修 士 論 文 の 和 文 要 旨

研究科・専攻	大学院　情報システム学 研究科　情報ネットワークシステム学　専攻　博士前期課程		
氏　　名	江頭　明	学籍番号	1152004
論 文 題 目	並列処理を用いた錯視シミュレーションシステムの構築		

要　　旨

人間の視覚は網膜の情報をそのまま投影しているわけではなく，視覚神経系と呼ばれる脳細胞が情報処理を行うことで実現されている機能である．この視覚の働きによって人間は物体の認識や表情の読み取りといった高度な情報処理をすることが実現できている．その視覚神経系のメカニズム解明の恩恵は工学的な分野にまで及ぶものであり，その代表的な例は顔認識技術や自動車の衝突回避技術などが挙げられる．

視覚神経系のメカニズム解明の鍵となるのが錯視現象である．錯視現象は，図 1 の様に静止している画像が勝手に移動して見える現象のことである．この現象は人間の視覚神経系の画像処理が誤動作を起こすことによって発生する現象である．よって，逆にこの現象を解明することは視覚神経系の解明にも期待されている．

錯視現象を研究するにおいて生理学的な研究の他に計算機による数理シミュレーションは有効な研究手段である．しかし，視覚神経系シミュレーションでは細胞の１つ１つの活動をシミュレートする特性上，膨大な計算量を要し，それに伴いシミュレーション時間は膨大なものとなることが問題となっている．

本研究では，GPU 搭載 PC クラスタを使用して，並列処理による高速な錯視シミュレーションシステムの構築をした．錯視シミュレーションの成果としては，単純な輝度パターンを使用した錯視シミュレーションにおいて，人間に対する反応をシミュレートすることに成功した．また，高速化にあたっては，GPU 内でのアルゴリズムの最適化，袖領域の交換処理の排除など，通信処理の最適化によって，基礎的な実装と比較して 70 %の処理時間の最適化した．

さらに，人間の視覚の特性の一つである”中心視・周辺視”についてもシミュレートも実現した．“中心視・周辺視”対応化によって，処理速度が大幅に低下したが，パイプライン化・通信処理の最適化で高速化を図り，”中心視・周辺視”対応の基礎的な実装と比較して 50 %以上の処理時間の削減を達成した．

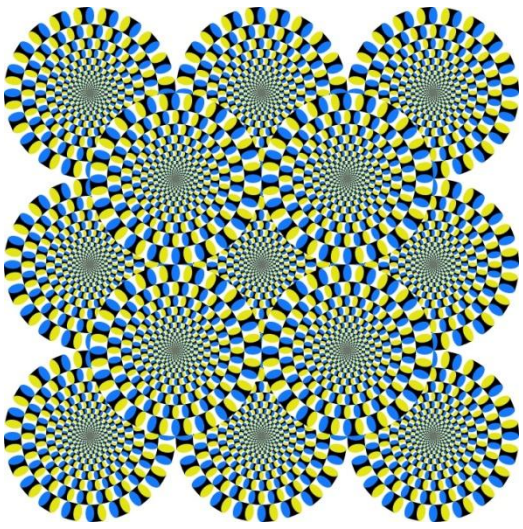


図１　錯視現象の例(静止している画像なのに移動して見える)

電気通信大学大学院情報システム学研究科

2013 Jan.

修　士　論　文

並列処理を用いた
錯視シミュレーションシステムの構築

主指導教員　吉永 努 教授

指導教員　入江 英嗣 准教授

指導教員　大坐畠　智 准教授

平成 25 年 1 月 24 日

提 出 者

所 属　大学院情報システム学研究科

情報ネットワークシステム学専攻

学籍番号　1152004

氏 名　江頭 明

(表紙裏)

並列処理を用いた錯視シミュレーションシステムの構築

ネットワークコンピューティング学講座 吉永 研究室　　1152004　江頭　明
指導教員：吉永 努 教授

1　背景

人間の視覚は視覚神経系と呼ばれる脳細胞が情報処理を行うことで実現されている機能である．視覚神経系のメカニズム解明の恩恵は医学的分野だけでなく，工学的な分野にまで及ぶ．その代表的な例は顔認識技術や自動車の衝突回避技術であり，今後も視覚神経系による画像処理技術は様々な分野で応用が期待されている．

2　錯視現象

この有益な視覚神経系のメカニズムを解明するにあたって有力な手がかりとなりうるのが錯視現象である．錯視現象は，図 1 の様に静止している画像が勝手に移動して見える現象のことである．

錯視現象は視覚神経系の画像処理と密接に関係している．通常，人間の眼球は “固視微動” と呼ばれる運動を常時行なっている．視覚神経系はこの “固視微動” を検出したオプティカルフロー値を元に補正をかけている．しかし，図 1 のような錯視画像の場合は正常なオプティカルフロー値が検出されず，“固視微動” の補正が正常に機能しない．これにより，錯視現象は発生する (詳細は図 2 を参照)．

このように錯視現象は視覚神経系の働きに密接に関係しているため，錯視現象の解明は視覚神経系の解明にもつながると期待されている．本研究ではこの錯視現象を再現するシミュレーションシステムを構築する．

本研究では，錯視現象を再現するために，図 3 の様な数理モデルを用いており，畳み込み処理を行うことで神経細胞の働きをシミュレートする．本数理モデルで用いる畳み込み処理は，通常の OpenCV の畳み込み処理と比較して **1600** 倍もの計算量を要する．そのため本研究では，この膨大な計算量の問題に対して並列処理による高速化で対処している．

3　基本的実装と高速化案

基本的実装 (以降は baseline) は以下の流れで処理を行う．各ノードは画像領域別に処理を分担する．また，計算量が特に多い畳み込み処理とオプティカルフロー計算処理は GPU にて演算する．

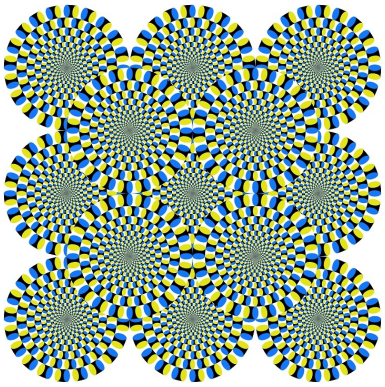


図 1: 錯視現象の例 (rotating snake [1])

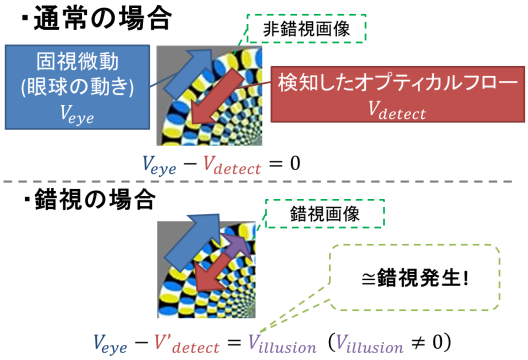


図 2: 錯視現象が発生するメカニズム [2]

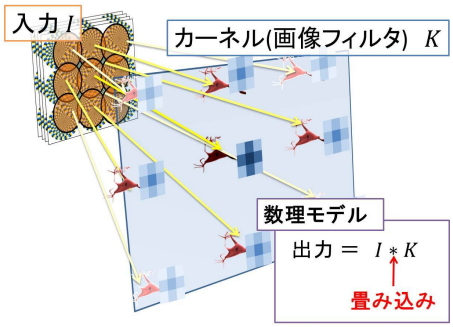


図 3: 視覚神経系の基本数理モデル．

- Step.1 画像データの分配
- Step.2 画像データから入力動画を生成
- Step.3 畳み込み処理
- Step.4 袖領域の交換処理
- Step.5 オプティカルフロー計算
- Step.6 Gather 処理

高速化については，動画生成処理の省略 (gpu_eye_sim) と畳み込み処理の最適化 (packed_conv)，そして，交換処理の省略 (no_exchange) を実施した．

4　錯視シミュレーションの検証実検

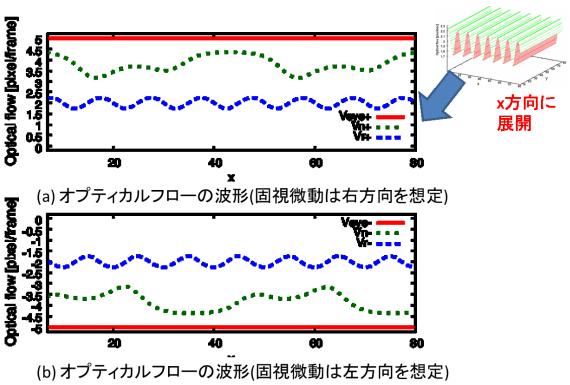
本錯視シミュレーションシステムが錯視現象を再現できているかを検証するため実験を行った．図 4 は検証実験の結果である．

図 4 に示すように，錯視現象を引き起こす画像の場合 (青点線)，実際の眼球の移動量 (赤実線) とは大きくかけ離れるオプティカルフロー値を検出することが確認された．

対して，錯視現象を引き起こさない画像の場合 (緑点線)，実際の眼球の移動量に近い値を検出することができた．これらの結果は，人間が錯視・非錯視画像を見た場合と類似した結果であり，本錯視シミュレーションシステムで錯視現象を再現できたと結論づけられる．

5　性能評価

図 6 で示すように，高速化をすすめるに従って処理時間が削減された．特に，gpu_eye_sim から packed_conv では，60 %以上の処理時間の削減が実現できた．こ



(a) オプティカルフローの波形(固視微動は右方向を想定)

(b) オプティカルフローの波形(固視微動は左方向を想定)

図 4: 錯視シミュレーションの結果．赤の実線は実際の眼球の移動量，緑の点線は錯視画像を用いたシミュレーション結果，青の点線は非錯視画像を用いたシミュレーション結果を表している．

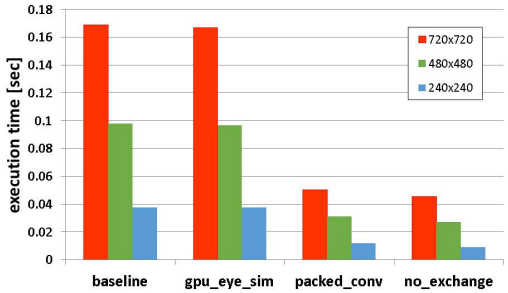


図 5: 各高速化案においての処理時間 (ノード数: 16)

れは畳み込み処理の最適化により計算量を $\frac{1}{10}$ に削減した効果である．

6 中心視周辺視

視覚には，中心では明瞭に認識をし，周辺部になるに従って不明瞭になってゆく性質がある [3]．錯視現象においても中心視周辺視は大きく影響する．中心視周辺視の現象を再現するには，数理モデルとして空間方向のカーネルを修正する必要がある．具体的には，周辺部には強い空間ボケ処理を，中心部には弱い空間ボケ処理をかけるようにカーネルを設定する必要がある．

6.1 基本的実装と高速化案

基本的実装 (baseline_CP) ではカーネルの設定処理を新たに追加した．また，より多様なカーネルに対応するため，畳み込み処理のカーネルのサイズを拡大した．これにより，baseline 時の計算量と比較して 8.5 倍もの計算量となった．

高速化にあたっては，まずカーネルの設定処理をパイプライン化でオーバーラップさせた (pipeline)．次にパイプライン化により顕在化した通信の衝突の問題を，Gather 処理の最適化によって対処した (2step_gather)．

6.2 性能評価結果

性能評価の結果は図 6 に示す．各高速化を適用したことで，最終的には 50 %以上の処理時間の削減を実現できた．

6.3 錯視シミュレーションの検証実験

図 7 は錯視シミュレーション結果である．なお，シミュレーション結果は色の明暗でオプティカルフロー値の大きさを表している (明るいほどオプティカルフロー値が大きい)．中心視周辺視を適用していない場合の結果と比較すると，周辺部へゆくにしたがってオプティカルフロー値が正確に検出おらず，錯視が発生

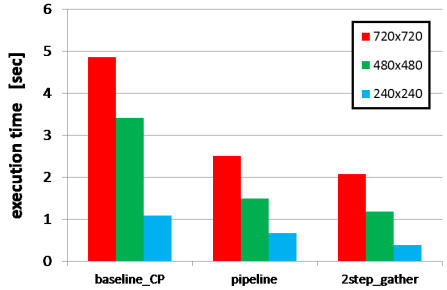


図 6: 中心視周辺視を適用した場合の各高速化案においての処理時間 (ノード数: 16)

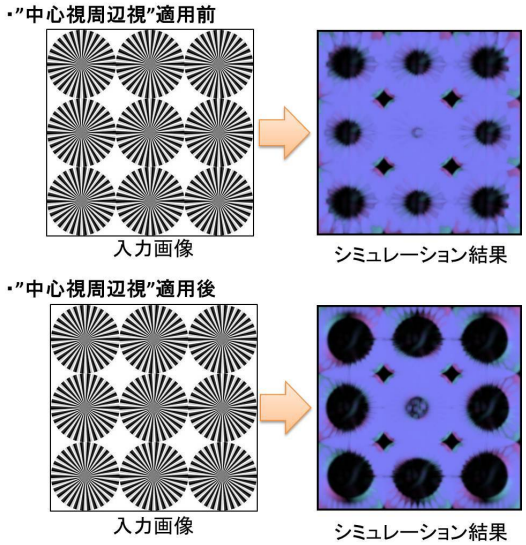


図 7: 錯視シミュレーション結果 (中心視周辺視を適用)

することを示している．これは，周辺部にいくに従って錯視現象が発生する人間の反応と同様の反応である．よって，本システムで中心視周辺視の性質を再現した錯視現象の再現に成功したと判断する．

7 まとめ

本研究では，GPU 搭載 PC クラスタを使用した錯視シミュレーションシステムの構築について述べた．単純な輝度パターンを使用した錯視シミュレーションにおいて，人間に対する反応をシミュレートすることに成功した．高速化にあたっては，GPU 内でのアルゴリズムの最適化，袖領域の交換処理の排除など，通信処理の最適化によって，基礎的実装と比較して 70 %の処理時間を短縮した．

さらに，人間の視覚の特性の一つである “中心視・周辺視” についてもシミュレートを実現した．“中心視・周辺視” 対応化によって，処理速度が大幅に低下したが，パイプライン化などの各高速化により，“中心視・周辺視” 対応の baseline_CP と比較して 50 %以上の処理時間の削減を達成した．

参考文献

[1] 北岡明佳. 北岡明佳の錯視のページ. http://www.ritsumei.ac.jp/ akitaoka.

[2] Anton L. Beer, Andreas H. Hecke, and Mark W. Greenlee. A motion illusion reveals mechanisms of perceptual stabilization. *PLoS ONE*, Vol. 3, p. e2741, 7 2008.

[3] Neil R.Carlson. カールソン神経科学テキスト-脳と行動. 丸善, 2006.

目 次

図 目 次

表目次

第1章　序論

1.1　背景

1.1.1　視覚神経系

人間の視覚は網膜の情報をそのまま投影しているわけではなく，視覚神経系と呼ばれる脳細胞が情報処理を行うことで実現されている機能である．この視覚の働きによって人間は物体の認識や表情の読み取りといった高度な情報処理をすることが可能となる．

視覚神経系のメカニズム解明の恩恵は医学的分野だけでなく，工学的な分野にまで及ぶ．その代表的な例は顔認識技術や自動車の衝突回避技術であり，今後も視覚神経系による画像処理技術は様々な分野で応用が期待されている．

1.1.2　錯視現象

この有益な視覚神経系のメカニズムを解明するにあたって有力な手がかりとなりうるのが錯視現象である．錯視現象は，図 1.1.1 の様に静止している画像が勝手に移動して見える現象のことである．本研究ではこの錯視現象を再現するシミュレーションシステムを構築する．錯視現象は人間の視覚神経系の画像処理が誤動作を起こすことによって発生する現象である．また，逆にこの現象を解明することは視覚神経系の解明にも期待されている．

1.1.3　視覚神経系シミュレーション

視覚神経系の研究において，生理学的実験と同様に数理学的なシミュレーションが広く用いられている．視覚神経系の詳細なシミュレーションを対象とした研究として，Blue Brain Project[3] が挙げられる．このプロジェクトでは IBM Blue Gene/L を用いて，コンパートメントモデルと呼ばれる詳細な神経細胞モデルを並列計算によりシミュレーションしている．コンパートメントモデルのシミュレーションによって，細胞個々の複雑な特性を記述・再現することができる．しかし，シミュレーション対象としているモデルの粒度が小さすぎるため，視覚神経系の機能の再現や理解のためには適切なモデルとは言えない．一方で，モデル粒度を大きくし，細胞の入出力関

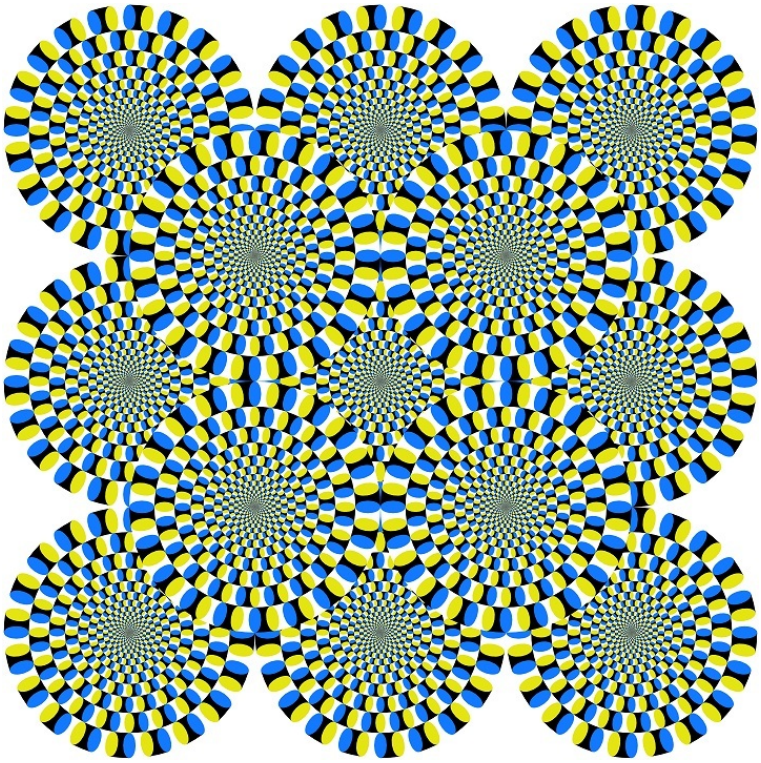


図 1.1.1: 錯視現象の例 (rotating snake [1])

係だけに着目したモデルが様々提案されている．その中でも線形モデルは最も単純なモデルであるが，視覚機能の再現と理解のために十分なモデルと言える．事実，テクスチャ解析や色情報処理などの視覚機能モデルの多くが線形モデルによって表現されている [4, 5, 6]．すなわち，視覚系の機能をシミュレーションするためには単純な線形モデルで十分である場合が多い．

しかし，視覚神経系シミュレーションでは細胞の１つ１つの活動をシミュレートする特性上，膨大な計算量を要し，それに伴いシミュレーション時間は膨大なものとなることが問題となっている．

1.1.4 並列処理

視覚神経系シミュレーションシステムでは畳み込み処理が大半を占める．この膨大な量の畳み込み処理を高速化することがシミュレーションシステム全体の高速化の一番の近道となる．

畳み込み計算の代表的な高速化手法として，FFT(Fast Fourier Transform) が挙げられる．FFT は高速な畳み込み計算を実現し，ターゲット (入力信号) とカーネル (フーリエ基底) の畳み込みを高速に計算する際に用いられる．しかし，利用できるのはカーネルが時空間で均一である場合に限られる．視覚神経系シミュレーションで用いられる畳み込み計算は，空間的に異なるカーネルを使用するため FFT を適用することができない．

FFT とは別に有効な高速化の手段としては並列処理技術が挙げられる．本研究では OpenMPI[7](複数ノード) と CUDA[8](1 ノード) を用いた 2 段階での並列処理で高速化を図る．

1.2 先行研究と課題

視覚神経系シミュレーションの研究は多く存在する．特に本研究と同じように，網膜に近い段階の視覚神経系 (V1 細胞) に特化した研究では，理研が視覚神経系シミュレータを作成している [9]. また米軍では，同様の視覚神経系のシミュレーターを PS3 のクラスタで実装している [10]．この研究では 2.6 億個もの神経細胞のシミュレーションを実現している．

しかし，錯視現象という特別な現象に特化したシミュレーターは，本研究以外にはいまだ存在してしない．また本研究は，膨大な計算量に起因する処理時間の問題も GPU 搭載 PC クラスタによる並列処理にて対処している．これらの点に本研究の独自性がある．

本研究は以下のような過去の研究から発展させていったものである．

1. 1 次元輝度パターンを対象とした錯視シミュレーションシステム [11].
   錯視を誘発する 1 次元の輝度パターンを対象とした錯視シミュレーションシステムの研究である．

2. 2 次元のオプティカルフローの視覚神経系シミュレーションシステム [12].
   1. の研究で用いられた，オプティカルフロー計算を 1 次元から 2 次元に拡張し，さらに GPU を活用することでより高速な視覚神経系シミュレーションシステムを構築した．

本研究は，2. の研究を元にしている．入力情報を動画から 2 次元画像を対象とするように変更するなど，錯視シミュレーションに特化した錯視シミュレーションシステムの構築を行なっている．

1.3 研究の概要

本研究では，錯視シミュレーションシステムの構築と並列処理技術による高速化手法について研究を行った．

本論文では，大きく分けて視覚神経系の研究の観点と並列処理の研究の観点の２つの観点で研究を報告する．視覚神経系の研究の観点では，錯視画像を元にした錯視シミュレーションの結果について報告する．並列処理の研究の観点では，GPU(NVIDIA tesla C1060) を用いた１ノードレベルの並列処理の高速化，OpenMPI を用いた多ノードレベル (16 ノード) の高い並列処理効率について報告する．

よって、本書の構成は、以下の通りとなる。

- 第 2 章では、錯視現象についてとその錯視現象をシミュレートする際に用いた数理モデルについてまとめる.
- 第 3 章では、本研究で作成した環境の概要と錯視シミュレーションシステムの実装方法についてまとめる.
- 第 4 章では、錯視現象が実際に再現できたのかを検証するための検証実験についてまとめる.
- 第 5 章では、高速化の効果を確認するために行った性能評価についてまとめる.
- 第 6 章では、回転運動の錯視シミュレーションの実装法と錯視シミュレーションの結果についてまとめる.
- 第 7 章では、視覚神経系の性質の一つである中心視周辺視を錯視シミュレーションシステムに新たに適用し，実装法・高速化・シミュレーション結果をまとめる.
- 第 8 章では、結論と今後の課題を述べる。

第2章　錯視シミュレーションシステム

本章では，錯視シミュレーションシステムをどのように構築すればいいか，その数理学的な手法について説明する．始めに，視覚神経系の説明とその視覚神経系の処理によって発生してしまう錯視現象のメカニズムについて述べる．次に，この錯視現象を再現するために必要な数理モデルについての説明する．最後に，数理モデルを用いた錯視シミュレーションシステムの計算処理の全体像について解説する．

2.1　視覚神経系の全体像

人間の視覚神経系は図 2.1.1 のようなネットワークで情報処理を行う．網膜から入力された情報は外側膝状態を経由し V1 と呼ばれる一次視覚野に情報が伝わる．本シミュレーションシステムでは，この一次視覚野に注目したシステムを構築する．

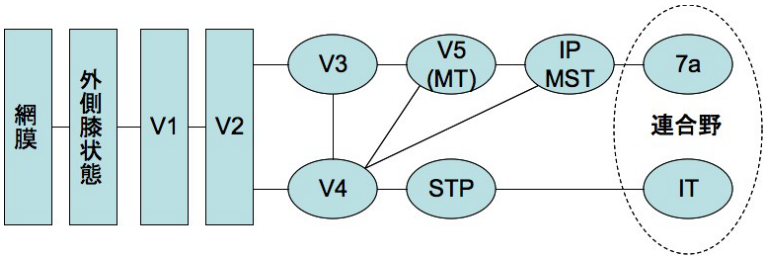


図 2.1.1: 視覚の処理経路

2.2　錯視が発生するメカニズム

錯視現象のメカニズムは以下の有力な仮説 [13] によって説明することができる．

実は，人間は物体を見ているときに完全に静止して見ているわけではない．人間の眼球は絶えず運動をしている．この運動のことを “固視微動” と呼ぶ．しかし人間の視覚は，静物を静止しているものと認識する．これは視覚神経系が固視微動の打ち消し処理をしているためである．

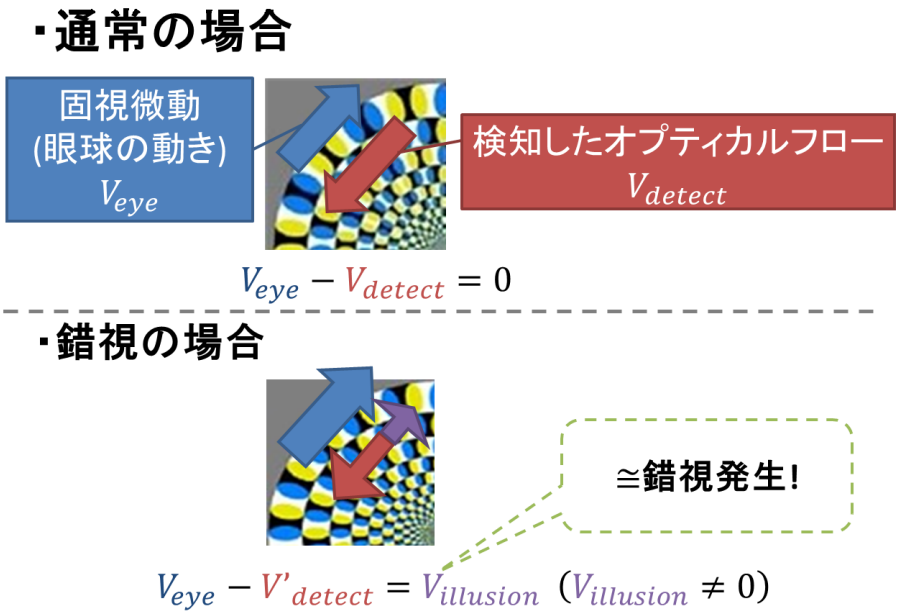


図 2.2.1: 錯視現象が発生するメカニズム

しかし，錯視現象を引き起こすような画像の場合だとオプティカルフローの誤検出が発生してしまう．図 2.2.1 で示されるように，誤検出されたオプティカルフローは打ち消し処理の失敗を招き，未完全な打ち消し処理によって残ったオプティカルフローが錯視として認識されてしまう．この仮説に基づき，検出したオプティカルフローである V_{detect} と実際の移動速度 V_{eye} を比較することで錯視現象かどうかを判断する．本研究の錯視シミュレーションシステムは，この V_{detect} を求める視覚神経系シミュレーションを行う．

2.3　基本的な数理モデル

本シミュレーションシステムは図 2.3.1 で示される数理モデルを採用している．

この数理モデルは具体的には図 2.3.1 の左下のような畳み込み演算によって表現できる．神経細胞は前段の神経細胞の出力信号を入力として受け，後段の細胞へ出力信号を伝える．この際，各入力信号 I はシナプス荷重 (カーネル)w と呼ばれる重み係数で積和される．すなわち，畳み込み (もしくは相関計算) で表現することができる．以降では，この畳み込み演算の内容とカーネルの算出法について述べる．

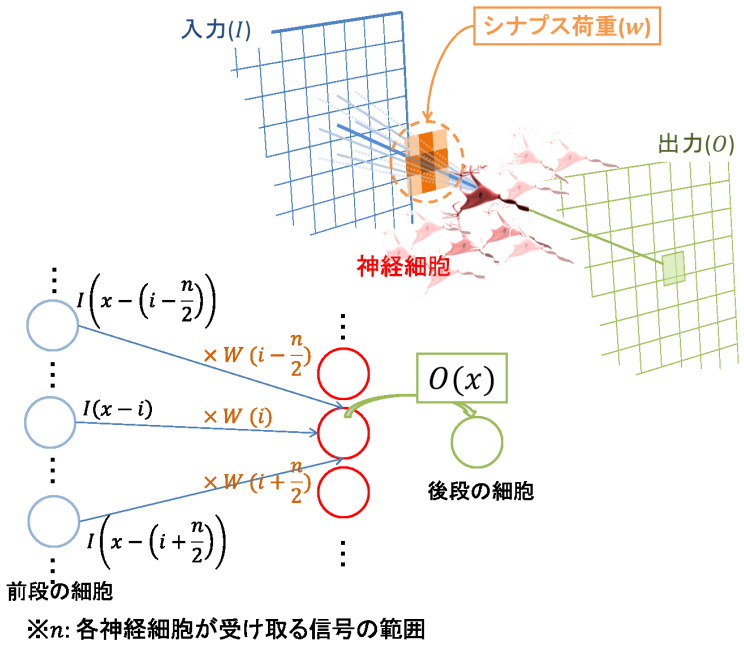


図 2.3.1: 視覚神経系の基本数理モデル

2.4　3 次元畳み込み

通常の画像処理に用いられる畳み込み処理では，2 次元の畳み込み処理が適用される．しかし，視覚神経系は時間的にも特性を有する．そのため，今回の数理モデルでは 2 次元から更に時間軸 t に拡張した 3 次元の畳み込み処理を採用している．

以下の数式は，時間 t における空間座標 (x,y) に位置する細胞の出力 $(O(x,y,t))$ に注目した数理モデルである．下式 2.4.1 のように，入力動画 (I) と 3 次元カーネル (K) を元にした 3 次元畳み込みで表現されている．

$$O(x,y,t) = \sum_{\xi=-\frac{n}{2}}^{\frac{n}{2}} \sum_{\eta=-\frac{m}{2}}^{\frac{m}{2}} \sum_{\tau=0}^{l} K(\xi,\eta,\tau) I(x-\xi, y-\eta, t-\tau) \tag{2.4.1}$$

(n,m,l: 3 次元カーネル (K) の時空間的範囲 (カーネルサイズ))

以降では，入力動画 I と視覚神経系の細胞の働きを再現するために設定する 3 次元カーネルについて説明する．

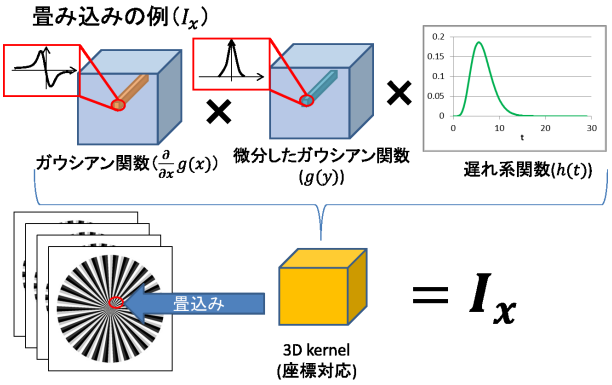


図 2.4.1: 3 次元畳み込み処理のイメージ図

2.4.1 入力画像から入力動画の生成

本錯視シミュレーションシステムでは，錯視を引き起こす入力画像から入力動画 I を生成している．これは，錯視の要因とされている固視微動を再現するためである．具体的には，図 2.4.2 のように入力画像を移動させることで動画情報を生成する．

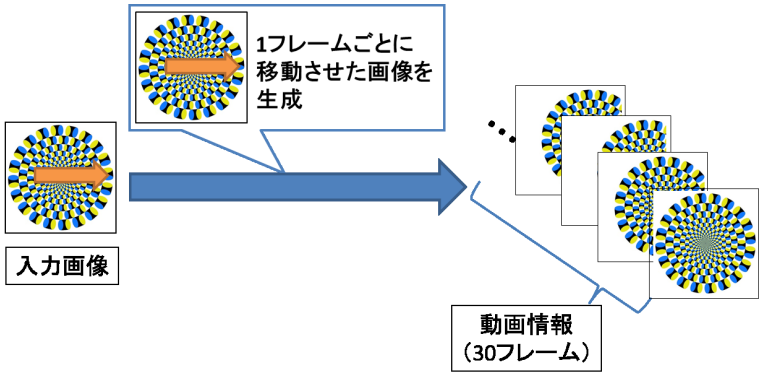


図 2.4.2: 入力動画生成法

2.4.2 ガウス関数と遅れ系を用いた 3 次元カーネル

視覚神経系の神経細胞の特徴として，空間的なボケと時間的な遅れという 2 つの特徴がある．これらの特徴は数学的に再現することが可能で，空間的なボケならガウス関数 ($g(x)$, $g(y)$)，時間的な遅れならば遅れ系伝達関数 ($h(t)$)，とそれぞれ再現することが可能である．これらの関数で構成されるカーネルと入力動画 I の畳み込みにより視覚神経系の観測結果である I' を算出することができる (式 2.4.2).

$$I' \simeq \{g(x)g(y)h(t)\} * I \tag{2.4.2}$$

この式の両辺を偏微分することで，空間微分・時間微分の観測結果である I_x, I_y, I_t は，以下のような式でそれぞれ表される (式 2.4.3, 2.4.4, 2.4.5,).

$$\begin{aligned} I'_x = \frac{\partial}{\partial x} I' &\simeq \frac{\partial}{\partial x} \left[\{g(x)g(y)h(t)\} * I\right] \\ &= \left\{\frac{\partial g(x)}{\partial x} g(y)h(t)\right\} * I \end{aligned} \tag{2.4.3}$$

$$\begin{aligned} I'_y = \frac{\partial}{\partial y} I' &\simeq \frac{\partial}{\partial y} \left[\{g(x)g(y)h(t)\} * I\right] \\ &= \left\{g(x)\frac{\partial g(y)}{\partial y} h(t)\right\} * I \end{aligned} \tag{2.4.4}$$

$$\begin{aligned} I'_t = \frac{\partial}{\partial t} I' &\simeq \frac{\partial}{\partial y} \left[\{g(x)g(y)h(t)\} * I\right] \\ &= \left\{g(x)g(y)\frac{\partial h(t)}{\partial t}\right\} * I \end{aligned} \tag{2.4.5}$$

I_x, I_y, I_t はオプティカルフロー計算に用いられ，最終的には人間の目が知覚したオプティカルフロー値である V_{detect} を算出することができる．

2.4.3 ガウス関数

まずは，空間ボケを再現するためのガウス関数について説明する．ガウス関数は，一般的に正規関数として統計学で用いられている．また画像処理においても，画像平滑化の手法として用いられている．今回の研究では，空間ボケを再現するために使用する．

ガウス関数は次の式 2.4.6 として表される．σ はガウス関数のぼかしの強さを表している．

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{2.4.6}$$

また，ガウス関数を x 方向に微分すると次式 2.4.7 になる．

$$\frac{\partial g(x)}{\partial x} = -\frac{x}{\sigma^3\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} \tag{2.4.7}$$

これら式 2.4.6, 2.4.7 を用いて 2 次元上に展開する (式 2.4.8, 2.4.9).

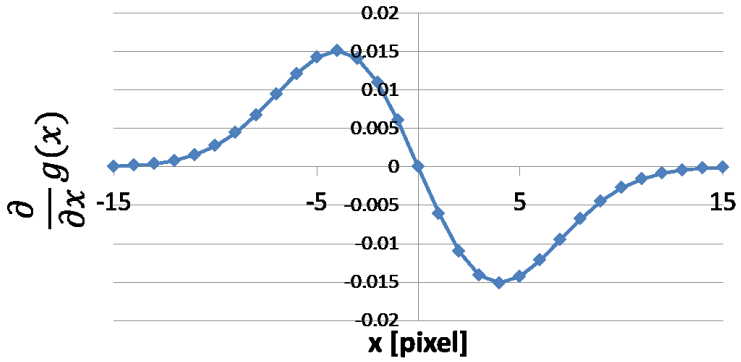


図 2.4.3: 微分したガウス関数

$$\frac{\partial g(x)}{\partial x}g(y) = \frac{-x}{\sigma^3\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}}\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{y^2}{2\sigma^2}} \tag{2.4.8}$$

$$g(x)\frac{\partial g(y)}{\partial y} = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{x^2}{2\sigma^2}}\frac{-y}{\sigma^3\sqrt{2\pi}}e^{-\frac{y^2}{2\sigma^2}} \tag{2.4.9}$$

図 2.4.3 は x 方向に微分したガウス関数を表している．

xy 方向にぼかす場合は，図 2.4.4 の様に $g(x)g(y)$ と xy 方向ともにガウス関数を適用すれば良い．対して，x 方向のみ微分する場合は図 2.4.5 の様に $\frac{\partial g(x)}{\partial x}g(y)$ と $g(x)g(y)$ を x 方向に偏微分したカーネルを適用することで実現できる．これにより，空間ボケを再現しつつ x 方向の微分が可能になる．y 方向についても同様で，y 方向に偏微分したカーネルを適用すればよい．

2.4.4 遅れ系

次に，遅れ系は入力が変化しても出力がそれに対応して変化するまでには，一定の時間を要するという細胞の特性を表している．本研究では遅れ系を用いることで，図 2.4.6 のようにフラッシュのような光がゆっくりと変化する光として認知される現象を再現する．

遅れ系の伝達関数は次のような関数で表される．

$$\frac{1}{\left(s+\frac{1}{\tau}\right)^n} \tag{2.4.10}$$

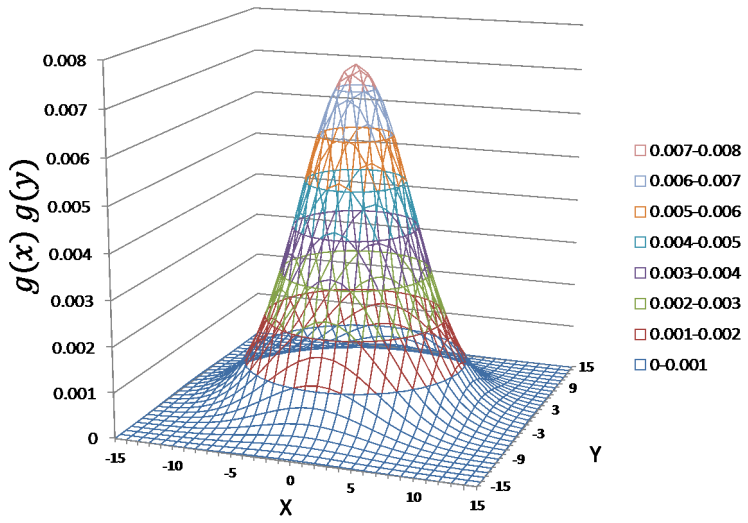


図 2.4.4: 2 次元ガウス関数

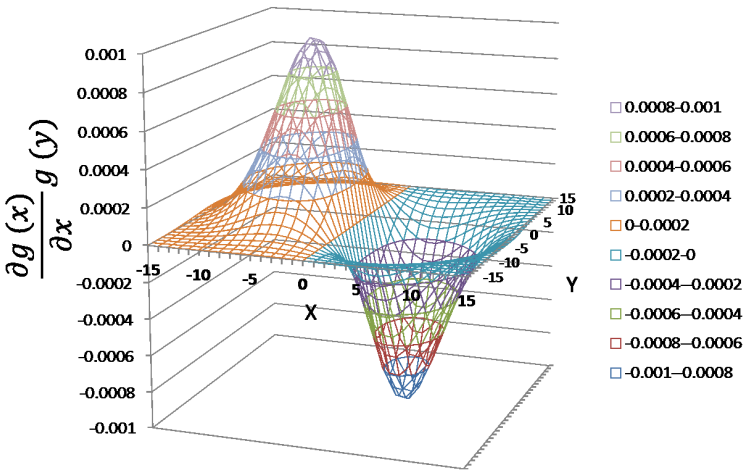


図 2.4.5: x 方向に偏微分した 2 次元ガウス関数

図 2.4.6: 遅れ系の例

逆ラプラス変換を行い得られる伝達関数は，以下の式で与えられる．

$$h(t) = \mathcal{L}^{-1}\left[\frac{1}{\left(s+\frac{1}{\tau}\right)^n}\right] \tag{2.4.11}$$

$$= \frac{t^{n-1}}{\tau^n \Gamma(n)} e^{-\frac{t}{\tau}} \tag{2.4.12}$$

例えば，$\tau = 1, n = 8$ とすると図 2.4.7 のようなグラフとなり，これは 7 フレーム目の画像が一番影響度の高くなるように畳み込み処理が施される．ここで，τ は時定数である．時定数とは，ある現象の即応性のことである．本研究では時定数が小さいほど物理速度に近づき，大きいほど物理速度よりも遅れることを意味する．また，n は目の網膜からこの式を適用する脳の部位までのステージ数を意味している．本研究では一次視覚野のシミュレーションを行うため，$n = 8$ で 8 ステージ目の部位を想定している．

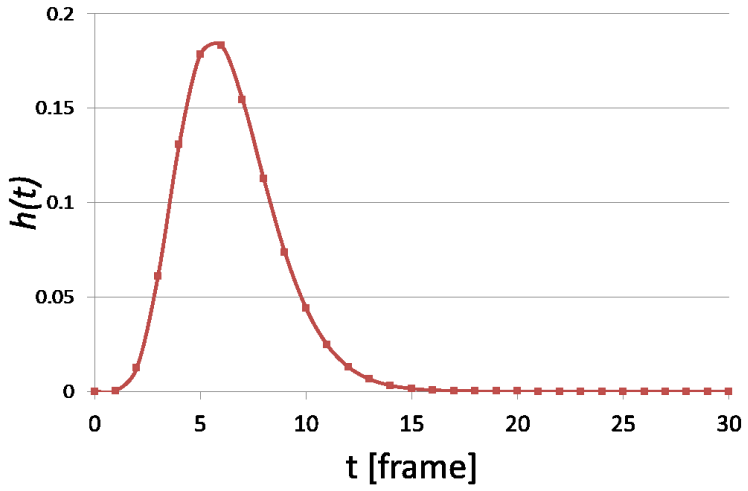


図 2.4.7: 遅れ系の伝達関数 ($\tau = 1, n = 8$)

時間微分のカーネルには式 2.4.12 を微分した次の関数を用いる．

$$\frac{\partial h(t)}{\partial t} = \mathcal{L}^{-1}\left[\frac{s}{\left(s+\frac{1}{\tau}\right)^n}\right] \tag{2.4.13}$$

$$= \frac{t^{n-2}}{\tau^{n+1}\Gamma(n)}\{\tau(n-1)-t\}e^{-\frac{t}{\tau}} \tag{2.4.14}$$

微分された関数も同様に図 2.4.8 のようになる．

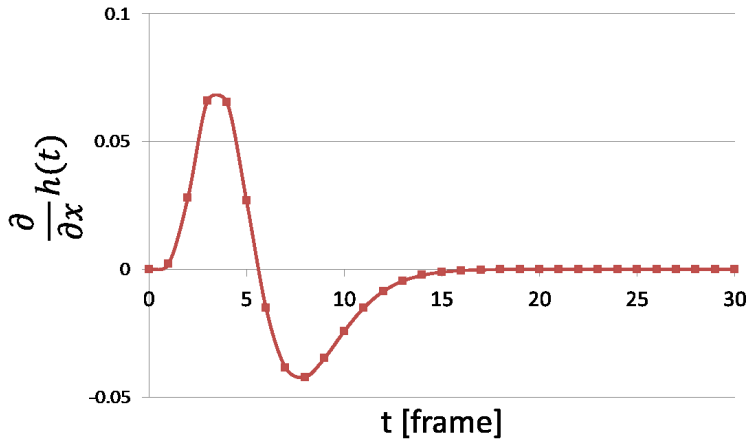


図 2.4.8: t 方向に微分した遅れ系の伝達関数 ($\tau = 1, n = 8$)

2.5 オプティカルフロー

オプティカルフロー (以降では，OF と略称する) とは画像から観測される物体の動きをベクトルで表したものである．本研究では，工学的な OF の推定法である Lucas-Kanade 法 [14] を用いる．

Lucas-Kanade 法は，OF 拘束方程式と付加条件から OF の推定を行うことで算出する手法である．

2.5.1 OF 拘束方程式

まず，入力動画情報を $I(x, y, t)$，OF ベクトルの x 成分，y 成分をそれぞれ $u(x, y)$,$v(x, y)$ と仮定する．すると，時刻 t における輝度は時刻 $t + \delta t$ の点 $(x + u\delta t, y + v\delta t)$ における輝度と等しいと期待できる．このことから，下式 2.5.1 が小さい時間間隔 δt に対して成り立つ．

$$I(x + u\delta t, y + v\delta t, t + \delta t) = I(x, y, t) \tag{2.5.1}$$

もし $I(x, y, t)$ が x, y, t に対して滑らかに変化するならば，式 2.5.1 の左辺をテイラー級数展開しすることによって，以下のような式を導くことができる．

$$I(x, y, t) + \frac{\partial I}{\partial x}u\delta t + \frac{\partial I}{\partial y}v\delta t + \frac{\partial I}{\partial t}\delta t = I(x, y, t) \tag{2.5.2}$$

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \tag{2.5.3}$$

式 2.5.3 は，OF 拘束方程式と呼ばれる．しかし，1 画素につきこの拘束方程式 1 つでは，OF ベクトルの 2 つの未知数 (u, v) を一意に決定することはできない．そのため，Lucas-Kanade 法では，次の項で説明するような負荷条件を設け，フロー推定を行う．

以降では簡単のため，$\frac{\partial I}{\partial x}$，$\frac{\partial I}{\partial y}$，$\frac{\partial I}{\partial t}$ をそれぞれ I_x，I_y，I_t と表記する．

2.5.2 付加条件

以下のように注目画素近傍の各画素 $q_1, q_2, ..., q_n$ について複数の拘束方程式を立てる．

$$\begin{aligned}
I_x(q_1)u + I_y(q_1)v + I_t(q_1) &= 0 \\
I_x(q_2)u + I_y(q_2)v + I_t(q_2) &= 0 \\
&\vdots \\
I_x(q_n)u + I_y(q_n)v + I_t(q_n) &= 0
\end{aligned}$$

式 2.5.4 は，$Av = b$ の形で行列式化ができる．

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, v = \begin{bmatrix} u \\ v \end{bmatrix}, b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix}$$

2 つの未知数 u, v に対して，未知数の数を超える方程式が存在するため，最小二乗法を用いて u, v の最適解を求める．これは，以下のような 2×2 システムの解を求めることに相当する．

$$A^T Av = A^T b \tag{2.5.4}$$

$$v = (A^T A)^{-1} A^T b \tag{2.5.5}$$

先ほどの A, v, b の定義より，u, v は以下の式で求められる．

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_x(q_i)I_y(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i)I_t(q_i) \\ -\sum_i I_y(q_i)I_t(q_i) \end{bmatrix} \tag{2.5.6}$$

ここで，各総和の計算時に注目画素からの距離に応じた重み係数を掛けることでさらに良い結果が得られる．式 2.5.6 は重み係数を w_i と置くと，以下のようになる．

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i w_i I_x(q_i)^2 & \sum_i w_i I_x(q_i)I_y(q_i) \\ \sum_i w_i I_x(q_i)I_y(q_i) & \sum_i w_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i w_i I_x(q_i)I_t(q_i) \\ -\sum_i w_i I_y(q_i)I_t(q_i) \end{bmatrix} \tag{2.5.7}$$

重み係数は，注目画素との距離からガウス関数により求められる値を用いることができる．このような重み係数を用いた積和処理は，一種の畳み込み計算として処理することができる．

2.6　シミュレーションの処理の流れ

畳み込み処理から OF 計算までの流れを図 2.6.1 にて示す．視覚神経系のシミュレーションは，このように計算量の大きな畳み込み計算を複数回実行することで実現できる．当然，計算量は膨大になるため並列処理技術は必要不可欠である．

実装の詳細については後の章で図を再掲の上で説明する．

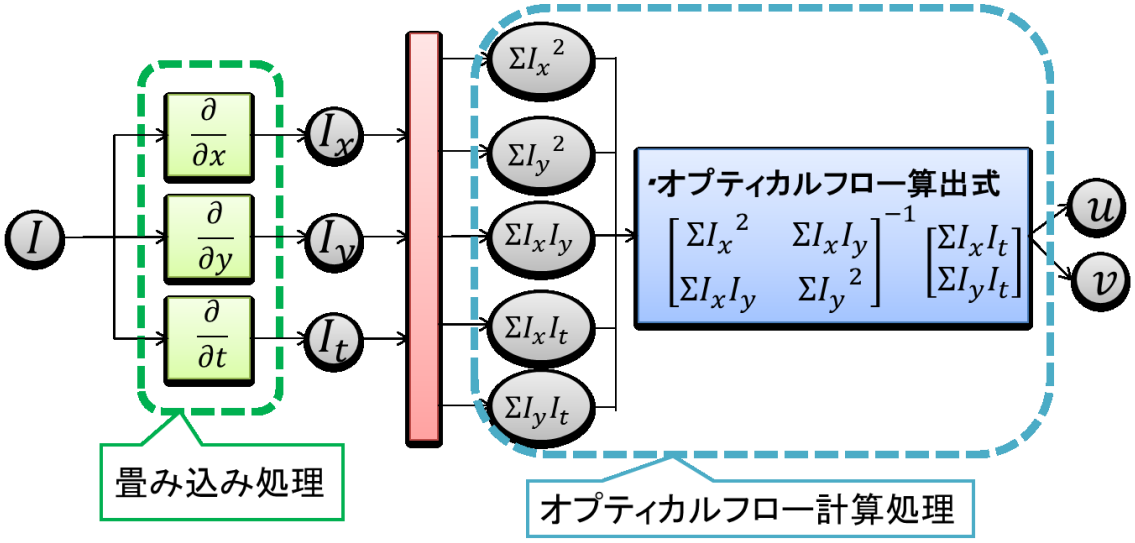


図 2.6.1: シミュレーションの処理の流れ

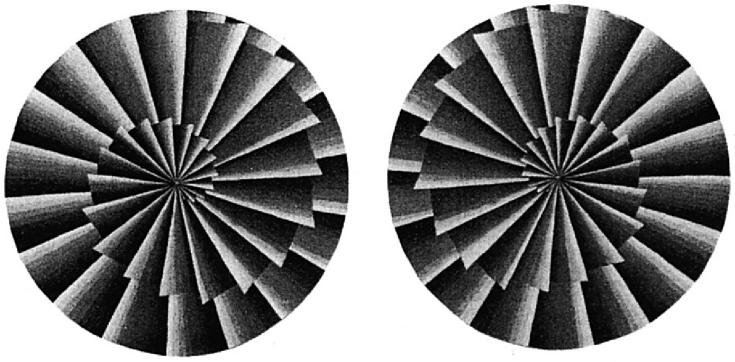


図 2.7.1: フレーザー・ウィルコックス錯視．図 1.1.1 の rotating snake と呼ばれる錯視もこの錯視画像を元にしたものである [2].

2.7 中心視周辺視

中心視周辺視は，視界の中心では明瞭に認識をし，周辺部になるに従って不明瞭になってゆく性質のことである [15]．これは視覚神経系において，中心部分を担当する神経細胞が多く，周辺部分を担当する神経細胞は少なくなることが大きな要因と考えられている．錯視現象においても中心視周辺視は大きく影響している．図 1.1.1 や図 2.7.1 も周辺視で視認する方の方が錯視量が大きいことが知られている．

中心視周辺視の現象を再現するには，数理モデルとして空間方向のカーネルを修正する必要がある．具体的には，周辺の σ は大きくし，中心の σ は小さくするように各画素位置別に異なるカーネルを設定する必要がある．

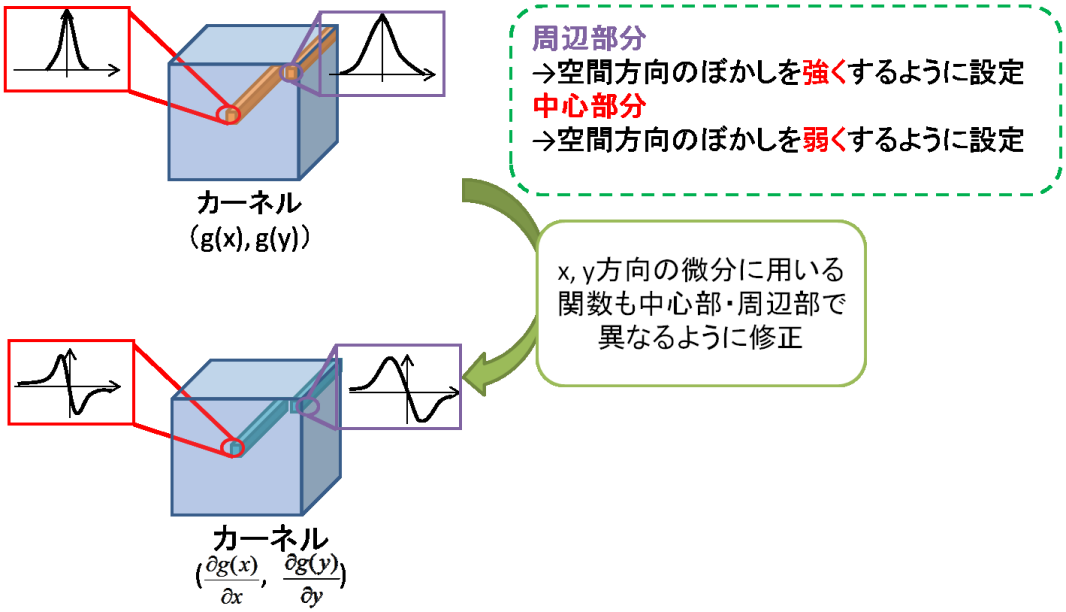


図 2.7.2: 中心視周辺視適用時のカーネルの設定法

第3章　錯視シミュレーションシステムの実装内容と高速化手法

本章では，錯視シミュレーションシステムの実装内容と GPU 搭載 PC クラスタを用いて高速化した手法についてまとめる．なお，シミュレーションの対象となる数理モデルとしては．前章で紹介した OF のモデルを想定する．

3.1　錯視シミュレーションシステムの概要

まずは，錯視シミュレーションシステムはどのような流れで処理を行なっているかについて説明する．

本研究は，参考文献 [12] の実装を基本としており，GPU 搭載 PC クラスタに特化しているのが特徴である．この基本的な実装を以降では，baseline を呼ぶこととする．

3.1.1　基本的な実装 (baseline)

baseline は以下の流れで行う．

Step.1 画像データの分配 (複数ノードのみ)

本システムは画像データを入力データとし，一番最初の段階として，複数ノード実行の場合，画像データを root ノードから他ノードに分配する必要がある．分配するデータは，図 3.1.1 の様に画像領域的に分割した部分である．以降，ノード間通信は MPI 通信にて行う．

Step.2 入力動画生成

次に，画像データから固視微動を再現した動画データを生成する (図 2.4.2 参照)．画像データは 1.1.1 などの錯視画像を想定している．

Step.3 畳み込み処理

生成した動画データと前述した 3 次元カーネルを用いて 3 次元畳み込み処理を実行する．畳み込み処理は GPU により高速に演算をする．

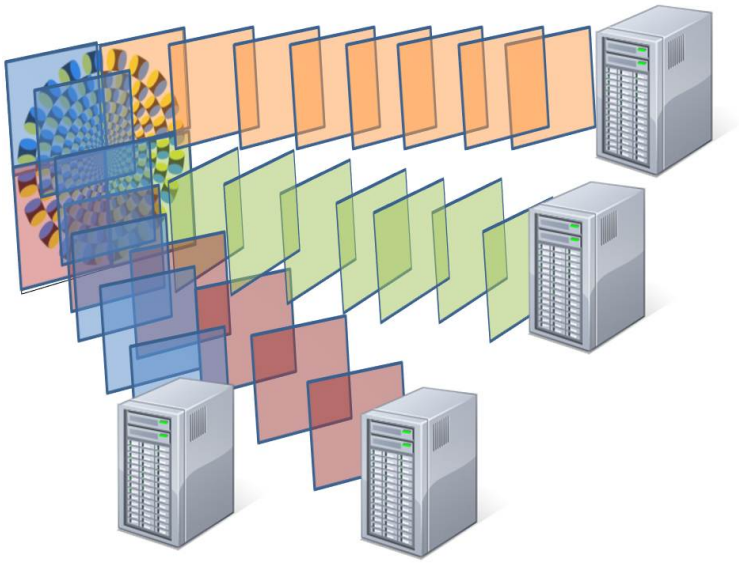


図 3.1.1: 画像領域別に分割したデータを各ノードに分配する．

Step.4 袖領域の交換 (複数ノードのみ)

各ノードは OF 計算のために自身の担当領域よりも外部のデータが必要になる．そのため，隣接するノードから外周部分のデータを交換する．

Step.5 OF 計算

OF 計算も GPU にて高速に演算する．

Step.6 Gather 処理 (複数ノードのみ)

OF 計算の演算結果を root ノードに集約する．これにより全画像領域分の錯視シミュレーション結果が生成される．

以上の流れで，錯視シミュレーションを行なっている．最も処理時間がかかる処理は畳み込み処理である．この処理が最大で全体の処理時間の 9 割以上を占める (入力画像：720x720，ノード数: 1)．なぜ，これほどまで畳み込み処理の時間がかかるのか．それは 3 次元カーネルのサイズに原因がある．

本システムで採用しているカーネルのサイズは $31 \times 31 \times 30$ である．通常，OpenCV などの OF 計算で使用されるカーネルが $3 \times 3 \times 2$ であるので，このようなカーネルと比較すると約 1600 倍も要素数が増加している．このカーネルのサイズはシステム全体の計算量にも大きく影響している．

下式 3.1.1 は 1 画素当たりの本システムの計算量を表したものである．

$$\underbrace{\{4 \times (31 \times 31 \times 30)\} \times 3}_{\text{畳み込み処理の計算量}} + \underbrace{\{5 \times (15 \times 15) + 13\} \times 2}_{\text{OF 計算の計算量}} = \quad 348,236 \tag{3.1.1}$$

例えば，入力画像が 720×720 の場合，1800 億回もの浮動小数点数演算を要する．特に畳み込み処理は全体の計算量の 99%以上を占めている．よって，高速化にあたってはこの畳み込み処理を高速化することに重点を置いている．

以降では，高速化案について述べてる．

3.1.2 画像データのみの転送による高速化 (gpu_eye_sim)

畳み込み処理の高速化の前に初期化処理の高速化について着目した．baseline では，2.4.1 のように入力画像から動画データの生成をした後に畳み込み処理を行うのだが，この動画生成プロセスを省略して畳み込み処理を実行する手法を考案した．

その方法としては，図 3.1.3 にもあるように畳み込み処理の時に ROI [1] を移動させて画素値を取得し畳み込み処理を行うという方法である．また，ROI 移動時に画素値がない領域に移動する恐れがあるので，入力画像は 3×3 に拡張する (図 3.1.2 参照)．本高速化案は，入力画像から GPU で動画データを生成するので，以降では本高速化案を gpu_eye_sim と呼ぶ．

baseline と比較すると主に変更した部分は，入力動画生成と畳み込み処理に 2 点である．

baseline では動画データを CPU から GPU に転送した後に畳み込み処理を行うが，gpu_eye_sim では入力画像のみでよい．そのため，動画データ入力の省略による高速化が見込まれる．

3.1.3 3 次元畳み込み処理を 2 次元畳み込み処理に一括化 (packed_conv)

図 3.1.4 にもあるように錯視シミュレーションシステムの畳み込み処理にはいくつかの特徴が存在する．

- 入力動画の各フレームデータは単純に x 方向にシフトしただけの画像データ
- 3 次元畳み込み処理は 2 次元畳み込み処理の集合

この特徴を考慮に入れると，3 次元カーネルデータは 2 次元データに一括化ができるのではないかと考えた．図 3.1.4 はその一括化を図示したものである．緑の部分がカーネル，赤の部分が入力データである．

[1] ROI ··· 英語では，“Region of Interest” と呼称される．画像処理において処理をしたい特定の領域のことを指す．

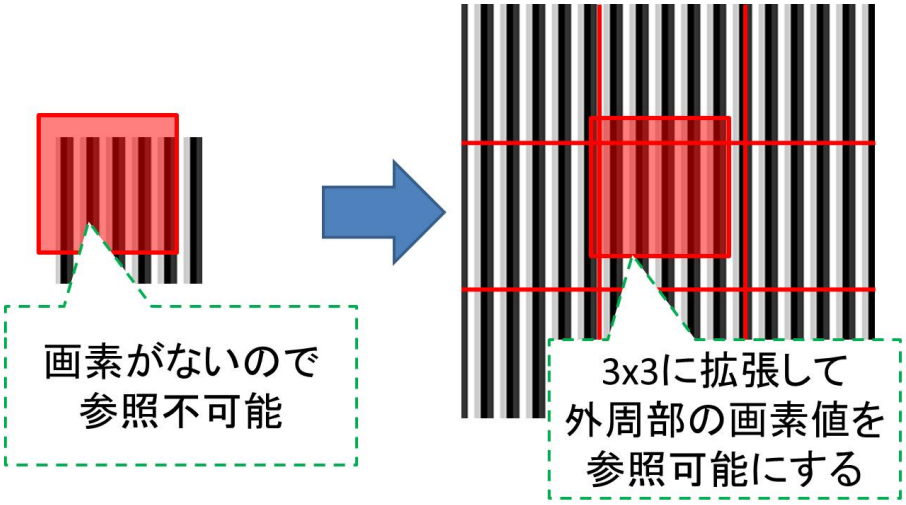


図 3.1.2: 外周部分の画素値が取得できるように入力画像を 3 × 3 の領域に拡張する

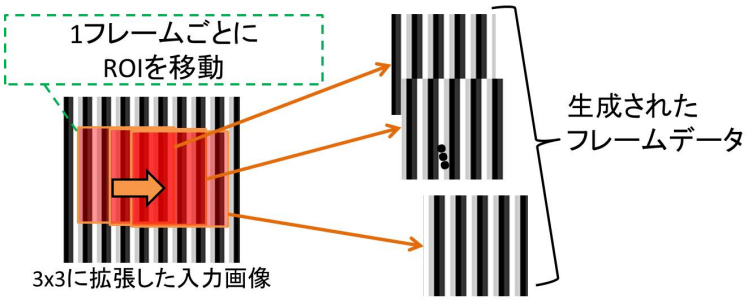


図 3.1.3: ROI を移動させることで固視微動を再現したフレーム情報を生成し、畳込み処理を行う

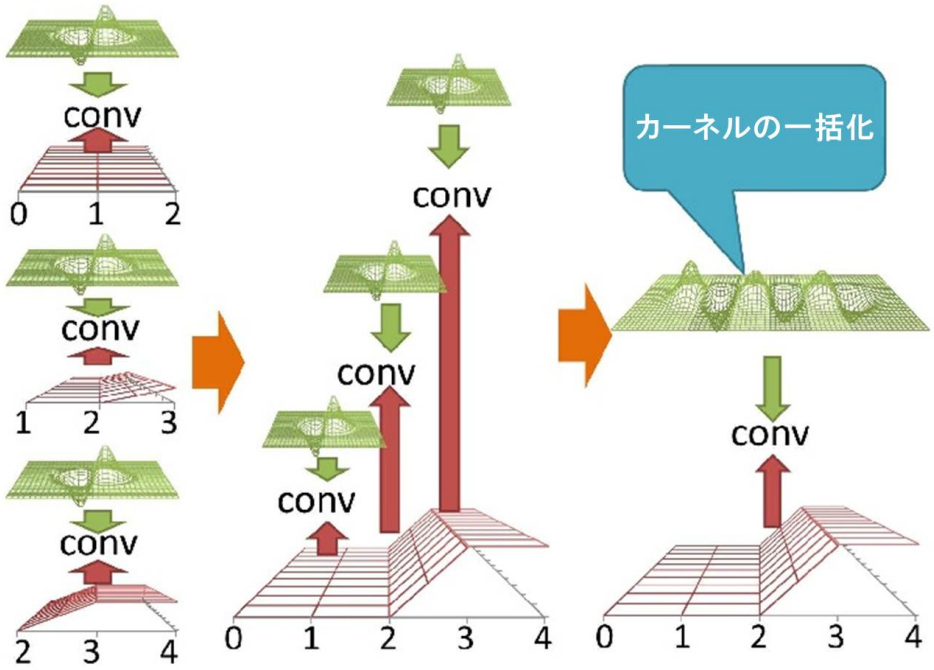


図 3.1.4: 3 次元畳み込み処理を 2 次元畳み込み処理に一括化

図の左側部分は 3 次元畳み込みを 2 次元の畳み込みに分解していることを表している．また，中間部分では，これはカーネルを移動させた処理に置き換えられることを示している．さらに，右側部分ではこの移動させたカーネルは一括化できることを表している．これにより，入力画像と 2 次元カーネルデータの 2 次元畳み込み処理に一括化が可能となる．

表 3.1 は，baseline と本高速化案の計算量の比較である．畳み込み処理の一括化により，約 **1/10** にまで計算量を削減することができる．

3.1.4 袖領域の交換処理の省略 (no_exchange)

packed_conv により畳み込み処理の計算量が減少すると，相対的に通信処理の処理時間が顕在化する．Gather 処理は OpenMPI ですでに最適化されている．よって，さらなる高速化案として袖領域の交換処理の省略を考案した．本高速化案は袖領域の部分を自前で計算してしまうことで，交換処理の省略を実現している．

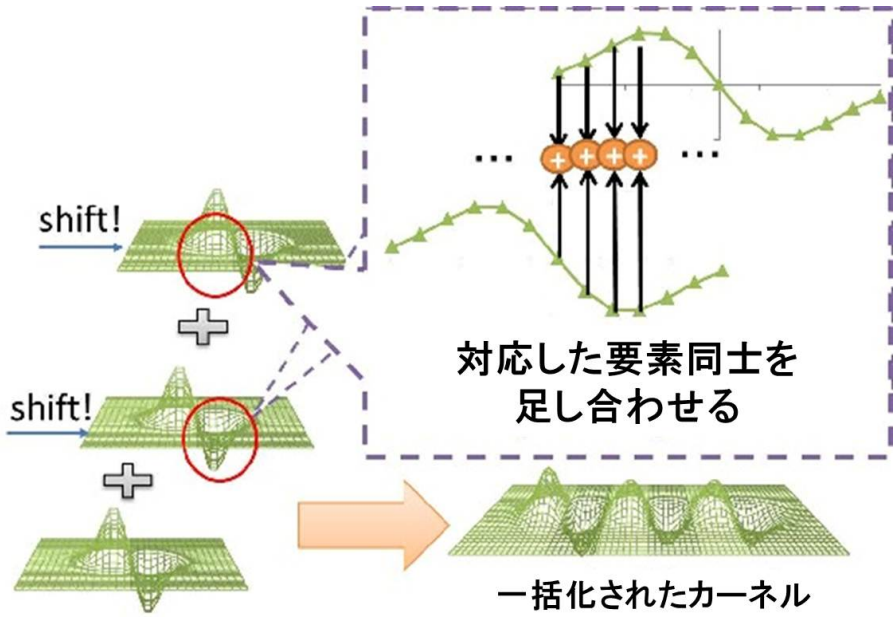


図 3.1.5: 2 次元化したカーネルの詳細

表 3.1: 通常の OF 計算と本研究の OF 計算の浮動小数点数計算回数の比較

	浮動小数点計算の回数 [2^{30} 回]	
入力画像の解像度	baseline	packed_conv
240x240	181	19
480x480	727	74
720x720	1635	168

第4章　錯視シミュレーションの検証実験

本章では，実装した錯視シミュレーションシステムが本当に錯視現象を再現できているか検証する．

4.1　錯視シミュレーションに最適なパラメータの探索

2章でも述べたように，錯視シミュレーションの数理モデルでは様々なパラメータが存在する．その中でも錯視シミュレーションに最適なパラメータを選出するパラメータ調整を行う．パラメータ調整には各種パラメータの全パターンのシミュレーションを試行するという方法を採る．そして，各シミュレーション結果の中でも錯視現象が最も顕著に再現できたものを最適なパラメータとして選出する．

なお，パラメータ調整の対象となるパラメータは以下の点である．

- σ ··· 空間方向のカーネルの数値に影響するパラメータ．空間ボケの強度に関係する．
- τ ··· 時間方向のカーネルの数値に影響するパラメータ．何フレーム目の影響を最大にするかに関係する．
- ϵ ··· OF計算でゼロ除算を回避するために使用するパラメータ [16](式2.5.7参照)．
- dx ··· 固視微動の移動量を表すパラメータ．入力画像から動画を生成する場合，1フレームごとに x 方向に移動する移動量．

4.1.1　パラメータ調整のためのシミュレーションの全体像

シミュレーションの流れを以下に列挙した．

Step1 OF計算の簡易テスト

シミュレーションにあたって，錯視・非錯視画像だけでなく図4.1.1のように単純な画像も用いて簡易テストも行った．この簡易テストを通過できないものはStep2以降に進めない．これにより，通常のOF計算が正常に行われているかを検証する．この段階で，全パターン中70%まで候補を絞り込むことができる．

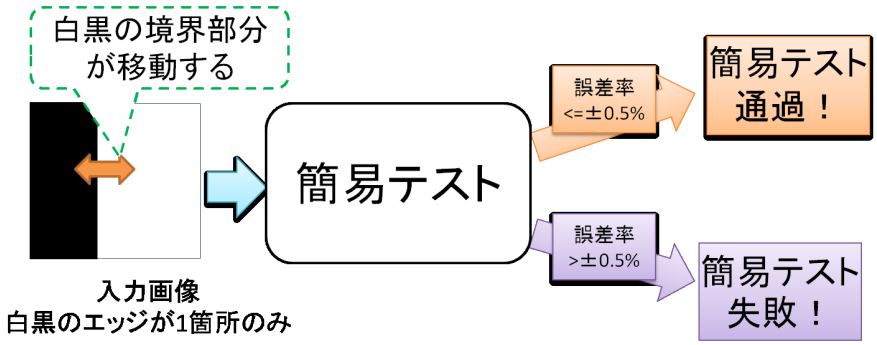


図 4.1.1: 通常の OF 計算が正常に行われているかの簡易テスト

表 4.1: 錯視現象が最も再現できた錯視シミュレーションの結果

				OF のピーク値 [pixel/frame]			
τ	σ	ϵ	dx	V_{r+}	V_{n+}	V_{r-}	V_{n-}
0.7	4.6	0.001	5	2.24	4.35	-2.25	-4.35

注：表中の V_{r+} などは，各場合における OF のピーク値である．

Step2 錯視画像を用いた実験

錯視画像 (図 4.1.2) を入力画像としたシミュレーションを行う．固視微動を左方向，右方向両方のパターンで錯視シミュレーションを行う．OF のピーク値を結果とする．

Step3 非錯視画像を用いた実験

Step2 と同様．入力画像を非錯視画像 (図 4.1.3) として錯視シミュレーションを行う．

4.1.2 パラメータ調整後の錯視シミュレーションの結果

パラメータ調整後の錯視シミュレーションの結果を表 4.1.2 と図 4.1.4 に示す．

V_{r+}: OF のピーク値 (入力画像: 錯視画像, 固視微動の方向: 右方向)

V_{n+}: OF のピーク値 (入力画像: 非錯視画像, 固視微動の方向: 右方向)

V_{r-}: OF のピーク値 (入力画像: 錯視画像, 固視微動の方向: 左方向)

V_{n-}: OF のピーク値 (入力画像: 非錯視画像, 固視微動の方向: 左方向)

V_{eye+}: 固視微動の大きさ (右方向時、今回は +5 [pixel/frame])

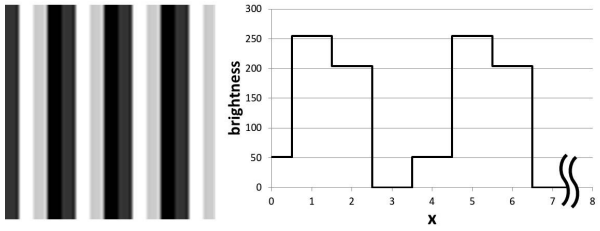


図 4.1.2: 錯視画像 (**rotating snake**).

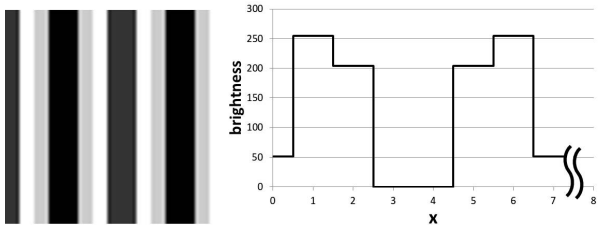


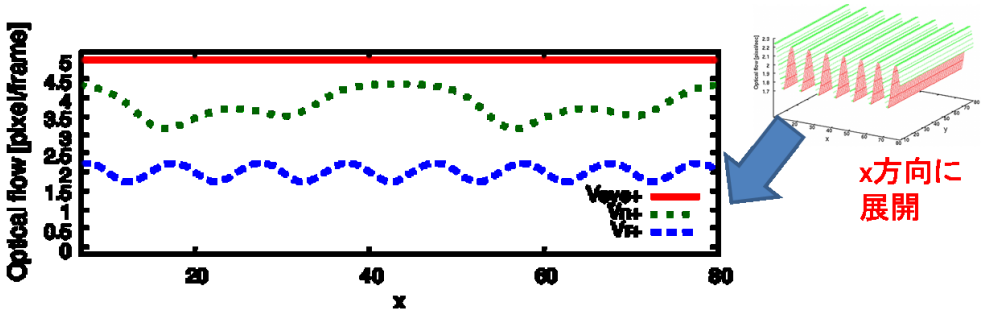
図 4.1.3: 非錯視画像 (**non-rotating snake**).

V_{eye-}**:** 固視微動の大きさ (左方向時、今回は -5 [pixel/frame])

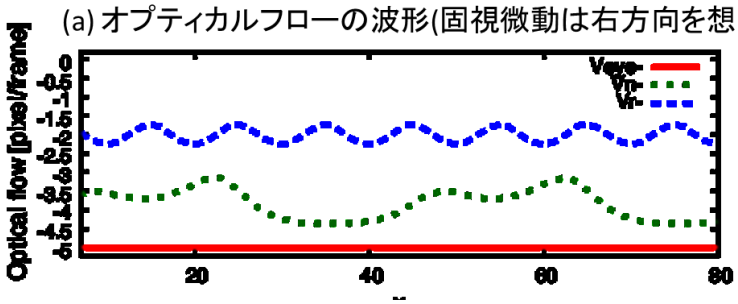
上記の結果で，OF の波形を x 方向にそって展開したグラフが図 4.1.4 である．

結果を見るとわかるように，V_{r+} と V_{r-} は大きく元の固視微動の移動量を下回っている．対して，V_{n+} と V_{n-} は固視微動の移動量にほぼ等しい結果となった．

錯視量は固視微動と検出した OF の差分で決まる．よって，この結果から錯視画像を用いた場合では，$V_{r+} = 2.76$ [pixel/frame] と $V_{r-} = -2.75$ [pixel/frame] もの錯視が発生したと言える．一方，非錯視画像を用いた場合だと，$V_{n+} = 0.65$ [pixel/frame] と $V_{n+} = -0.65$ [pixel/frame] であった．錯視画像の場合だと，非錯視画像と比較して 4 倍以上も錯視量が増加している．これらの結果は，人間が錯視・非錯視画像を見た場合と類似した結果であり，本錯視シミュレーションシステムで錯視現象を再現できたと結論づけられる．



(a) オプティカルフローの波形(固視微動は右方向を想定)



(b) オプティカルフローの波形(固視微動は左方向を想定)

図 4.1.4: 錯視・非錯視画像の OF 波形

第5章　錯視シミュレーションシステムの性能評価

本章では錯視シミュレーションシステムの性能評価について述べる．

本錯視シミュレーションシステムは，計 16 ノードの GPU 搭載 PC クラスタにて実装した．より詳細な環境は表 5.1 の通りである．

性能評価にあたって，3 種類の画像サイズにおける処理時間測定をした．また，処理時間の逆数をスループットとした (1 秒間あたりのシミュレーション回数).

5.1　baseline と gpu_eye_sim の性能比較

gpu_eye_sim は以下の 2 点で高速化すると予想していた．

- CPU から GPU へ動画ではなく，画像のみを転送する
- 画像のみを GPU に格納すればいいので，キャッシュがより有効に活用できる

図 5.2.1 に示すように，240×240 の場合でスループットが最大で 7.6%増加した．これは，畳み込み処理の時間が 0.032 秒から 0.030 秒になったことが大きな原因であることが判明した．キャッシュ効率の向上で畳み込み処理が高速化した可能性が考えられる．しかし，480×480 と 720×720 の場合では大きな速度向上は確認出来なかった．画像サイズが拡大したことにより，gpu_eye_sim の場合でもキャッシュ効率が悪化し，baseline と比較して速度向上が出来なかったためと考えられる．

5.2　gpu_eye_sim と packed_conv の性能比較

図 5.2.1 が示すように，packed_conv は前述した計算量の削減により大きな性能向上が見込まれていたが予想通りとなった．

処理時間は最大で，61 %(入力画像の解像度: 720×720，ノード数: 16) の削減となった．

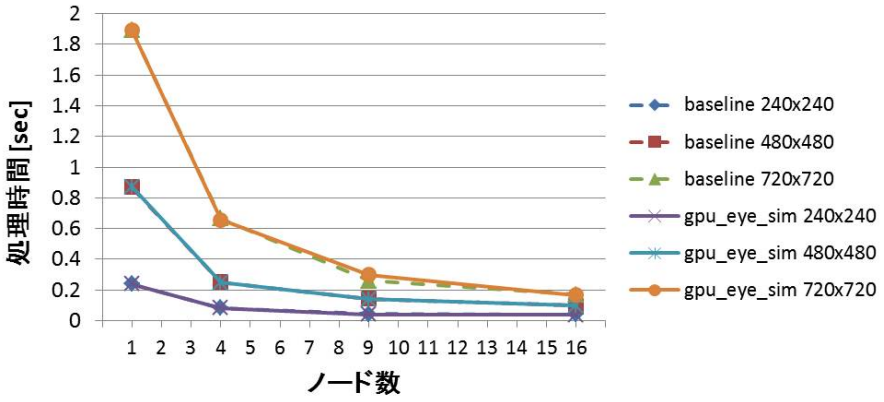
しかし，ノード数が 12 ノードから 16 ノードになる場合速度向上が鈍化することが確認された．これは，ノード数が増えたために計算時間が削減された効果よりも，交換処理や Gather 処理でノード間通信時間が増加したこと影響の方が大きかったためである．

表 5.1: ハードウェア・ソフトウェア環境

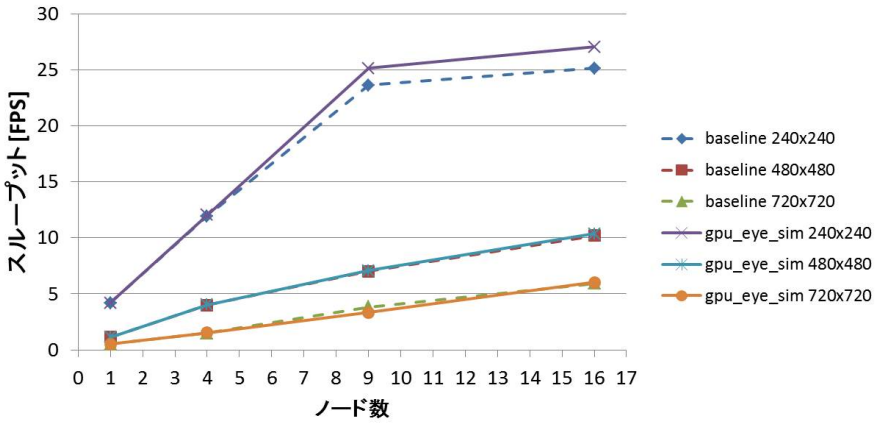
CPU	Intel Xeon Quad-Core CPU W3520
Clock speed	2.67 GHz
memory	6 GB
GPU	NVIDIA C1060 (GT200 architecture)
Clock speed	1.296 GHz
Number of Streaming Processor	240
Peak performance	933 GFLOPS
Memory	4 GB
Memory bandwidth	102 GB/sec
Graphics bus	PCI Express x16 Generation 2.0
OS	CentOS 5.3
C Compiler	Intel C compiler 11.1
CUDA	CUDA Toolkit 3.2

表 5.2: 各処理の処理時間の内訳 (入力画像の解像度: 720×720, ノード数: 16)

処理内容	*gpu_eye_sim*	*packed_conv*
畳み込み処理 [sec]	0.1404 (84.6 %)	0.0378 (59.2 %)
交換処理 [sec]	0.0025 (1.5 %)	0.0028 (4.4 %)
オプティカルフロー計算 [sec]	0.0007 (0.4 %)	0.0007 (1.0 %)
Gather 処理 [sec]	0.0223 (13.5 %)	0.0226 (35.4 %)

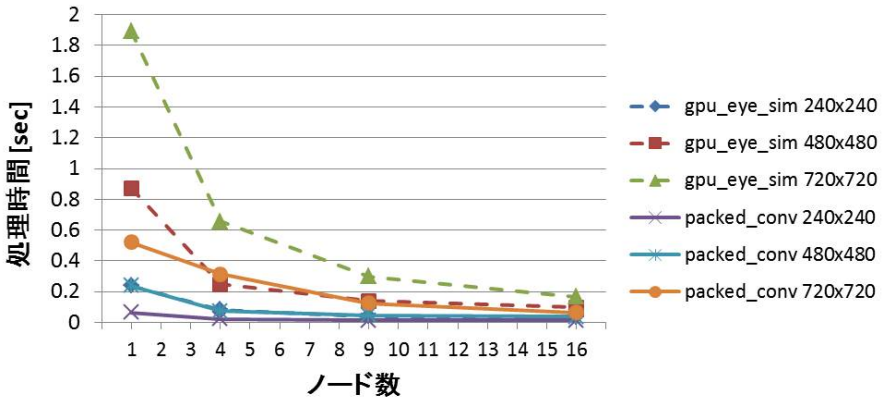


(a) 処理時間

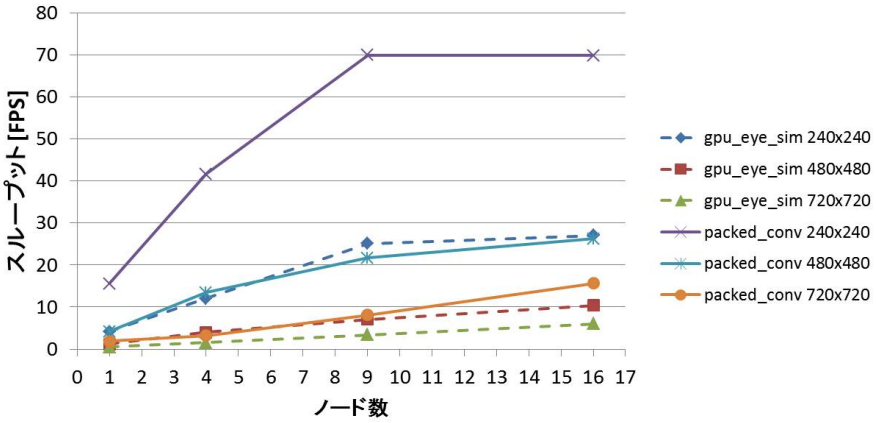


(b) スループット

図 5.1.1: baseline と gpu_eye_sim の性能比較

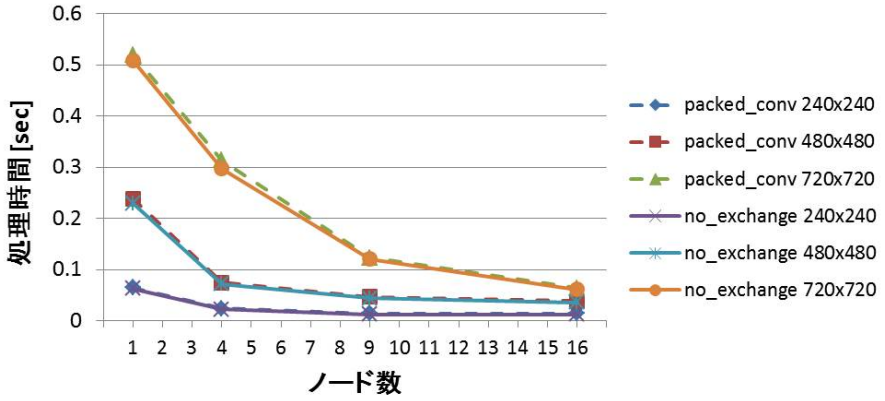


(a) 処理時間

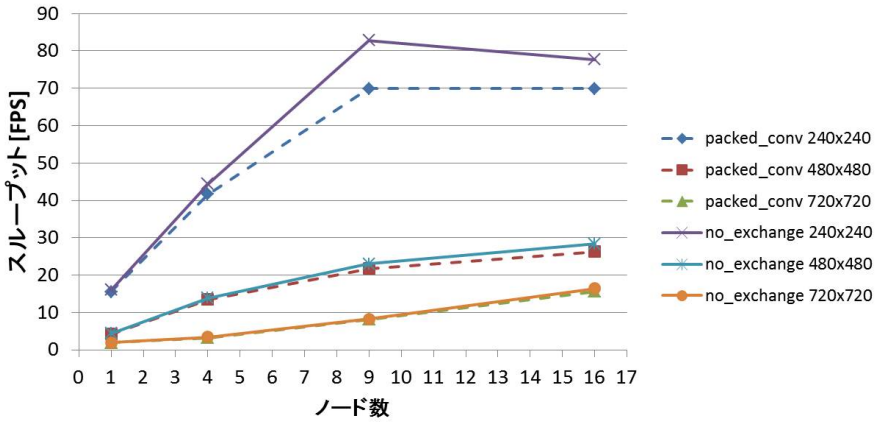


(b) スループット

図 5.2.1: gpu_eye_sim と packed_conv の性能比較



(a) 処理時間



(b) Throughput

図 5.3.1: packed_conv と no_exchange の性能比較

この結果の考察を確認するため，各処理の処理時間を計測した．表 5.2 に示すように，packed_conv で畳み込み処理の処理時間が大幅に削減されたことにより交換処理と gather 処理の処理時間が顕在化している．

5.3 packed_conv と no_exchange の性能比較

前章で多ノード時に通信処理の処理時間が問題となっていたので，袖領域の交換処理と省略した no_exchange の性能向上を予想したがその通りとなった．

図 5.3.1 は処理時間とスループットの性能比較の結果である．12 ノードから 16 ノードに変化するときの性能向上が改善した．さらに詳細を確認するため，図 5.3.2 にて性能向上率をグラフ化した．低解像度時の方が速度向上の比率が高く，高解像度になるに従って速度向上が限定的になるという傾向が確認できる．これは図 5.3.3 を見ても明らかで，交換処理の処理時間自体が高解像度

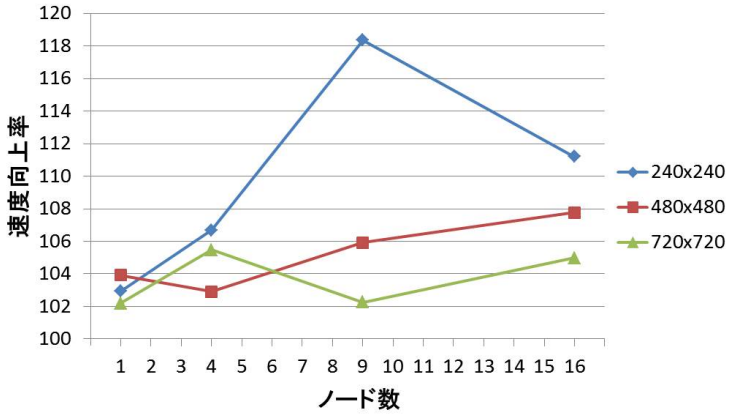


図 5.3.2: packed_conv から no_exchange の性能向上率

になるに従って減少するためで，交換処理を高速化させたことによる効果が限定的になるためである．

5.4 各高速化案に対しての処理時間の比較

図 5.4.1 は，各高速化案の処理時間をひとまとめにしたグラフである．図 5.4.1 で示すように，baseline→gpu_eye_sim→packed_conv→no_exchange と高速化をすすめるに従って処理時間が削減された．しかし，この速度向上が限定的になっており，表 5.2 の処理時間の内訳からもわかるように，今回の高速化で改善出来なかった Gather 処理がボトルネックとなっている．

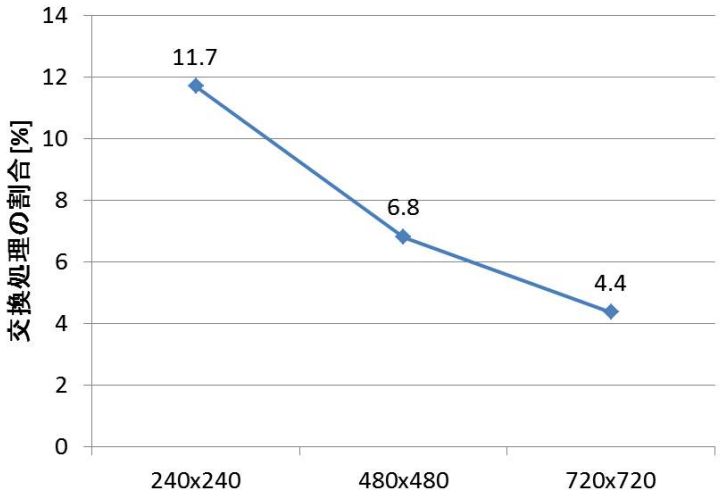


図 5.3.3: 全体の処理時間に対して，交換処理の処理時間の占める割合 (16 ノード時)

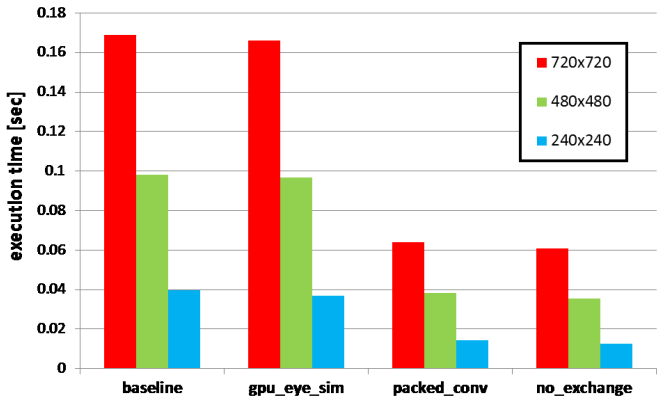


図 5.4.1: 各高速化案においての処理時間の比較 (ノード数: 16)

第6章　回転運動の錯視シミュレーション

前章までの錯視シミュレーションシステムは直線運動の固視微動しかサポートしていなかった．そこで，baseline を元として回転運動に対応した錯視シミュレーションを実装した．

6.1　回転運動対応化のための実装

回転運動対応化のため新らたな動画生成を考案した．baseline の実装時と同じように動画生成には 1 フレームごとに移動させたフレーム情報を生成する．この際，1 フレームごとに入力画像を回転させることで回転運動の入力動画の生成を行う．なお，回転中心は画像中心部分とする．

入力画像は図 6.2.1 と図 6.2.2 を用いる．図 6.2.1 は図 4.1.2 の錯視画像を円形に変形した画像である．図 6.2.2 も同様に図 4.1.3 を円形に変形した画像である．

6.2　回転運動の錯視シミュレーション

次に，錯視画像を回転させるような固視微動の錯視シミュレーションを行う．4 章の直線運動の固視微動の場合の錯視シミュレーションと同じような流れで，パラメーター調整を行う．

また，回転運動時には各画素位置ごとに移動速度が異なるので，図 6.2.3 のように決められた円周上の OF の精度をシミュレーション結果として採用する．例えば，図 6.2.3 内のオレンジの円周上での OF の大きさを抽出しピーク値を選定．その最大値から下式 6.2.1 を用いて OF の精度を算出する．

$$OF_{precision} = \frac{OF}{r \times \theta \times \frac{2\pi}{360}} = \frac{OF}{V_{circle}} \tag{6.2.1}$$

(θ: 回転速度 [degree/frame], r: 円周の半径 [pixel] OF: OF のピーク値 [pixel/frame], $OF_{precision}$: OF の精度, V_{circle}: 円周上の速度 [pixel/frame])

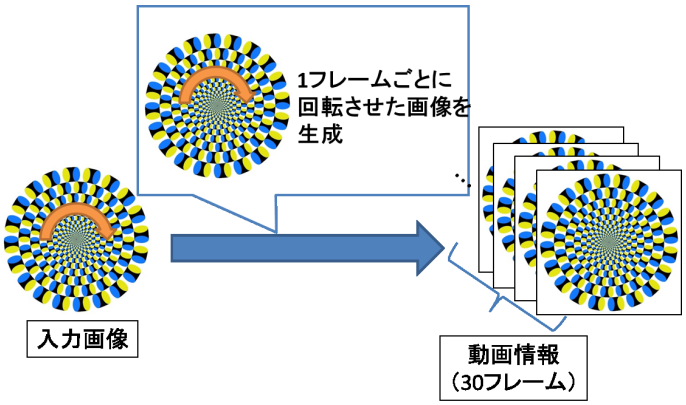


図 6.1.1: 回転運動の動画の生成法

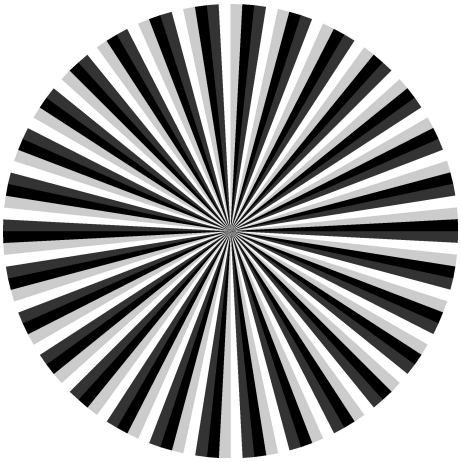


図 6.2.1: 円形に変形させた錯視画像

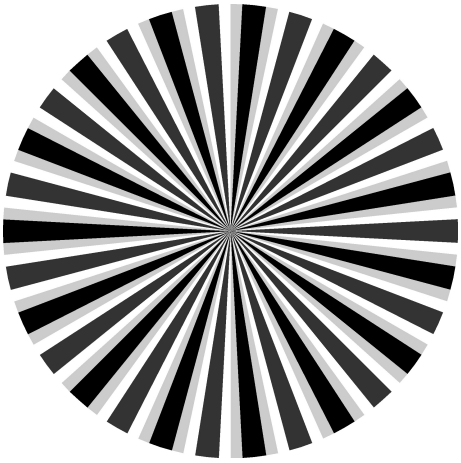


図 6.2.2: 円形に変形させた非錯視画像

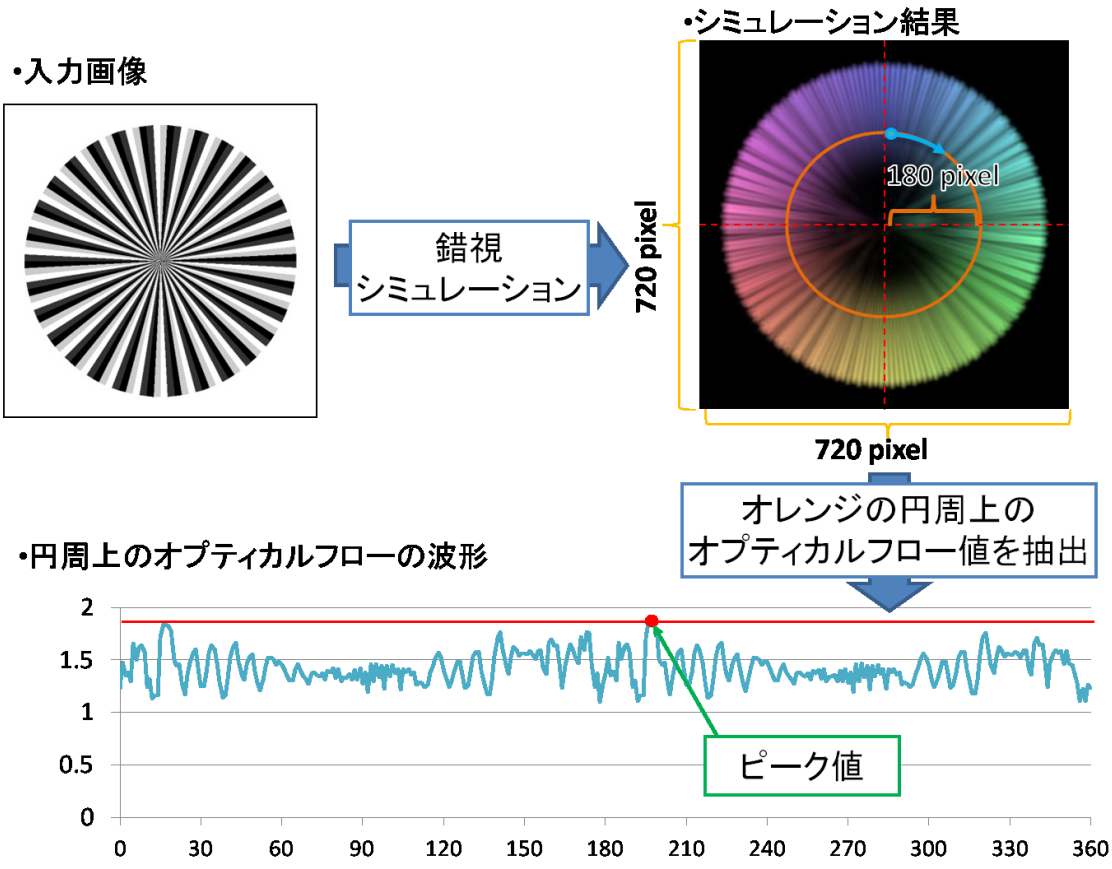


図 6.2.3: 特定の円周上から OF の大きさを抽出

表 6.1: 錯視現象が最も再現できた錯視シミュレーションの結果

				OF のピーク値[1] [pixel/frame] (精度)			
τ	σ	ϵ	θ [degree/frame]	V_{r+}	V_{n+}	V_{n-}	V_{r-}
0.8	2.5	0.001	1	1.87 (60 %)	2.87 (91 %)	1.86 (59 %)	2.88 (92 %)

6.3 回転運動の錯視シミュレーションの結果

パラメーター調整の結果，以下の錯視現象を再現できたと言えるようなシミュレーション結果が求められた.

図 4.1.4 の錯視シミュレーションと同じように円周方向に沿った形の OF の波形を示す.

非錯視画像の場合は精度が OF の精度がよく，錯視画像の場合は OF 値の精度が悪くなるような結果が確認できる．これにより，回転運動においても錯視シミュレーションを確認できた.

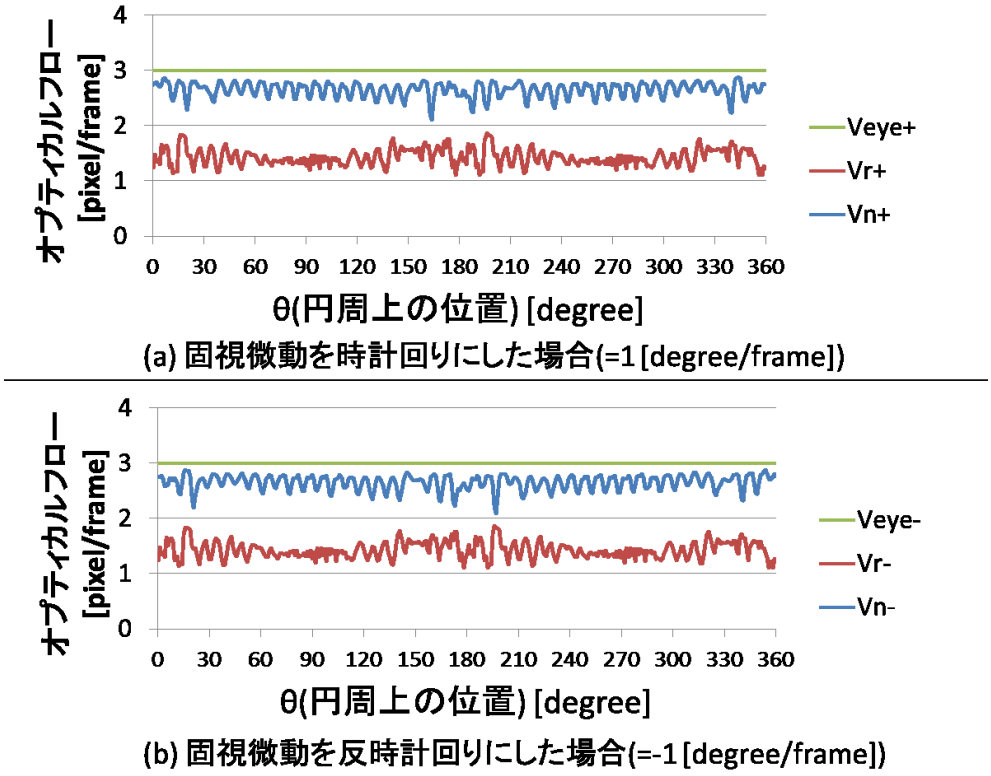

オプティカルフロー
[pixel/frame]
4
3
2
1
0
0 30 60 90 120 150 180 210 240 270 300 330 360
θ(円周上の位置) [degree]
Veye+
Vr+
Vn+
(a) 固視微動を時計回りにした場合(=1 [degree/frame])
オプティカルフロー
[pixel/frame]
4
3
2
1
0
0 30 60 90 120 150 180 210 240 270 300 330 360
θ(円周上の位置) [degree]
Veye-
Vr-
Vn-
(b) 固視微動を反時計回りにした場合(=-1 [degree/frame])

図 6.3.1: OF の波形.

第7章　中心視周辺視の概念を使用した錯視シミュレーション

この章では，中心視周辺視の概念を適用した錯視シミュレーションシステムの実装・高速化，そしてシミュレーション結果までを述べる．なお，本章での実装は，前章の回転を適用した実装が基礎となっている．

7.1　実装法

7.1.1　σ の設定法

2 章でも述べたように，中心視周辺視を適用するにあたって，カーネル関数各画素位置ごとに異なり中心の σ は小さく，周辺部分の σ は大きくなるように設定する必要がある．σ の設定法は，皮質拡大係数と呼ばれるものを基準にした．皮質拡大係数は，ある局所的な視野を担当する神経細胞の量の逆数である．この神経細胞の量により空間ボケの強度が決定するので，$CMP \propto \frac{1}{\sigma}$ と仮定し，図 7.1.1 の右側のように σ を設定する．

7.1.2　カーネルのサイズについて再考

中心視周辺視を適用するにあたって，問題となるのが巨大になったカーネルのサイズである．カーネルのサイズは，前章の実装システムまでは，$31 \times 31 \times 30$ であるのに対し，$91 \times 91 \times 30$ というサイズにまで拡大する．これは，σ を拡大したことにより図 7.1.2 のようにサンプリングをするためカーネルのサイズを拡大せざるをえないためである．

また，カーネルのサイズが拡大することにより，計算量も増大する．下式 7.1.1 のように，実に 8.5 倍となる．

$$\underbrace{\{4 \times (91 \times 91 \times 30)\} \times 3}_{\text{畳み込み処理の計算量}} + \underbrace{\{5 \times (15 \times 15) + 13\} \times 2}_{\text{OF 計算の計算量}} = \quad 2,983,436 \tag{7.1.1}$$

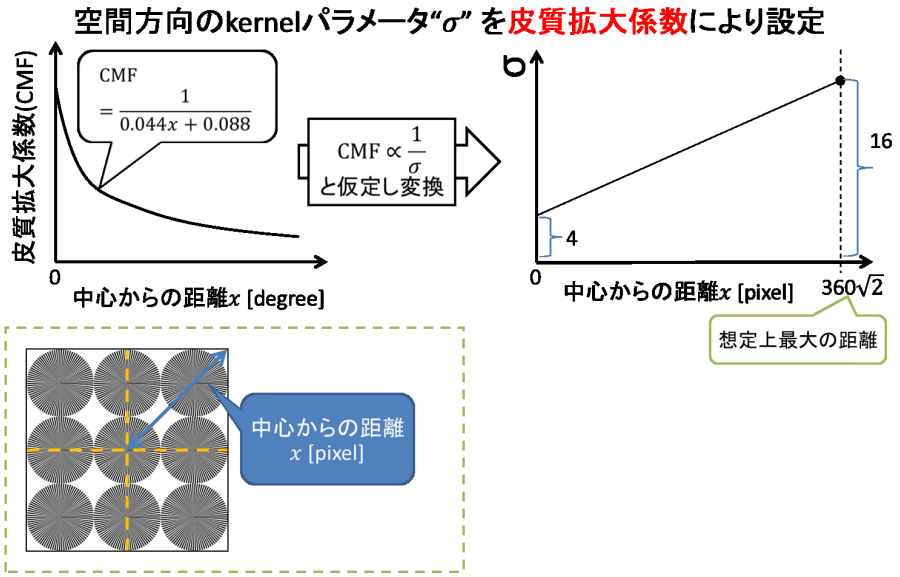


図 7.1.1: σ の設定法

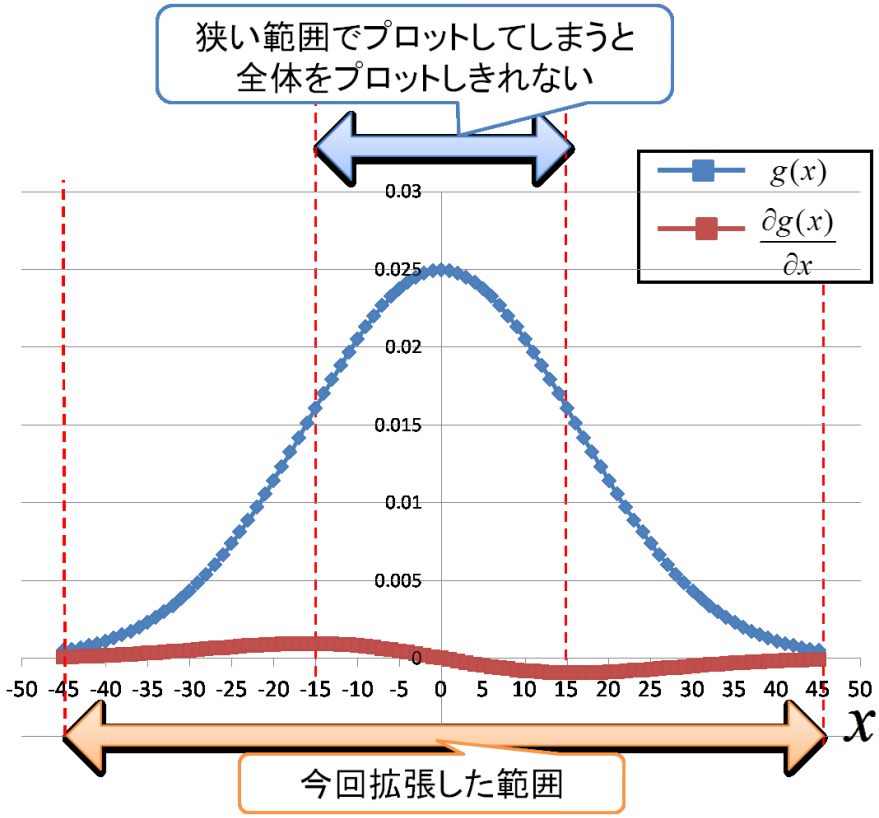


図 7.1.2: カーネルサイズが狭すぎると，カーネル関数全体をサンプリングできない．

処理時間は最大で 1.9 秒から 38.3 秒まで増大した (1 ノード，720×720 の画像を入力画像とした場合)．実に約 20 倍もの処理時間の増大となる，この増加量は何百何千とシミュレーションを行う錯視シミュレーションにおいては，致命的な差となる．例えば，1 時間で完了していたシミュレーションが 20 時間以上も要するものとなる可能性がある．

7.1.3 基本的な実装 (baseline_CP)

7.1.3.1 実装内容

実装の流れは，以下のようになる．カーネルの設定を大きなセクションとする．処理の流れは以下のようになる．今回変更した処理は，Step1 と Step2 の 2 点である．他の Step3 から Step5 の処理は baseline と同じ処理内容である．

Step1 カーネル設定 (Kernel)

1 画素 1 画素ごとのカーネルを σ に基づき設定をしていく．

Step2 畳み込み処理 (Conv)

み処理を行う．基本は baseline と同じだが，各画素位置ごとにカーネルデータを読み込む処理が追加される．

Step3 袖領域の交換処理 (Exchage)

Step4 OF 計算 (Optflow)

Step5 Gather 処理 (Gather 処理)

7.1.3.2 baseline_CP の性能評価

以上の実装で処理時間とスループット，さらには各処理の処理時間の内訳を測定した．処理時間は図 7.1.3，スループットは図 7.1.4，処理時間の内訳は図 7.1.5 で示す．

480×480 や 720×720 の場合では台数効果が確認できるが，240×240 の場合では，明らかに 12 ノードで速度向上が頭打ちとなってしまう結果となった．また，全体のパフォーマンスが 1 FPS 未満と以前の最大 80FPS を発揮していた no_exchange を考慮すると大幅に処理時間が増加したことがわかった．まずは，全体的な高速化が見込めるパイプライン化から実装した．

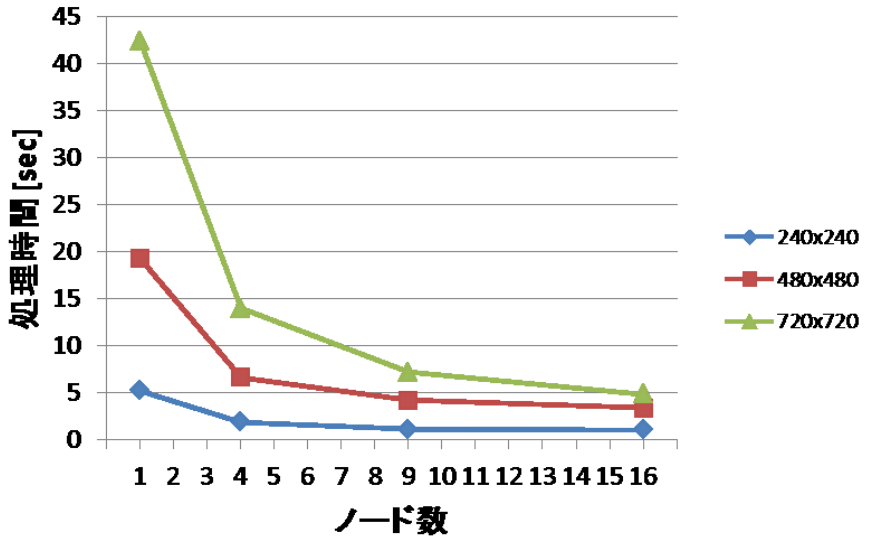


図 7.1.3: baseline_CP の処理時間

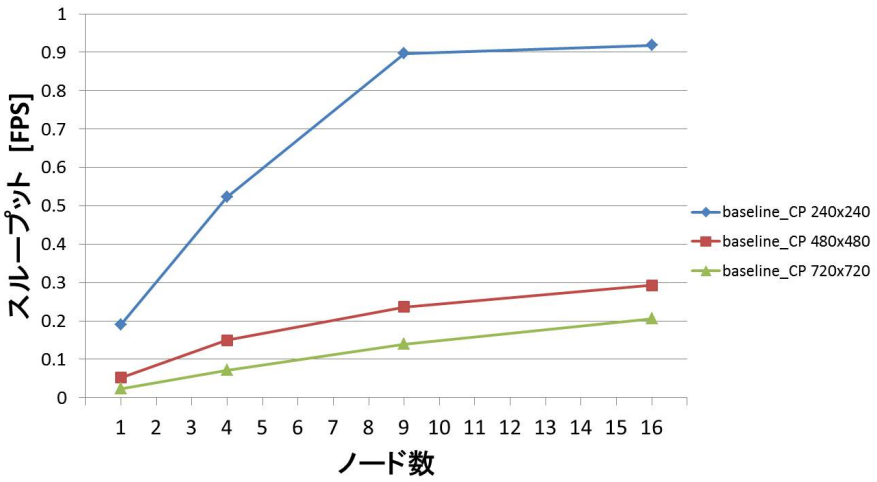


図 7.1.4: baseline_CP のスループット

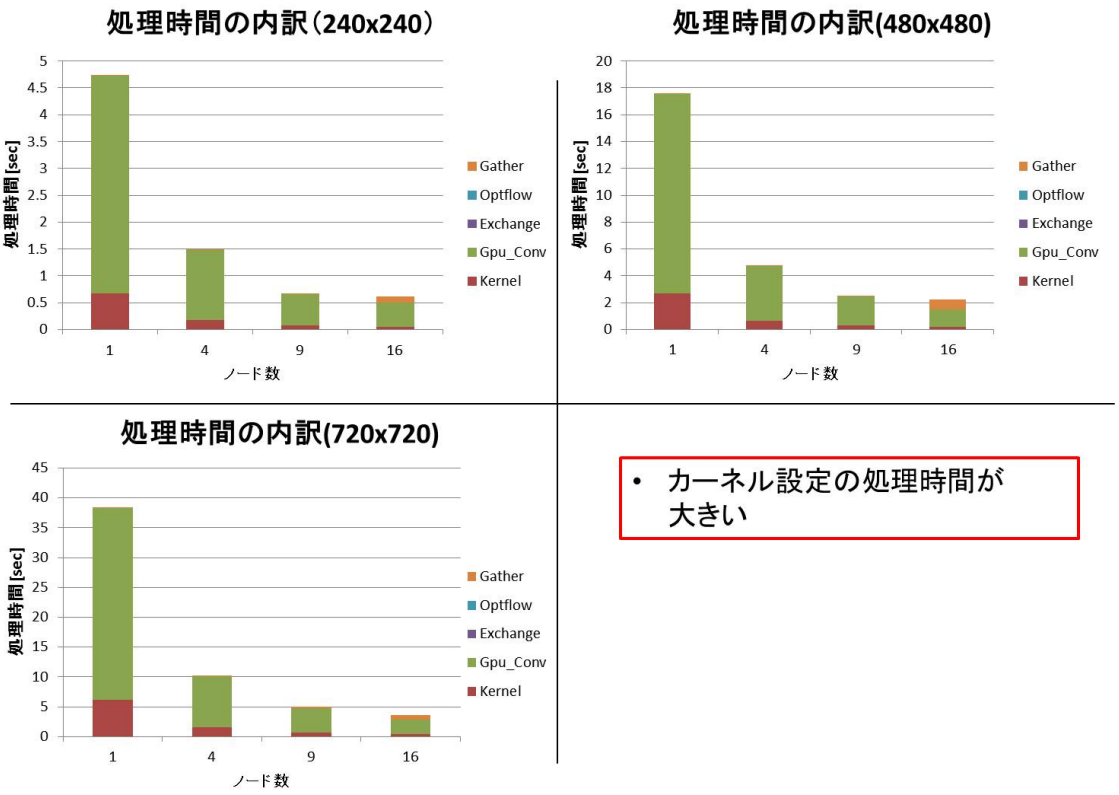


図 7.1.5: baseline_CP の各処理時間の内訳．ノード数が 12→16 ノードに増加するケースで，Gather 処理の処理時間が急増している．

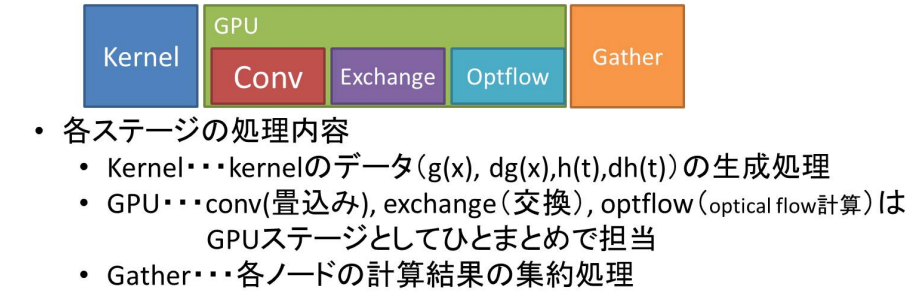


図 7.2.1: パイプライン処理の内容

7.2 高速化案

7.2.1 パイプライン化 (pipeline)

図 7.1.5 の示すように，畳み込み処理の次に処理時間がかかる処理はカーネルの設定であることが判明した．よって，パイプライン処理にて CPU ではカーネルの設定，MPI 通信を行い，GPU では畳み込み処理や OF 計算を行うことでオーバーラップ化を図ることとした．

7.2.1.1 実装内容と高速化の狙い

パイプライン化は図 7.2.1 のように，これまで逐次的に行なっていた処理を大きく分けて 3 ステージに分割し，各ステージのオーバーラップを図ることで高速化を狙った実装法である．特に今回は，カーネル処理がボトルネックとなっているので，カーネル処理と GPU 処理のオーバーラップを期待している．

- Stage1 カーネル設定 (Kernel)
- Stage2 GPU 処理 (Conv)
- Stage3 Gather 処理 (Gather)

7.2.2 pipeline の性能評価

図 7.2.2 で分かるに，パイプライン化で全体的に処理速度が向上した．しかし，12 ノードから 16 ノードに増加するときなどに台数効果が確認できなくなってしまった．

この問題の原因は，図 7.1.5 に示されているように Gather 処理の処理時間が 16 ノード時で増加することが原因である．

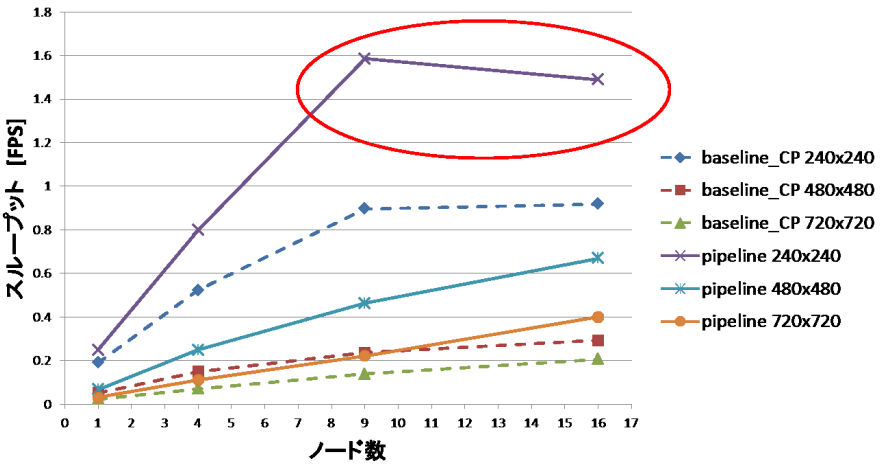


図 7.2.2: baseline_CP のスループット．赤い丸枠の箇所で速度向上できなくなっている

7.2.3 Gather 処理の最適化 (2step_gather)

pipeline 処理で Gather 処理が問題となることが判明したので，Gather 処理の最適化案を考案した．

7.2.3.1 実装内容と高速化の狙い

そもそも，16 ノード時で Gather 処理の処理時間が増大してしまうのは，通信衝突によるものだと予想される．よって，通信衝突を回避するため通信回数を低減するような方式を採用した．

図 7.2.3 で示すように，今回の Gather 処理では通信を 2 段階に行うように変更した．1 段階目は横方向のノード同士での小規模の Gather 処理を，2 段階目は縦方向のノード同士で小規模の Gather 処理を行うことで，通信回数の削減を図った．

7.2.3.2 2step_gather の性能評価

Gather 処理を高速化した効果で 16 ノード時でも台数効果を確認することができた．最終的には 240×240 時で 2.6FPS を達成することに成功した．

7.3 中心視周辺視適用の効果の確認

中心視周辺視適用により，中心視周辺視を適用しなかった場合に比べてどのように錯視シミュレーションが変化したかを確認する．

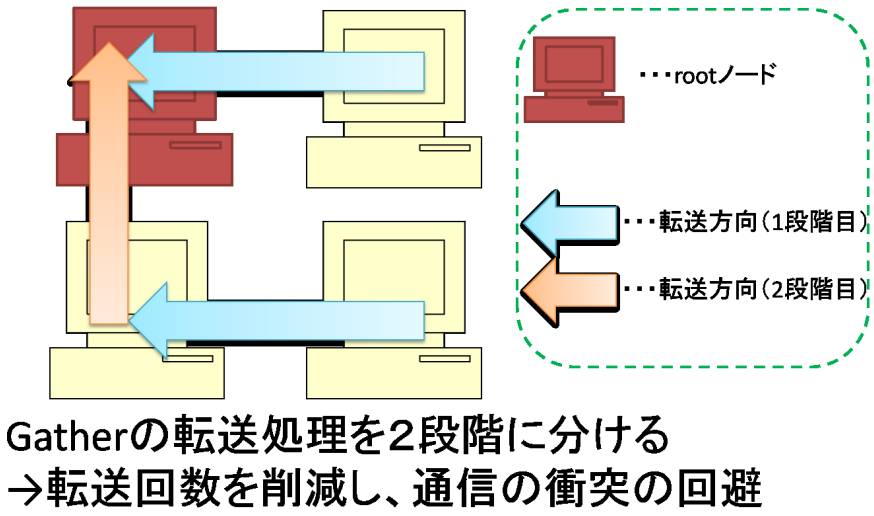


図 7.2.3: Gather 処理の最適化．転送処理を 2 段階に分けることで，通信回数の削減と通信衝突の回避を図っている．

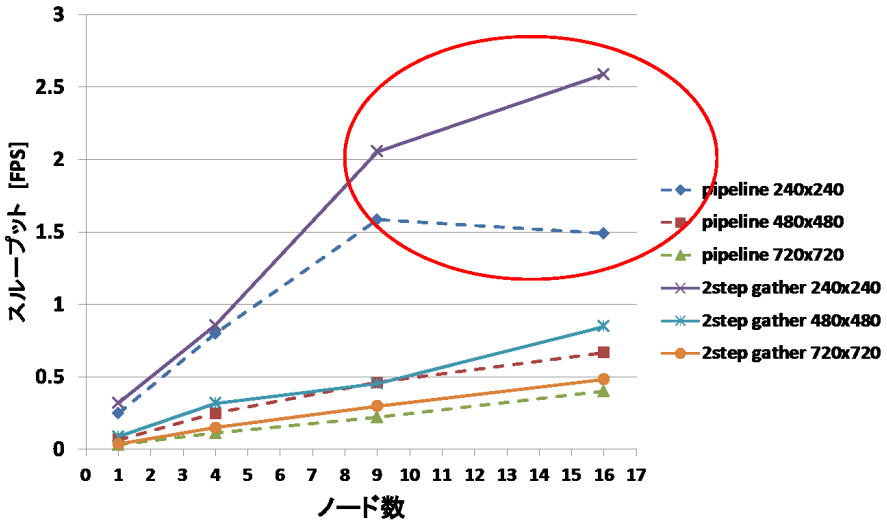


図 7.2.4: Gather 処理の最適化した結果．スループットが大幅に向上した．

表 7.1: 比較実験 1 に用いるパラメーター

τ	σ	ϵ	dx
0.7	4.0	0.001	5

7.3.1 単純な錯視画像での錯視シミュレーション (比較実験 1)

まずは，単純な錯視画像を用いた錯視シミュレーションを行う．

7.3.1.1 シミュレーション方法

中心視周辺視適用の錯視シミュレーション (以降，CP_sim) と中心視周辺視非適用の錯視シミュレーション (以降，non_CP_sim) で比較実験を行った．

入力画像は 4 章でも使用した，図 4.1.2 と図 4.1.3 を使用する．パラメーターは以下の表 7.3.1.1 に基づいている．表中の σ は non_CP_sim が使用するパラメーターであり，全画像領域で $\sigma = 4.0$ である．CP_sim は図 7.1.1 にあるように，σ は中心は $\sigma = 4.0$，対角部では $\sigma = 16$ である．これらのパラメーターは，パラメータの全パターンを結果選出したものである．

7.3.1.2 シミュレーション結果

シミュレーション結果を図 7.3.1 と図 7.3.2 にて示す．図 7.3.1 は中心視周辺視を適用した錯視シミュレーションの結果，対して，図 7.3.2 は中心視周辺視を適用していない錯視シミュレーションの結果である．比較すると特に錯視画像を用いた結果が大きく異なることが分かる．図 7.3.1 では錯視画像を用いた場合の結果は，周辺部分の OF が全く検出されないのに対して，図 7.3.2 では中心部分と同じように OF は検出される．

7.3.2 より実用的な錯視画像を用いた錯視シミュレーション (比較実験 2)

次に，より実際の錯視研究で用いられているような錯視画像を用いて錯視シミュレーションを行う．

7.3.2.1 シミュレーション方法

入力画像には新たに図 7.3.3 と図 7.3.4 を使用する．比較実験 1 と同様にパラメーターは表 7.3.2.1 に基づいている．またこれも同様に，表 7.3.2.1 のパラメーターはパラメーターの全パターンを結果選出したものである．

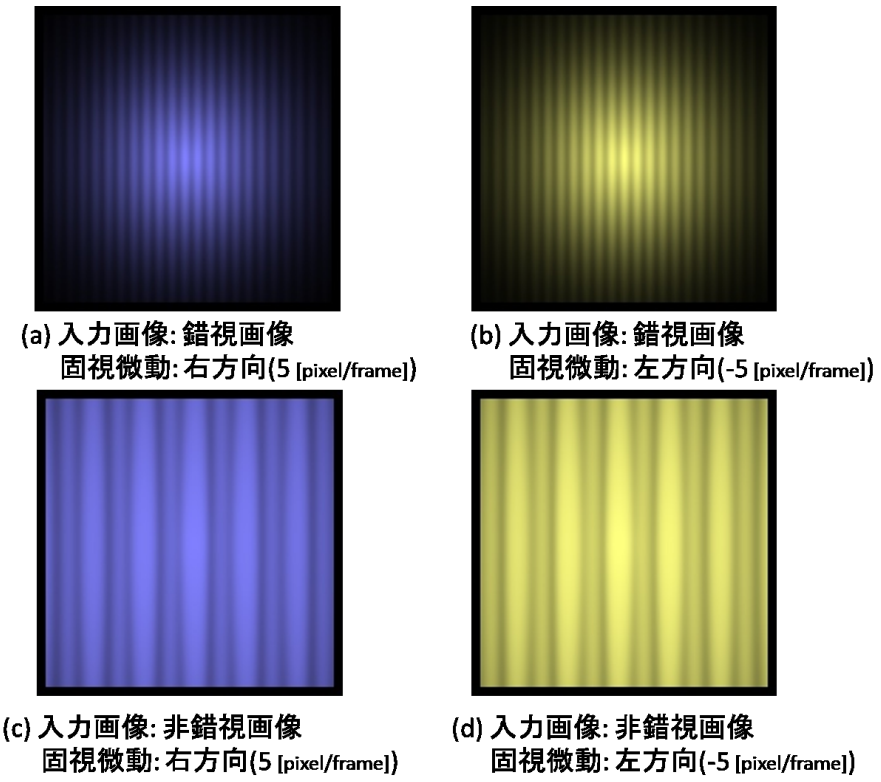


図 7.3.1: 錯視シミュレーション結果．HSV 形式の表示にしており，OF のベクトルの長さを色の濃淡，OF のベクトルの向きを色相にそれぞれ変換している．

表 7.2: 比較実験 2 に用いるパラメーター

τ	σ	ϵ	dx
0.7	4.0	0.00001	1

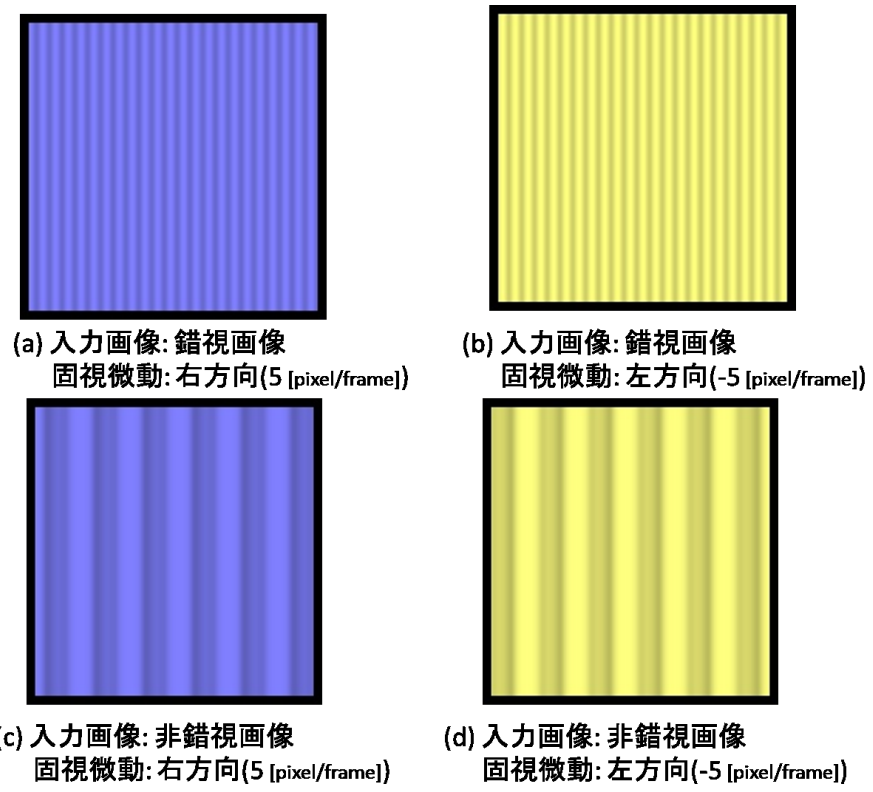


図 7.3.2: 比較のため，中心視周辺視の適用をしていない場合のシミュレーション結果を示す.

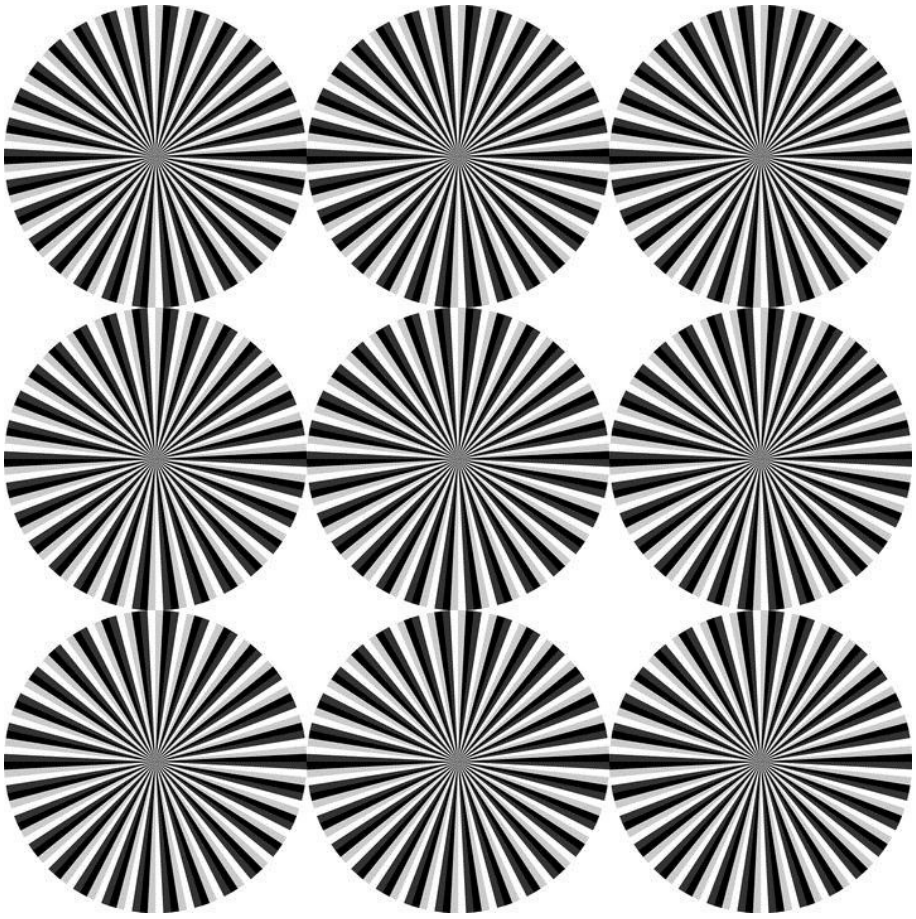


図 7.3.3: 比較実験 2 に用いる錯視画像．図 6.2.1 の円形の錯視画像を 3 × 3 に配置したものである．

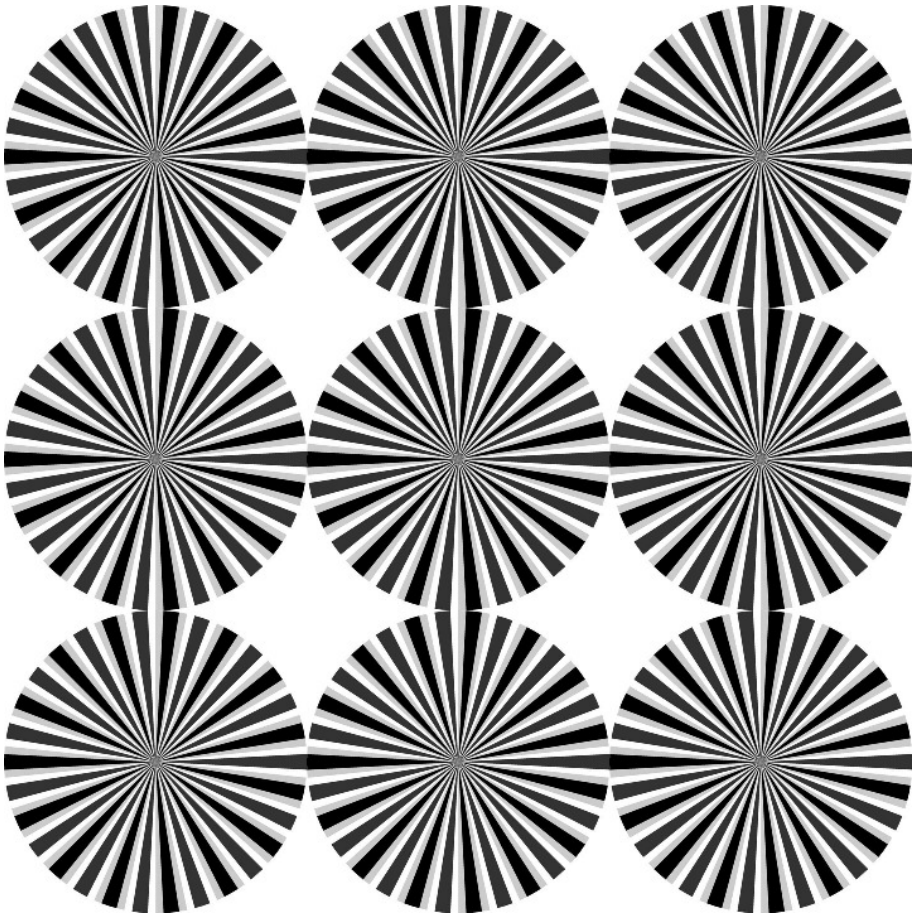


図 7.3.4: 比較実験 2 に用いる非錯視画像．図 6.2.2 の円形の非錯視画像を 3*time*3 に配置したものである．

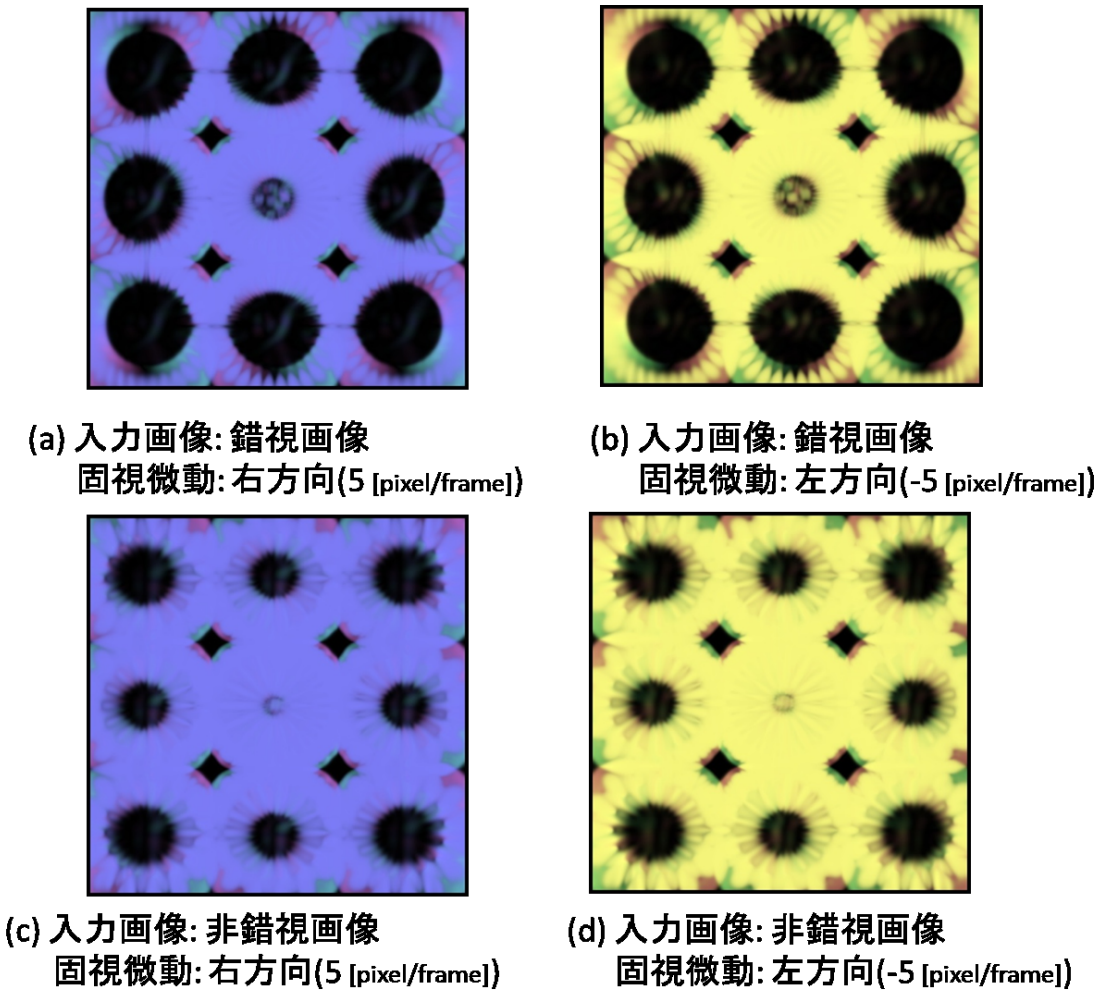


図 7.3.5: 中心視周辺視適用時の錯視シミュレーション結果．外周部で OF が検出できていない箇所が確認できる．

7.3.2.2　シミュレーション結果

図 7.3.5 と図 7.3.6 の両結果から，中心視周辺視の適用で，中心部分の OF 計算は正確に，周辺部分の OF 計算は不正確な傾向となることが確認できた．対して，図 7.3.6 では中心周辺ともに OF 計算は同様の結果となった．この点から，本実装によって中心視周辺視の性質を再現することが実現できたと判断する．より詳細な結果を確認するため，シミュレーション結果の断面図を図 7.3.7 に示す．なお，中心部分・周辺部分の断面の位置は図 7.3.9 に示す．

問題の箇所では，rotating_snake と non_rotating_snake を比較して最大で 0.8[pixel/frame] も OF 値が異なる結果となった．つまり，外周部で錯視が発生しやすい箇所が存在することを示している．この結果は，実際に人間が錯視画像を見た時と同様の反応であるいえる [2]．以上のことから，中

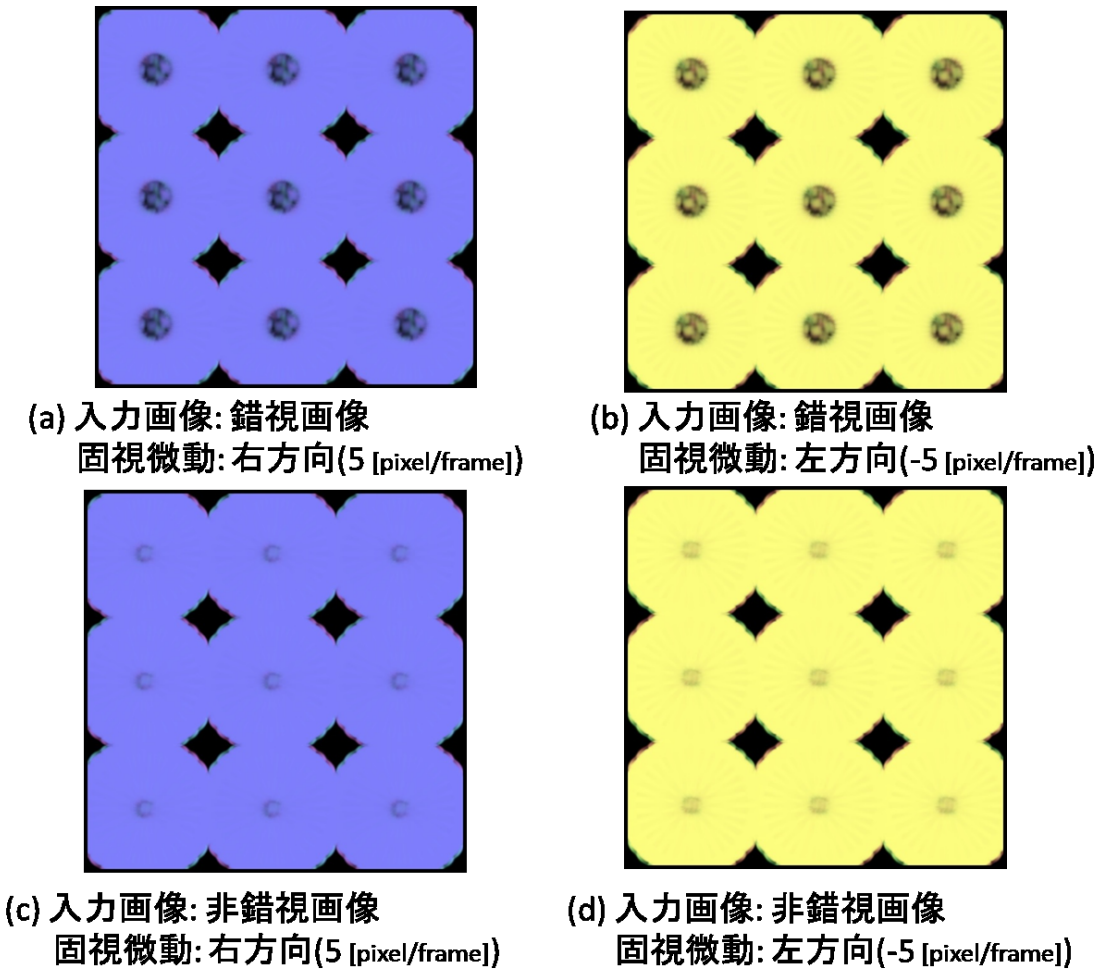
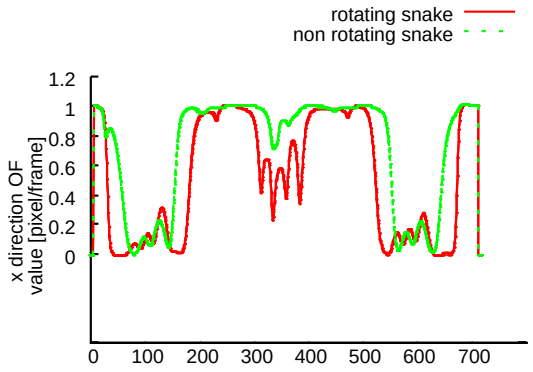
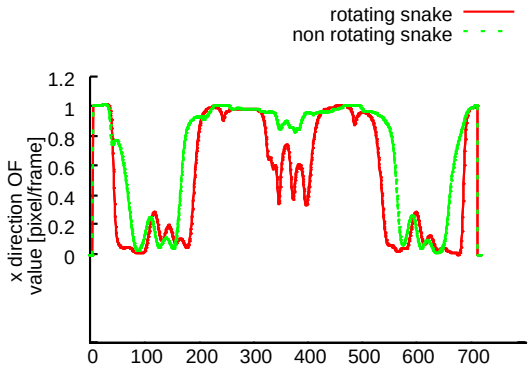


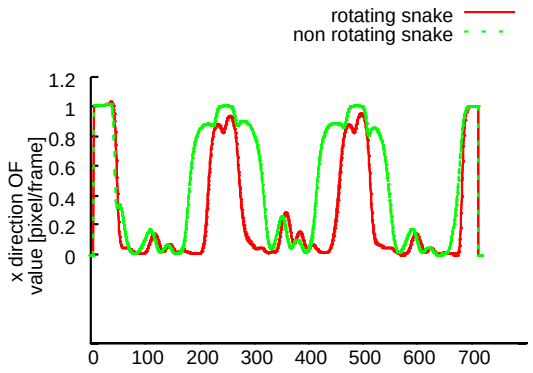
図 7.3.6: 中心視周辺視の適用をしていない場合のシミュレーション結果



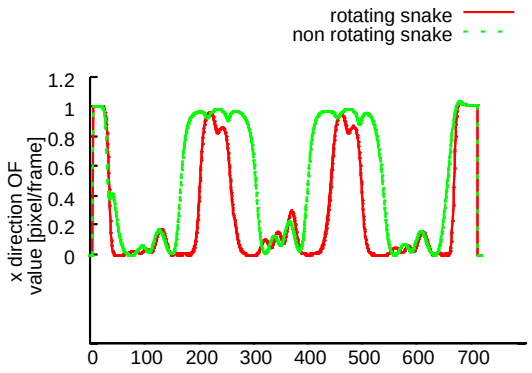
(a) 位置：中心，
固視微動: 右方向 (1 [pixel/frame])



(b) 位置：中心，
固視微動: 左方向 (-1 [pixel/frame])

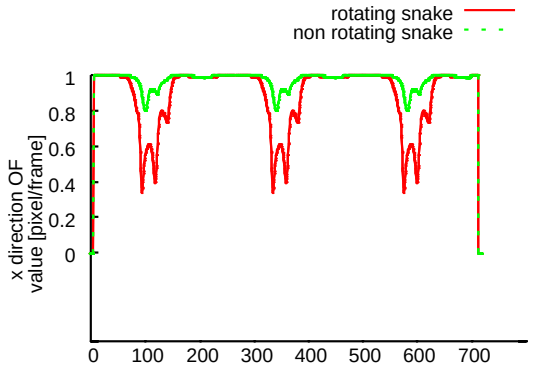


(c) 位置: 外周部，
固視微動: 右方向 (1 [pixel/frame])

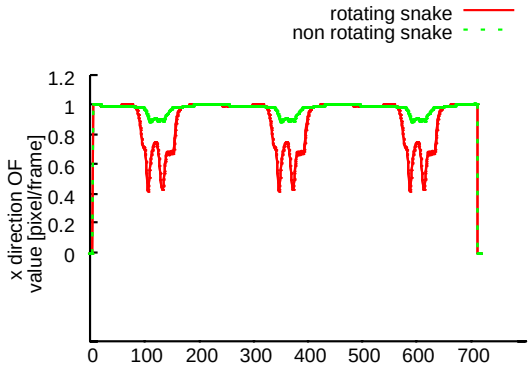


(c) 位置: 外周部，固視微動: 左方向 (-1 [pixel/frame])

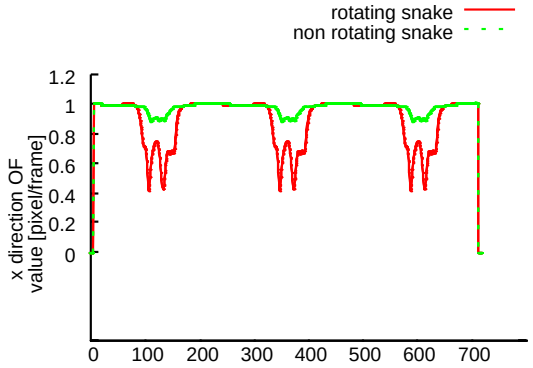
図 7.3.7: 中心視周辺視を適用した錯視シミュレーション結果をより詳細な結果を示したもの．(a) と (b) は中心部分の断面を表示したもの．対して，(c) と (d) は周辺部分の断面を表示したもの．



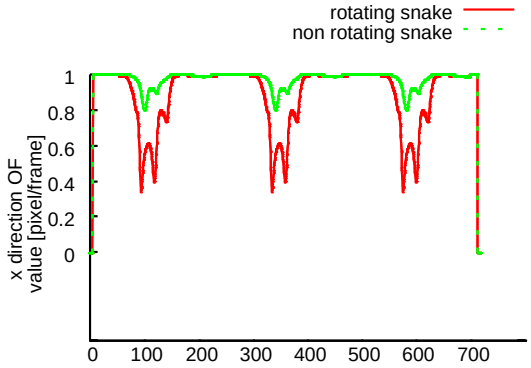
(a) 位置：中心，
固視微動: 右方向 (1 [pixel/frame])



(b) 位置：中心，
固視微動: 左方向 (-1 [pixel/frame])



(c) 位置: 外周部，
固視微動: 右方向 (1 [pixel/frame])



(c) 位置: 外周部，固視微動: 左方向 (-1 [pixel/frame])

図 7.3.8: 比較のため，中心視周辺視を適用していない，錯視シミュレーション結果をより詳細な結果を示したもの．(a) と (b) は中心部分の断面を表示したもの．対して，(c) と (d) は周辺部分の断面を表示したもの．

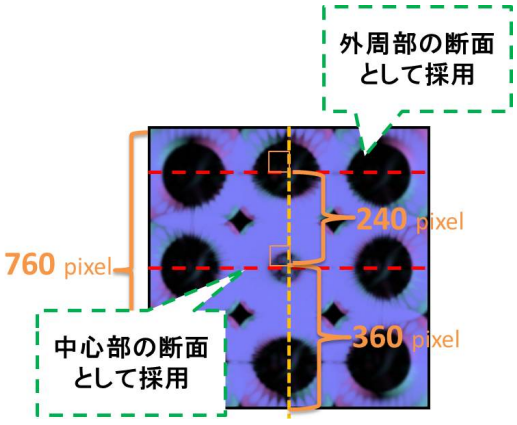


図 7.3.9: 中心部分と周辺部分の断面の位置

心視周辺視に関連した錯視現象を再現できた結論づける.

第8章　結論と今後の課題

8.1　結論

本研究では，GPU 搭載 PC クラスタを使用した錯視シミュレーションシステムの構築について述べた．単純な輝度パターンを使用した錯視シミュレーションにおいて，人間に対する反応をシミュレートすることに成功した．また，高速化にあたっては，GPU 内でのアルゴリズムの最適化，袖領域の交換処理の排除など，通信処理の最適化によって，基礎的な実装と比較して 70 %の処理時間の最適化した．

さらに，人間の視覚の特性の一つである “中心視・周辺視” についてもシミュレートを実現した．“中心視・周辺視” 対応化によって，処理速度が大幅に低下したが，パイプライン化・通信処理の最適化で高速化を図り，“中心視・周辺視” 対応の基礎的な実装と比較して 50 %以上の処理時間の削減を達成した．

8.2　今後の課題

視覚神経系の研究の観点からは，V1 以降の高次の視覚神経系の細胞をシミュレートすることが今後の課題である．本研究では，V1 細胞のみしかシミュレートしていないが，生理学的な研究ではすでに V1 細胞以外の細胞も錯視現象と関連があることが明らかになっている [17].

また，並列処理の研究の観点からは，1 ノード 4GPU といったように複数台の GPU を搭載した PC による実装が今後取り組むべき課題と言える．今回の研究では，通信処理がボトルネックになる傾向が多々存在した．1 ノード 4GPU などの環境では 1 ノード内での通信となるので，通信処理がボトルネックとして発生する可能性も低下すると考えられる．

謝辞

本研究を進めるにあたり，ご指導を頂いた指導教員の吉永努教授に感謝の意を表します.

情報メディアシステム学専攻の佐藤俊治先生には，専門知識と実験の進め方に対し指導していただきました．ここに感謝します.

また，多くのアドバイスや議論をしていただいた研究室のメンバーの皆さんに感謝します.

参考文献

[1] 北岡明佳. 北岡明佳の錯視のページ. http://www.ritsumei.ac.jp/ akitaoka.

[2] Frazer.A and Wilcox.K. Perception of illusory movement. *Nature*, Vol. 281, pp. 565–566, 1979.

[3] Henry Markram. The blue brain projcet. *Neuroscience*, pp. 153–160, 2006.

[4] Zhaoping Li. A neural model of contour integration in the primary visual cortex. *Neural Computation*, Vol. 10, pp. 903–940, 1998.

[5] Isamu Motoyoshi and Frederick A. A. Kingdom. Differential roles of contrast polarity reveal two streams of second-order visual processing. *Vision Research*, Vol. 47, pp. 2047–2054, 2007.

[6] Shunji Satoh and Shiro Usui. Computational theory and applications of a filling-in process at the blind spot. *Neural Networks*, Vol. 21, pp. 1261–1271, 2008.

[7] Openmpi: Open source high performance computing. http://www.unidata.ucar.edu/software/netcdf/.

[8] NVIDIA. Cuda toolkit, 2011. http://developer.nvidia.com/cuda-toolkit-sdk.

[9] Hiroki Sasaki, Shunji Satoh, and Shiro Usui. Neural implementation of coarse-to-fine processing in v1 simple neurons. *Neurocomputing*, Vol. 73, pp. 867–873, 2010.

[10] Robinson E. Pino, Michael Moore, Jason Rogers, and Qing Wu. A columnar v1/v2 visual cortex model and emulation using a ps3 cell-be array. pp. 1667–1674, 2011.

[11] Yusuke Saito, Shunji Satoh, Takefumi Miyoshi, Hidetsugu Irie, and Tsutomu Yoshinaga. Parallel numerical simulation for the linear model of visual neurons with mpi. *SIC Technical Report(IPSJ)*, Vol. 2011-HPC-129, pp. 1–8, 2011. (in Japanese).

[12] Junichi Ohmura, Shunji Satoh, Akira Egashira, Takefumi Miyoshi, Hidetsugu Irie, and Tsutomu Yoshinaga. Multi-gpu acceleration of optical flow computation in visual functional simulation. *2011 Second International Conference on Networking and Computing*, pp. 228–234, 2011.

[13] Anton L. Beer, Andreas H. Hecke, and Mark W. Greenlee. A motion illusion reveals mechanisms of perceptual stabilization. *PLoS ONE*, Vol. 3, p. e2741, 7 2008.

[14] Lucas B.D and Kanade T. An iterative image registration technique with an application to stereo vision. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence(IJCAI-81)*, pp. 674–679, 1981.

[15] Neil R.Carlson. カールソン神経科学テキスト-脳と行動. 丸善, 2006.

[16] Junichi Ohmura. Construction for a gpu-accelerated simulation enviroment for a human visual system. *The Univercity of Electro-Communications Graduate School*, p. 110, 2011.

[17] Ichiro Kuriki, Hiroshi Ashida, Ikuya Murakami, and Akiyoshi Kitaoka. Functional brain imaging of the rotating snakes illusion by fmri. *Joumal of Vision*, Vol. 8, pp. 1–10, 2008.

発表論文

[1] Junichi Ohmura, Shunji Satoh, Akira Egashira, Takefumi Miyoshi, Hidetsugu Irie, Tsutomu Yoshinaga: GPU-accelerating method for a human visual system simulation, 2011-HPC-130, p.8. Jul. 2011.

[2] Junichi Ohmura and Akira Egashira and Shunji Satoh and Takefumi Miyoshi and Hidetsugu Irie and Tsutomu Yoshinaga: Multi-GPU Acceleration of Optical Flow Computation in Visual Functional Simulation, 2011 Second International Conference on Networking and Computing, pp.228-234. Dec. 2011.

[3] Akira Egashira and Shunji Satoh and Takefumi Miyoshi and Hidetsugu Irie and Tsutomu Yoshinaga: Parallel Numerical Simulation of Visual Neurons for Analysis of Optical Illusion, 2012 Third International Conference on Networking and Computing, pp.130-136. Dec. 2012.