を行う．

これらの入力パラメータを適切に設定してステレオマッチングをおこなう．実際には，stereoSGBM のパラメータはほとんどデフォルトのままで，num_disparities については 16×5 を採用した．

図 2.8 の画像は 384×288 画素である．図 2.9 に計算した視差マップを示す．本来，視差マップはグレースケールで出力されるのだが，マップ上の深度が見やすいように，深度に応じてカラー化した画像を示した．この図において，マッチングする領域を指定する block_size として，図 2. 9(a)は 11, (b)は 5, (c)は 3 に設定した．その結果として画像内でのライトの脚の部分などを見るとより明確に表現できているのは block_size=3 となるのでこちらを用いた．図 2.10 は図 2.9(c)に視差値を重ねて表示した画像である．この図中の赤い部分は視差が一番大きな部分で最大値は 14.0 となっている．また，最小値は濃い青色で表示されており，-1.0 と表示されている．これはマッチングが行えなかったことを示す．図 2.10 で-1. 0 と表示されている部分が大きくなっている理由は num_disparities の値が大きいことが関係している．図 2.10 の視差の最大値が 14.0 になっていることから num_disparities を 16×5 から 16 に変更した．最終的な視差マップは図 2.11 となり，マッチングできない左端の領域が減少した．この視差マップを用いて，右視点画像から視差値

の半分を水平方向にずらせた画像を中間画像とした．また、視差マップを作成する際に，同じテクスチャが続くところやテクスチャが存在しないところ，片方の画像にのみ写っている部分では視差が計算できず，ステレオマッチングに欠損が生じてしまう．欠損が生じた視差マップで作成した中間画像には欠損部分が生じる．その場合，左隣のピクセルをコピーして補うこととした．これは右視点画像を基にしてピクセルの移動を行っているからである．その方法で作成した中間画像を図 2.12 に示す．図 2.12 の中間画像を，その上下の画像と見比べると，石膏像の画像が視差マップの値の半分だけ水平方向にずれていることがわかる．

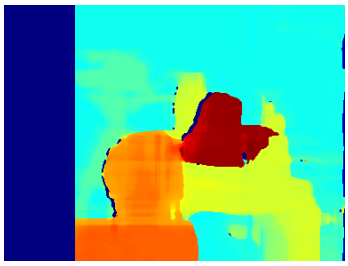

図 2. 9(a) block_size=11

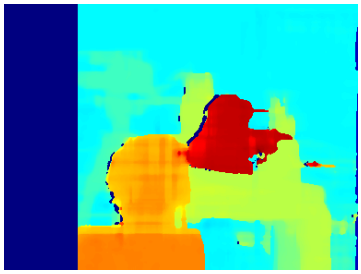

図 2. 9(b) block_size=5

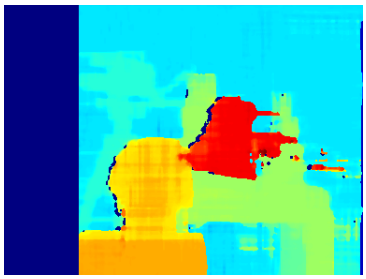

図 2. 9(c) block_size=3

図 2. 10 視差の数値を可視化した様子

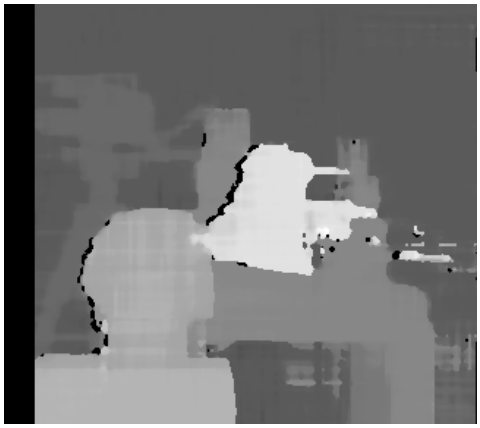

図 2. 11 num_disparities を 16 にした視差マップ

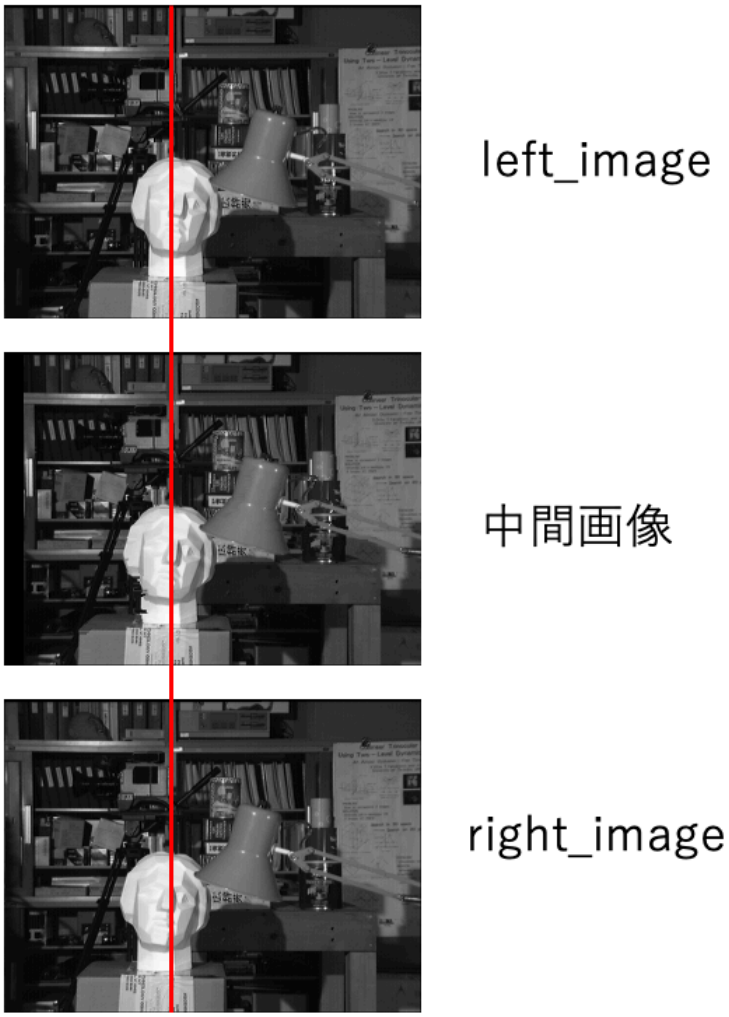

図 2.12 中間画像とその両端画像

2.3 最新の任意視点画像生成手法：3D Gaussian Splatting

3D Gaussian Splatting(以下 3DGS)[9]は 3 次元空間の立体情報を 3 次元ガウス分布の集合体として表現し，それをレンダリングすることで空間の見え方を生成する技術である．3DGS の特徴はガウス分布という連続的な表現を用いることでボクセルや三角形メッシュに比べて滑らかな表現が可能であり，計算負荷を軽減できる点である．ガウシアンは正規分布のことであり，それらを重ね合わせることで 3 次元空間をモデル化する．以下の図 2.13 の画像を例に 3DGS で 3 次元空間をモデル化する流れを説明する．まずは表示させたい物体を点群として表示する(図 2.14)．点群として表示させるためには，表示させたい物体を複数の角度から撮影した 2D 画像を用意し，Structure from Motion(以下 SfM)を使用する．今回は SfM のライブラリとして 3D Gaussian Splatting の論文でも使用されていた，

COLMAP を使用する．これを用いて点群の生成とカメラポーズの推定を行う．

COLMAP が出力する点群位置を初期値として 3D ガウス分布を生成する．ガウス分布の成分として色・透明度などの見え方に関するパラメータも設定する．その後，入力画像と 3D ガウス分布を元にしたレンダリングを比較することで，ガウス分布の伸縮度や方向，色，透明度などのパラメータを最適化する．また，適宜，ガウス分布の分割・統合を行う．これらの処理を繰り返すことで，入力画像群にマッチした 3DGS に収束させる．

図 2.13 の画像をはじめとしてこの空間を撮影した多数の画像を用いて，3DGS で 3 次元空間を再構成した．再構成した 3 次元空間を，論文内のプロジェクトで用意されているビューワを用いて表示させた例を図 2.15 に示す。図 2. 13 とほとんど同じ画像が生成できていることがわかる．

図 2.13 例として用いる画像

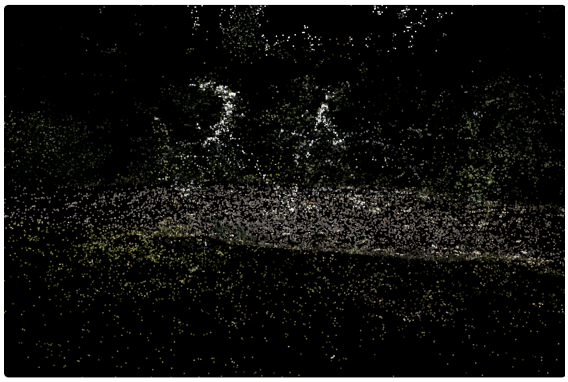

図 2.14 COLMAP を用いて点群表示させたときの様子.

図 2.15 3D Gaussian Splatting で表示させた様子

第 3 章 LFD に投影する中間画像の生成

本章では実物体をライトフィールドディスプレイに投影するため，2 視点の画像を両端画像とし，それらの画像からステレオマッチングで中間画像の生成を行う原理を説明する．また，3D Gaussian Splatting をもとにした Splatt3R という技術による 2 画像からの中間画像生成についても説明する．

3.1 輻輳画像を用いたステレオマッチング

2.2 章で説明した SGBM によるステレオマッチングで用いる画像は，エピポーラ線が画像の水平走査線に一致するステレオ画像である必要がある．しかし，ライトフィールドディスプレイに投影する画像は，図 3.1 のように 2 画像間のエピポーラ線が輻輳する輻輳ステレオ画像を用いることが多い．ここで図上の赤線部で示されている線分 C, C'は 2 台のカメラの中心を結んだ基線，その基線がそれぞれの画像平面を通る点 e, e'をエピポールという．また，対象の注目点 P と 2 台のカメラの中心 C および C'が形成する面をエピポーラ面と呼ぶ．そしてエピポーラ面が画像平面と交差する断面が図中の黄緑色で示されているエピポーラ線である．この節では二枚の画像のエピポーラ線を水平走査線に合わせる平行化を行う方法を説明する．平行化を行う手段として本研究では OpenCV が提供している stereoRectify クラスを使用する． stereoRectify クラスの入力パラメータを以下の表 3 を用いて解説する．なお，エピポーラ線が水平走査線に一致しない状態のままステレオマッチングを行うことも可能であるが，SGBM などの高度なステレオマッチングアルゴリズムを利用するためには，ステレオ画像の平行化は有効である．

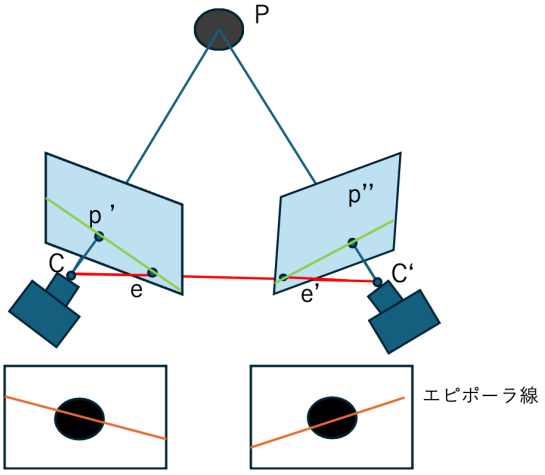

図 3.1 本研究でのカメラ配置の例

表 3 stereoRectify における入力パラメータ

パラメータ	説明
cameraMatrix1 cameraMatrix2	左カメラおよび右カメラの内部パラメータ行列 K1, K2
R_rel	R_rel は二つの画像の相対的な回転ベクトルを表す. それぞれ二つの画像の回転行列 R1, R2 は, ワールド座標系における 3×3 行列. これはカメラ間での方向を記述し, 左右のカメラ間の視差を補正するために必要である.
baseline	baseline は 1×3 の行列でカメラ間の位置関係を記述していて相対的なカメラの移動量を示す.
imageSize	入力画像の幅と高さを指定するパラメータである.
distCoeffs1 distCoeffs2	これは歪み係数であり, 左右カメラのレンズの歪みを補正するために使用される.
flags	Flags は, 平行化の方法を指定するために使用するオプションのパラメータである. 主に cv2. CALIB_ZERO_DISPARITY を用いて視差をゼロにするか, 0 に設定して元のカメラの設定を用いる.

ここで, 内部パラメータ行列 K はカメラの焦点距離 fx, fy と画像平面上の主点座標 cx, cy を用いて以下の行列式で記述する.

$$K = \begin{bmatrix} fx & 0 & c_x \\ 0 & fy & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

また, 二つの相対的な回転ベクトルを指す R_rel は以下の式で計算を行う.

$$R_{rel} = R_2 \cdot R_1^T \tag{2}$$

baseline は回転を考慮し, 以下の式で計算する.

$$baseline = t2 - R_{rel} \cdot t1 \tag{3}$$

これらの入力パラメータを元に関数の計算を行う.

これらのパラメータを用いて動作を確認した平行化の例を以下に示す. 図 3.2 のように仮想の空間上に 3 点(0, 500, 7000), (-400, -300, 5000), (400, -300, 7000)と, 青色で表される基準のカメラと, 赤色で表される, x 軸のマイナス方向に 200 移動させた上で y 軸を軸にして 10.0 度回転させたカメラを用意した.

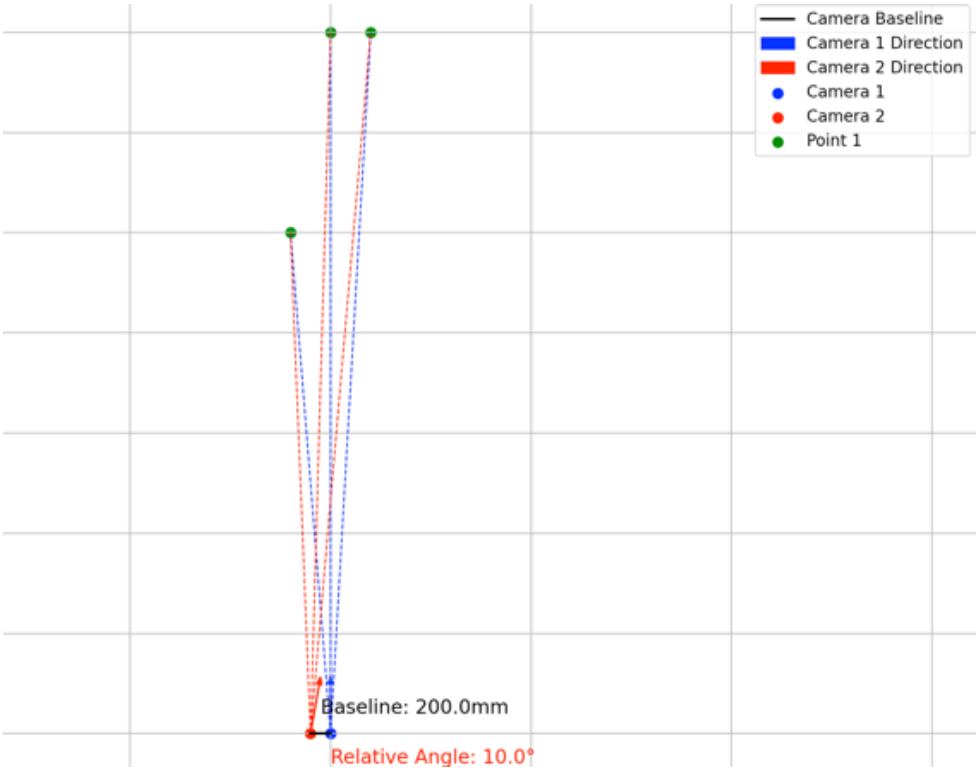

図 3.2 平行化の説明に用いる 2 台のカメラと観察点の様子

図 3.3(a)は，それぞれのカメラから見える観察点の様子を重ね合わせたものである．これを見ると 3 点の対応する点同士を結んだ線分に角度がついている．そこから平行化を行った結果が図 3.3(b)である．これを見ると対応する点同士は水平方向だけに移動している．

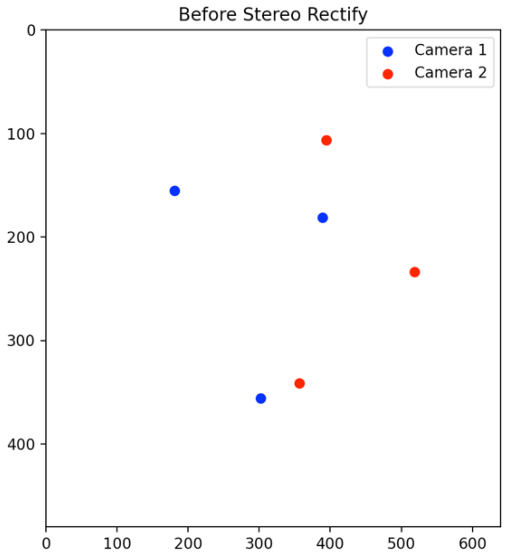

図 3.3(a) 平行化前の 3 点の見え方

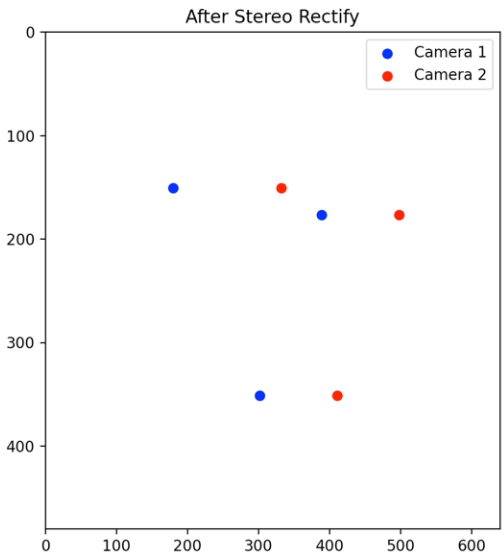

図 3.3(b) 平行化後の 3 点の見え方

3.2 Splatt3R を用いた 2 画像からの多視点画像生成

2.3 節で説明した 3D Gaussian Splatting から発展した技術として Splatt3R[10]が存在する. Spatt3R は異なる視点から撮影した未校正の画像ペアを入力として三次元構造の推定を行い, 新たな視点からの画像を生成する技術である. この手法は事前のカメラのキャリブレーションが不要であり, ゼロショットで高品質な任意視点画像を生成できる点が特徴である. 画像ペア間の対応点を抽出し, それをもとに 3D ガウス分布を構築して任意視点からの画像を生成するアプローチが行われている.

Splatt3R は github から簡単に試すことができる. Splatt3R で背景の広い画像を入力に用いると, 表示される自由視点画像で背景の黒い部分が強調されることがあるため, 対象物体が大きく撮影されており, 背景が少ないものにする必要がある. 図 3.4(a), (b)は入力した２画像の例であり, これに対する Splatt3R の出力の中から, 入力画像に該当する箇所をスクリーンショットしたものを図 3.5(a), (b)に示す. これは Splatt3R を用いて作成した自由視点画像で, 2 つの入力画像に最も近い 2 つの出力画像である。

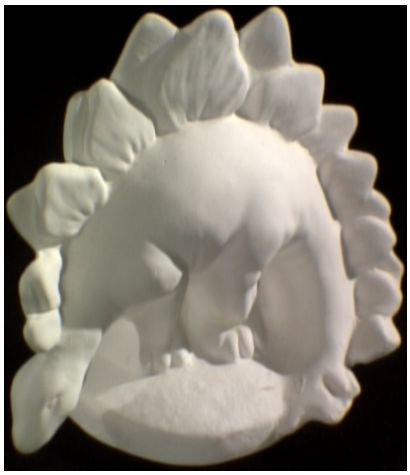

図 3.4(a) 左側からの撮影画像

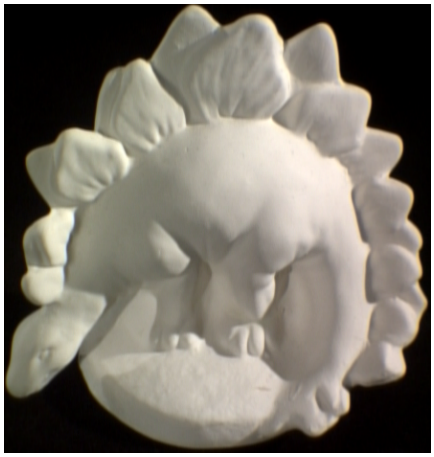

図 3.4(b) 右側からの撮影画像

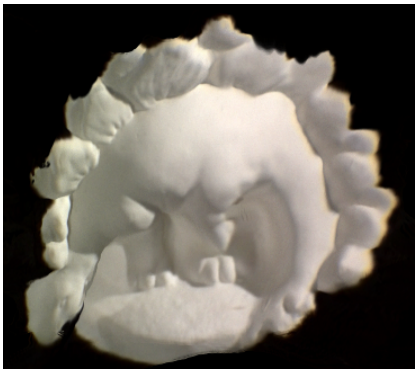

図 3.5(a) Splatt3R 上の左側にあたる画像

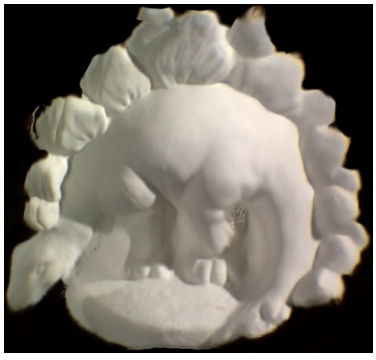

図 3.5(b) Splatt3R 上の右側にあたる画像

第 4 章 中間画像の生成とディスプレイへの投影

本章では輻輳画像を用いて生成した中間画像と，そのライトフィールドディスプレイへの投影について記述する．

4.1 恐竜画像を用いた中間画像の生成

カメラの内部パラメータ，カメラ間の回転ベクトルと並進ベクトルが既知の 2 画像を用いて，画像ペアの平行化を行ったのち中間画像の生成を行なった．使用した画像ペアは図 4.1(a), (b)である[11]．(b)の画像は(a)に比べて 20 度回転した方向から撮影している．この画像はスタンフォード大学の球面ガントリー[12]を使用して撮影している． 球面ガントリーは図 4.2 のような外観をしており，中心の球体にあたる部分に対象物体を設置する．左側のアームの先にカメラを取り付け，複数の関節が独立して動くことでさまざまな角度からの撮影が可能である．右側のアームの先に照明が取り付けられていて，中間の関節を用いて照明の角度を変更することが可能である．

図 4.1(a)左側からの撮影画像

図 4.1(b)右側からの撮影画像

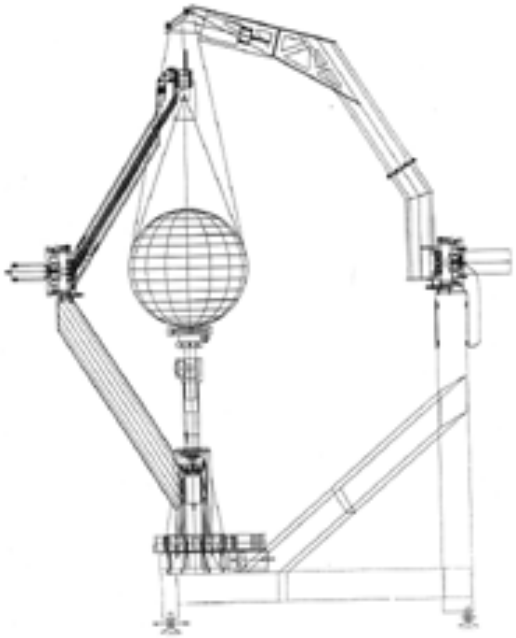

図 4.2 恐竜画像を生成した球面ガントリー

まず, 3.1 章の考え方を用いて画像の平行化を行う. 平行化された画像を図 4.3(a), (b)に示す. また、それらの平行化の様子を図 4.4 に示す. 図 4.4 には 2 本の線を引いており, 上側の線は大きな背鰭の付け根と背鰭の一つの先繋いだもの, 下側の線は目の中心と尾の先の窪みを繋いだものになっている。これらの線を見ると上側の二つと比べて下の二つの画像はそれらの線が平行になっていることがわかる。

図 4.3(a) 平行化後の左画像

図 4.3(b) 平行化後の右画像

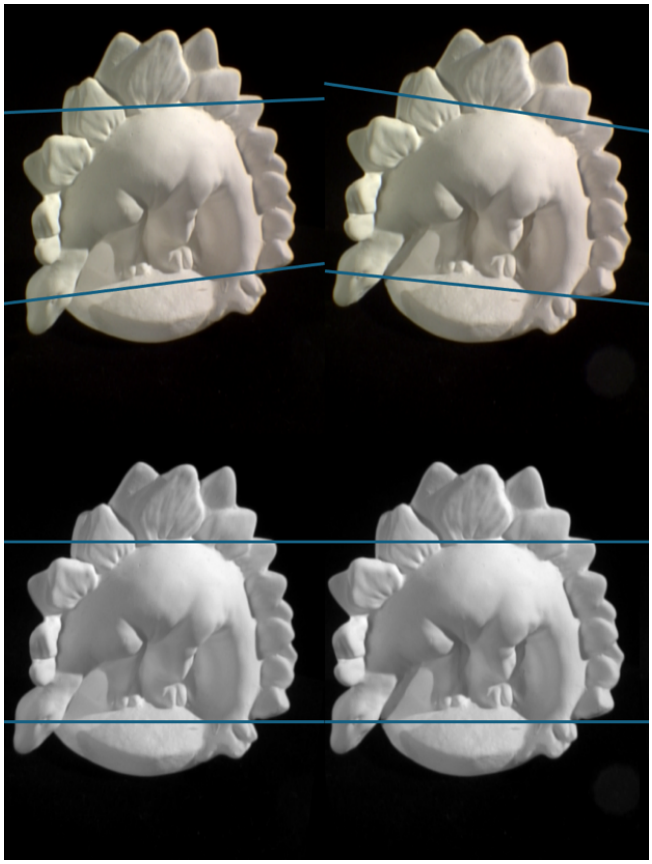

図 4.4 対応点同士を繋いで平行化の効果を示した画像. 上は入力画像, 下は平行化した画像

次に平行化された画像を用いて視差マップの作成を行なった．OpenCV のサイトを参考にして，試行錯誤的にパラメータの設定を行い，表 4 の数値を用いた。このパラメータを入力として視差マップの生成を行った結果，図 4.5 の視差マップを得た．視差マップの一部に欠損が生じている．これは左右の画像のマッチングにおいて，その部分にテクスチャが少ないためマッチングに失敗ことが理由である．この視差マップの値に基づいて各ピクセルの移動を行い，中間画像の生成を行った．その結果，中央の中間画像は図 4.6 となった．これを見ると視差マップにあるような欠損部分ができたため，OpenCV の INPAINT_NS を用いて周囲のピクセル情報を用いて補完を行なった．その結果図 4.7 となった．

また，カメラ間の輻輳角度が 30 度の画像を用いて，同じ処理を行なった．平行化の結果は図 4.8(a),(b)になった．試行錯誤的に視差マップを作成した結果を図 4.9 に示す．結果として足元部分に大きな欠損が発生し，適切な視差マップを得ることができなかった．

表 4 恐竜画像を用いて視差マップを生成する際の設定パラメータ

パラメータ	**設定値**
minDisparity	0
numDisparities	16×5
blocksize	5
P1	$8\times1\times(\text{blocksize})^2$
P2	$32\times1\times(\text{blocksize})^2$
Disp12MaxDiff	1
preFilterCap	100
uniquenessRatio	1
speckleWindowSize	50
speckleRange	1
mode	cv2. STEREO_SGBM_MODE_HH

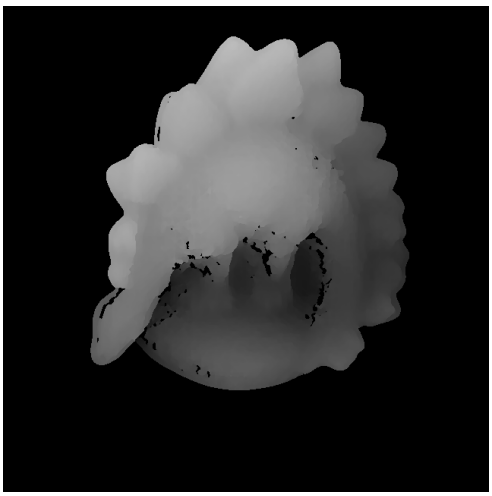

図 4.5 視差マップを作成した結果

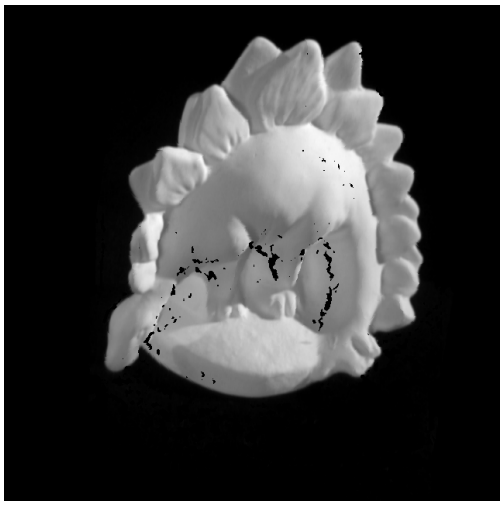

図 4.6 生成した中間画像の結果

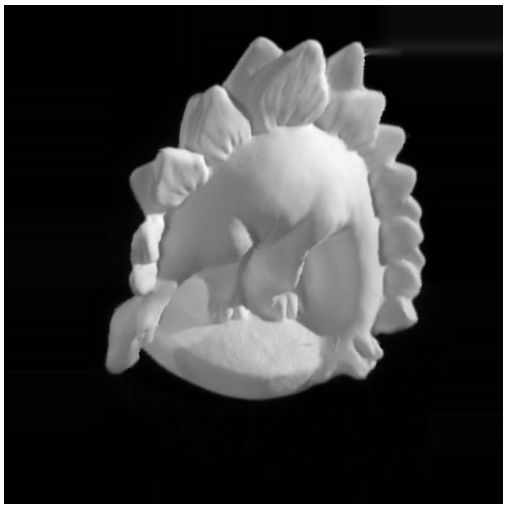

図 4.7 中間画像を補完した結果

図 4.8(a) 並行化後の左画像

図 4.8(b) 平衡化後の右画像

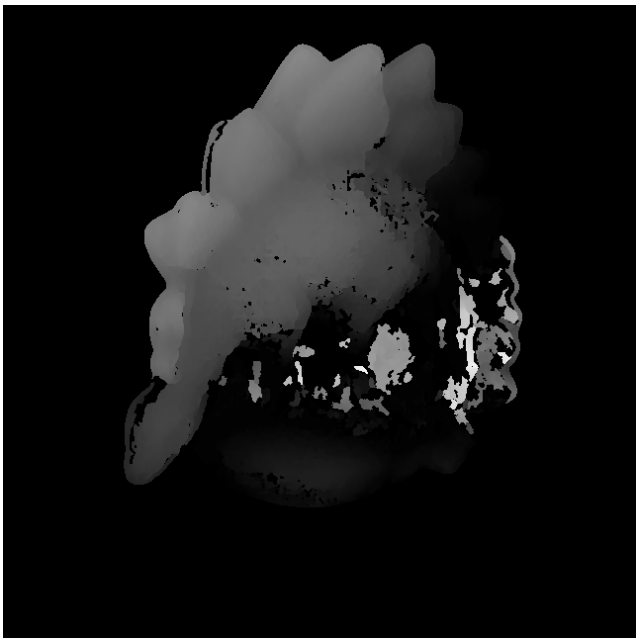

図 4.9 作成された視差マップの様子

また別の方法として，図 4.5 の視差マップの段階でペイントツール Gimp を用いて手動で欠損の補完を行なった．新たな視差マップを図 4.10 に示す．ここでの補完は直近のピクセルをコピーすることで行った．図 4.10 の視差マップを用いて作成した中間画像を図 4.11 になった．

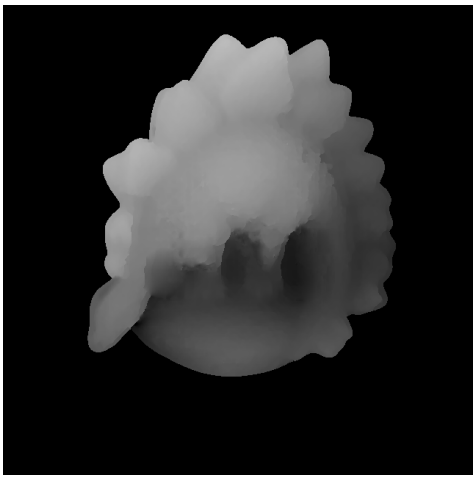

図 4. 10 Gimp を用いて補完した視差マップ

図 4. 11 補完された視差マップで生成した中間画像

図 4. 7 と図 4. 11 の恐竜画像の頭あたりを見比べてみると，図 4. 7 の方がより自然に見えるため，中間画像を生成した後に補完を行う方法を採用した．それに続いてピクセルの移動量を調整し，左右の画像の間の 67 視点の中間画像を生成し，合計 69 視点分の画像を生成した．

4.2 69 視点の画像からライトフィールドディスプレイへの投影

ライトフィールドディスプレイに投影する際に 69 視点の画像から 1440×2880 の 1 枚の画像に統合する．画像の統合は JDI が提供する SDK の内部で LFD.compute という名前のシェーダプログラムを用いて行われていた．それと同じ処理を自作する．

最終イメージはピクセルごとにカメラ ID を決定し，ピクセルごとにそれぞれ異なる画像，座標から r, g, b の画素値を得る．具体的には以下の式を用いて行われる．ここで，id.y は最終画像の y 座標，id.x は最終画像の x 座標を表している．各行に基づいてオフセット値を以下の式で計算を行う．

$$offsety = \Big(\big((2880-1) - id.y\big) \times 2 + 1\Big)\%69 \tag{4}$$

上記で求めたオフセットを利用してその行の各列で以下の式を用いてそのピクセル位置で R, G, B のそれぞれをどのカメラから所得するのかを決定する．

$$cam\ id\ r = \big((id.x \times 3) \times 3 + offsety\big)\%69 \tag{5}$$

$$cam\ id\ g = \big((id.x \times 3 + 1) \times 3 + offsety\big)\%69 \tag{6}$$

$$cam\ id\ b = \big((id.x \times 3 + 2) \times 3 + offsety\big)\%69 \tag{7}$$

上記の式を用いて各行でのオフセットを計算し，そのオフセットを用いてその行の各列のピクセルに r, g, b を割り当てる．本研究ではこの処理をより短時間で行うため，自分の環境

上の python で同じ処理を実行した．その結果図 4.12 の画像が作成され，その画像をライトフィールドディスプレイに投影する．その際，ライトフィールドディスプレイ上に綺麗に表示させる際，少し拡大する必要があった．

まず，JDI が提供しているアンモナイトを用いて実装を行なった．JDI が提供している SDK を用いて Unity 上で画像の処理を行った上で表示させた結果を図 4.13 に示す．また，python で自分自身のローカル環境で画像の処理を行い，表示させた結果を図 4.14 に示す．

図 4.12 ライトフィールドディスプレイに投影する前のアンモナイトの画像

図 4.13 JDI が提供している環境での表示

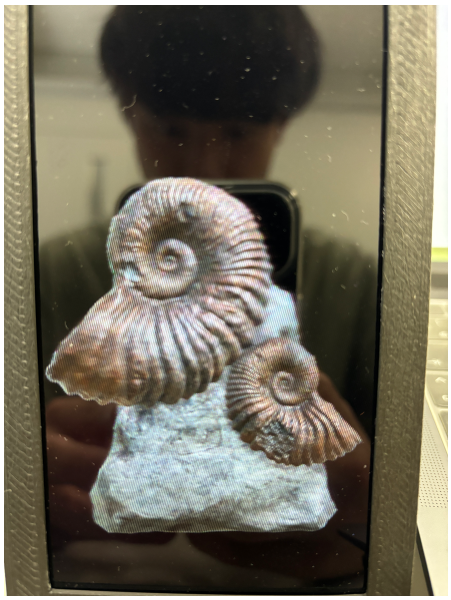

図 4.14 Python で書き直した表示

図 4.13 と図 4.14 を詳細に観察したところ，全く同じ表示になっていたため，正しく処理と表示ができていると考えた．

それを踏まえて恐竜の 69 枚画像を用いてライトフィールドディスプレイへの投影を行った．作成された画像は図 4.15 となり，その画像をディスプレイに投影し拡大処理した結果が図 4.16 となった．

ライトフィールドディスプレイに表示させた結果として，ノイズや画像の歪みは見られなかったが，アンモナイト画像の場合の見え方と比べて立体感が少ないという結果になった．理由としては，用いた二つの画像間の角度が約 20 度でアンモナイトでの 100 度と比べてかなり狭くなってしまったことが原因であると考えた．

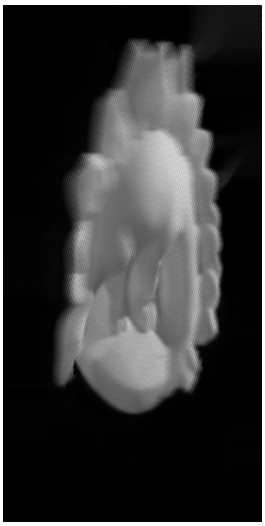

図 4.15 ディスプレイに投影する前の画像

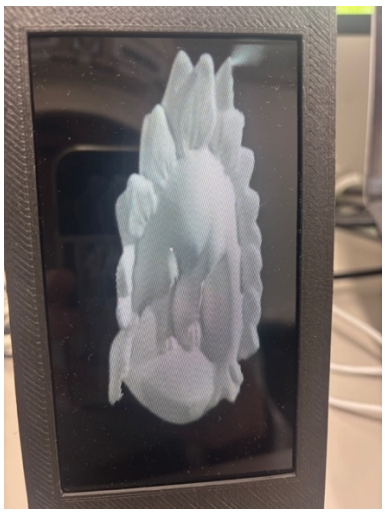

図 4.16 ディスプレイに投影した結果

第 5 章 結論

本論文では，実物体を 2 視点からカメラ撮影した画像を用いて，JDI 社のライトフィールドディスプレイに表示させる 69 個の多視点画像を生成する研究を行なった．JDI が提供するツールを使う場合，対象物体を 3D スキャンするか，3 次元 CG 技術で 3D モデルを作成し，Unity 環境でディスプレイ投影する．本研究では，実物体を 2 視点から撮影した画像を用いて中間の 67 視点分の画像を生成して，ライトフィールドディスプレイに投影させるために，ステレオマッチングを用いた方法を提案した．

結果としてカメラの内部パラメータとカメラの位置，角度が既知の場合，ステレオ平行化とステレオマッチングを行うことで，67 視点の中間画像を生成できた．また，Unity で行なっていた 69 視点分の画像統合処理を python を用いて行うことで，ライトフィールドディスプレイに物体を投影することができた．また，3D Gaussian Splatting を応用した Splatt3R を用いて二視点の画像から自由視点画像を作成することができ，そちらを使用して 69 視点分の画像を補える可能性を見出せた．．

残された課題はステレオマッチングを行う際に，平行化やマッチングを正確に行える角度に制限があったため，JDI のディスプレイで可能な約 100 度の視野角を実現することがで

きず，本研究では約 20 度の視野角にとどまった，そのため，ディスプレイを見た際の立体感が浅く見えてしまった．Splatt 3R で作成された自由視点画像が色情報を持たない PLY ファイルであったため，各視点画像を生成する方法としてスクリーンショットのみの提案になってしまったためそれ以外の方法を検討する必要がある．

参考文献

[1] Lambooij, M.T.M., et al., “Visual discomfort and visual fatigue in stereoscopic displays: a review”, *Journal of Imaging Science and Technology, 53*(3), 1-14, 2009

[2] Lanman, D. and Luebke, D., “Near-eye light field displays”, *ACM Transactions on Graphics (TOG), 32*(6), Article 220, pp. 1-10, 2013

[3] Maimone, A., et al., “Holographic near-eye displays for virtual and augmented reality”, *ACM Transactions on Graphics (TOG)*, 36(4), Article 85, pp.1-16, 2017

[4] Unity Technologies. (2018). 物理カメラ. Unity Manual. [Accessed Dec. 20, 2024] https://docs.unity3d.com/ja/2018.3/Manual/PhysicalCameras.html

[5] Horn, B. K. P. and Schunck, B. G., "Determining Optical Flow", *Artificial Intelligence*, 17(1-3), pp.185-203, Aug. 1981

[6] Dosovitskiy, A., et al., "FlowNet: Learning Optical Flow with Convolutional Networks. " *IEEE International Conference on Computer Vision (ICCV) 2015, pp.2758-2766*, Dec. 2015

[7] R. Szeliski, "Computer Vision: Algorithms and Applications. " *Springer*, 2022

[8] OpenCV. StereoSGBM – Camera Calibration and 3D Reconstruction. [Accessed Dec. 20, 2024] https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html

[9] Kreis, B. et al., "3D Gaussian Splatting for Real-Time Radiance Field Rendering, " ACM TOG, Vol. 42, Article 139, pp. 1-14, 2023.

[10] B. Smart, et al. , "Splatt3R: Zero-shot Gaussian Splatting from Uncalibrated Image Pairs" arXiv:2408.13912, 2024.

[11] Multi-View Stereo Datasets. [Accessed Dec. 20, 2024] https://vision.middlebury.edu/mview/data/

[12] Stanford Spherical Gantry [Accessed Dec. 20, 2024] https://graphics.stanford.edu/projects/gantry/

謝辞

本論文を作成するにあたり，蚊野浩教授には本研究の企画から執筆に至るまで多大なる丁寧なご指導，ご助言を賜りましたことを深く感謝いたします.