鏡面反射モデルのパラメタ推定と鏡面反射成分の生成

稲熊伸昭

平成 12 度卒業論文

目 次

第1章　序論

1.1　背景

従来のコンピュータビジョンにおける研究については、解析の対象となる物体の幾何的な側面が重視されることが多く、まず 2 次元入力画像中のエッジ領域といった基本的特徴を抽出するための前処理段階があり、その結果得られた基本特徴の幾何的な関係を解析するというのが主であった [?]。そのため、入力画像の画素輝度値から如何にして目的とするエッジなどの基本特徴を安定に且つ正確に求めるかという点が注目され、数多くのアルゴリズムが開発されてきた。また、このような研究においては、入力画像中の画素輝度値自体に関する考察、すなわちなぜ各画素がそのような輝度値で観察されるのかという点に関する考察が不十分であり、入力画像の画素輝度値は単なる 2 次元配列として与えられるものとして考えられることが多かった。

しかしながら、実際には入力画像の画素輝度値とは、実世界における光源、光源物体表面上の反射、センサ特性という 3 つの要素間の光学的関係より決定されるものであり、任意の 2 次元配列として与えられるものでは決してない。このため、実世界の光学的な側面を考慮しない画像処理アルゴリズムはおのずと限界があった。例えば、両眼視法や領域分割法などにおいては、物体表面上に観察されるハイライトなどは処理を失敗させる大きな原因と考えられてきた。

1970 年代に入ると、Horn により、このように幾何的な側面も重点を置いた画像処理アルゴリズムの開発に対して、実世界の光学的側面を積極的に利用することを目指した新たなアプローチが提唱され、従来の画像処理アルゴリズムでは困難であると考えられてきた問題を解決するために非常に有効であることが示された。その後、コンピュータビジョンの研究において、光学的側面を考慮した画像処理アルゴリズムが開発されてきた。そして 1990 年代に入り、このような光学的モデルに基づく画像処理手法全般は物理ベースドビジョン (Physics-Based Vision) と呼ばれコ

ンピュータビジョンにおける一分野を形成することとなった。

現在、この物理ベースドビジョンにおいて利用される光学的モデルとしては、屋内・屋外環境における光源モデル、物体表面間の相互反射モデル、滑らかな表面それに粗い表面における反射モデル、反射光における偏光状態に関する偏光反射モデル、撮像装置における結像モデル、など多くのものが存在する。

本論文ではこのいくつかの反射モデルを利用し、平面物体における鏡面反射成分の生成を行う。

1.2 コンピュータ・ビジョンにおける実物体のモデリング

近年、コンピュータ・ビジョンの分野において実物体のモデリングに関する研究が盛んに行われている。コンピュータ・ビジョンにおけるモデリングとは、物体の形状情報や、色やハイライトなどの「見え」つまり反射率に依存する情報をモデル化することを指す。このモデルは画像処理による物体認識等への応用が考えられる一方、物体のパラメータがモデル化されていることを逆に利用して、物体の画像を生成することも可能となる。反射率、形状の両方がモデル化されていれば、ある任意の方向から見た画像、様々な光源下での見え方など、実際の物体を見る場合と同じような変化に対応することができる [?]。

このような実物体のモデリング技術は、実際に存在するものを画像上で見せる際に非常に有用であり、様々な用途への応用が期待される。例えば、重要文化財のデータ保存に大いに役立つ。特に木造建築や木製美術品の多い日本では、腐食、老朽化、損壊、火災などの心配を常にしていなければならないが、電子データとしてコンピュータの中に保存すると、そのような心配に捕らわれず半永久的に重要文化財の姿を後世に残すことができる。また電子図書館や仮想美術館など、最近になって現実化しつつある、コンピュータを使って書籍や絵画、美術品をディスプレイする必要のあるものには大いに利用される可能性がある。他にも、インターネットの普及に伴って実用化されてきた電子モールやネットショッピングなどにも同様に需要が考えられる。

以上のように、コンピュータ・ビジョンの実物体モデリングは将来様々な応用が期待されるが、反射モデルのパラメータを推定し、鏡面反射成

分を生成しようという試みは少ない。

1.3 研究の目的

コンピュータグラフィックスにおいて、照明モデルやカラーモデルと呼ばれる、ある物体表面における反射を表現するモデルは現実感あるイメージを生成する手段として用いられてきた。物体の反射特性は物体を構成する物質に依存する。また、反射モデルは色や幾何学的要素に関するパラメータが含まれているが、実物体において、適切なパラメータを求めることは難しく、これまでのところ、これらのパラメータは試行錯誤の末ユーザが選んでいる [?] [?]。

本論文では、実物体のカラー写真画像からこれらのパラメータを推定する方法を提案し、鏡面反射成分を生成し、有効性を示す。

1.4 本論文の構成

第 2 章では、反射成分の分離を行うために必要である光学的原理を示す。可視光領域における反射、偏光の原理について述べる。

第 3 章では、鏡面反射成分の生成のために必要である反射モデルについて述べる。

第 4 章では、2.3 章の原理を踏まえて、入力となるカラー画像から、反射モデルのパラメータ推定についての原理を述べる。

第 5 章では、2、3、4 章の原理を踏まえ、具体的に実験方法を述べ、あわせて実験結果と考察を述べる。

第 6 章では、本論文の結論をまとめる。

また付録として、光学的、幾何学的キャリブレーションについて述べる。また、実験で用いたプログラムの外部仕様と内部仕様、そして、プログラムのソースを載せる。

第2章　可視光領域における反射・偏光の原理

2.1　はじめに

本章では、前章で述べた背景を踏まえ、反射成分の分離を行うために必要である光学的原理を示す。一般に波長域 300 μm 〜 800 μm 領域の可視光と呼ばれる光における反射、偏光の原理について述べる。

2.2　光の反射の仕組み

物体からの反射光は一様ではなく、物体の材質、形状によって様々な成分がある。不均質で透明な誘電体を考えた場合、反射光としては次の４つの成分が考えられる。

1. 入射光の波長よりも十分大きい平らな面から正反射方向に１回で反射する光
2. 入射光の波長よりも十分大きい微細面から成る粗い面の間で少なくとも２回以上反射してきた光
3. 物体表面を透過して、中で色素等の pigment のために反射を繰り返した後に再び空気中に透過してきた光
4. 入射光の波長と同程度か、より小さい微細面で回折された光

この４つの反射成分の様子を図 2.1 に示した。1 の反射光を鏡面反射光と呼び、2–4 の反射光を拡散反射光と呼ぶ [?]。ここで、4 の成分は非常に小さいので、物体が波長オーダーの周期構造を持つ場合以外はほとんど無視できる。完全に平らな面を持つ物体では反射成分は 1 と 3 のみになるが、大抵の物体の反射光は上にあげた反射成分の和になっている。そのため、物体からの反射の様子を模式的に表してみると図 2.2 に示したように、3 つのタイプに大別される。(a) は完全な鏡面の物体からの反射を表

しており、一方、(c) は面が粗く完全拡散反射面の物体からの反射を表している。(b) は面の粗さが先の 2 つの中間であり、どの程度正反射方向に光が集中するかは、表面の滑らかさに依存する。以上の反射成分を考えると、3 の反射成分は物体中で反射、屈折、吸収を繰り返し、物体の色を認識させる光となるのに対し、1 の反射成分は 1 回で反射するためにエネルギーが最も大きく、光源の色とほぼ同じ色で見える。2 の反射成分は、1 に比べるとエネルギーは小さくなるが 1 と同様に吸収はほぼないので光源の色と同じになる。1 の反射成分に 2 の反射成分を合わせて、物体には正反射方向のまわりにほぼ光源の色と同じである強い光が見えることになる。この部分の光をハイライトと呼ぶ。ハイライトは、粗い面であれば 2 の成分が大きくなるので、広く見えるが光自体は弱くなり、完全に滑らかな面であれば 1 の成分のみになるので、非常に狭い領域で強い光となる。

2.3 フレネルの反射の公式

反射光の測定のために、正反射における反射と透過の原理について以下に示す [?]。

図 2.3 に示したように屈折率が n_1、n_2 なる媒質 1、2 の境界面が $x-y$ 面内にあり、媒質 1 から平面の光が $x-z$ 面内で媒質 2 に入射する場合を考える。このとき、光は境界面で屈折して媒質 2 に透過していき、同時に一部分は境界面で反射する。透明な誘電体を考えるので、可視域において吸収は無視できる。入射光、反射光、透過光の $x-z$ 面に平行な成分、垂直な成分をそれぞれ添字 p、s で表す。

入射角 ϕ_1、反射角 ϕ_1'、透過角 ϕ_2 はそれぞれ図 2.3 に示したように定義する。ただし、入射光と反射光は同じ媒質中を通るので $\phi_1 = \pi - \phi_1'$ である。このとき、入射、反射、透過光の電界ベクトルのうち $x-z$ 面内に平行な成分、E_{ap}、E_{rp}、E_{tp} は次のように表せる。

$$\begin{aligned} E_{ap} &= A_p \exp[i\{\omega t - k_1(x\sin\phi_1 + z\cos\phi_1)\}] \\ E_{rp} &= R_p \exp[i\{\omega t - k_1(x\sin\phi_1 - z\cos\phi_1)\}] \\ E_{tp} &= T_p \exp[i\{\omega t - k_2(x\sin\phi_2 + z\cos\phi_2)\}] \end{aligned} \tag{2.1}$$

A_p、R_p、T_p は振幅、ω は角周波数、k_1、k_2 は媒質 1、2 中の波数である。垂直成分も同様に表せる。添字 a、r、t はそれぞれ入射光、反射光、透過光を表す。

ここで、光の反射率について考える。透過の際に光の進む方向が屈折することを表す、スネル (Snell) の法則を次に示す。

$$n_1 \sin\phi_1 = n_2 \sin\phi_2 \tag{2.2}$$

境界面において、電界と磁界の面内成分が連続でなければならないことから、媒質 1 側の入射光と反射光の振幅の和が媒質 2 側の透過光の振幅と x、y 方向で等しくなければならない。このことから次式を得る。

$$E_{aj} + E_{rj} = E_{tj}, \qquad H_{aj} + H_{rj} = H_{tj} \qquad (j = x, y) \tag{2.3}$$

E、H はそれぞれ電界、磁界を表す。以上の式 (2.1)(2.2)(2.3) を用いて平行成分、垂直成分についての振幅反射率 r_p、r_s が次のように求められる。

$$\begin{aligned} r_p &= \frac{\tan(\phi_1 - \phi_2)}{\tan(\phi_1 + \phi_2)} \\ r_s &= -\frac{\sin(\phi_1 - \phi_2)}{\sin(\phi_1 + \phi_2)} \end{aligned} \tag{2.4}$$

これを、フレネルの公式と呼ぶ。入射角に対する反射率の依存性を図 2.4 に示した。

次に、強度反射率を求める。光強度の大きさは次式で与えられる。

$$I = \frac{nE^2}{2\sqrt{\mu_0}} \tag{2.5}$$

n は各媒質の屈折率、μ_0 は真空透磁率である。これより、式 (2.4) を用いて強度反射率は次式のように得られる。

$$\begin{aligned} F_p &= \frac{\tan^2(\phi_1 - \phi_2)}{\tan^2(\phi_1 + \phi_2)} \\ F_s &= \frac{\sin^2(\phi_1 - \phi_2)}{\sin^2(\phi_1 + \phi_2)} \end{aligned} \tag{2.6}$$

強度反射率 F_p、F_s をフレネル反射係数と呼ぶ。このフレネル反射係数の入射角依存性を図 2.5 に示した。

式 (2.4) より、$r_p = 0$ とする入射角が存在することがわかる。このような入射角をブリュースタ (Brewster) 角 ϕ_b と呼ぶ。ブリュースタ角は、スネルの法則より次式で与えられる。

$$\tan\phi_b = \frac{n_2}{n_1} \tag{2.7}$$

2.4 光と偏光

電磁波は伝搬方向に垂直な面内で振動する横波であるので、その面内で方向性のある振動をする。この光波の振動の偏り、すなわち偏光の性質について考える。

偏光は、光の偏光の程度を表す偏光度と、偏光の性質を表す偏光状態の２つで表すことができる。偏光状態は、互いに直交する振動面の振動の振幅と位相関係によって、直線偏光、円偏光、楕円偏光に区別することができる。本研究においてはインコヒーレント光を扱うため、位相関係はランダムであり、さらに直線偏光子を用いるために円偏光や楕円偏光を検出することはない。そこで偏光状態については、完全偏光はすべて直線偏光として扱う。

一般に自然光は非偏光であり、すべての方向に対してランダムに振動している。図 2.6 に示すように白色光等の自然光を直線偏光子に通した場合、出射光は直線偏光となる。任意の方向の偏光成分の明るさは入射光の $1/2$ の明るさとなる。しかしこのような自然光でも、複屈折性のある結晶を透過したり、物体から反射されたりすると、偏光特性が現れてくる。このように自然光の一部が偏光されている光を部分偏光と呼ぶ。

ここで、物体からの反射光について考える。反射光は拡散反射と鏡面反射の和から成っているが、拡散反射は一般に非偏光であるため正反射である鏡面反射のみについて考える。

2.3 節で示したように方向によって反射率が違うため、偏光子を回したとき明るさに変化が現れる。図 2.5 に示したように入射面に対して垂直である成分の反射率が最も大きく、平行成分の反射率が最も小さくなる。このため、偏光子を回していくと、観察される明るさの変化はこの二つの成分を最大最小として図 2.7 に示したような正弦波形となる。反射光は非偏光と完全偏光の和である部分偏光であるので、非偏光成分として直流成分が現れているのがわかる。全体の明るさ、光強度は $I_{\max} + I_{\min}$、一方完全偏光、つまり直線偏光の光強度は $I_{\max} - I_{\min}$ で表される。ここで、$I_{\max}$、$I_{\min}$ はそれぞれ光強度の最大値、最小値である。拡散反射成分も考慮に入れて、反射光がどれだけ偏光されているかの目安として、偏光度 ρ を次のように定義する。

$$\rho = \frac{I_{\max} - I_{\min}}{I_{\max} + I_{\min}} \tag{2.8}$$

偏光度 ρ は、0 のとき非偏光であり、1 のとき直線偏光であることを表す。反射光が直線偏光になるときは、2.3 節の図 2.5 から平行成分が 0 となっ

て垂直成分のみになるときである。すなわちブリュースタ角で入射するときである。

2.5 反射光の偏光

ここで、鏡面反射成分の偏光について述べる。

偏光している光の $I_{\max}$ と $I_{\min}$ の和が全体の光強度になるので、鏡面反射強度を $I_{specular}$ とおくと、鏡面反射光の場合、次式が得られる。

$$I_{\max} = \frac{F_s}{F_p + F_s} I_{specular}, \qquad I_{\min} = \frac{F_p}{F_p + F_s} I_{specular} \tag{2.9}$$

上の式と式 (2.6) を式 (2.8) に代入し、スネルの法則を考慮すると、偏光度 ρ は次のようになる。

$$\rho = \frac{2 \sin\phi \tan\phi \sqrt{n^2 - \sin^2\phi}}{n^2 - \sin^2\phi + \sin^2\phi \tan^2\phi} \tag{2.10}$$

このように、偏光度 ρ は屈折率 n と入射角 ϕ の関数となっていることがわかる。屈折率 1.5 の場合の偏光度の入射角依存性を図 2.8 に示した。

2.6 反射成分の分離

前節までの原理を踏まえ、本論文で行う反射成分の分離の原理について述べる。

2.4 節で一般に自然光は非偏光であり、すべての方向に対してランダムに振動していることは述べた。

2 つの反射成分のうち、拡散反射成分については常にほぼ非偏光となるとみなすことが出来る。(厳密には物体表面法線と視線方向との角度が 90 度に近い場合には一部直線偏光されるが、それ以外の場合にはほぼ非偏光とみなすことが出来る。)

よって、拡散反射成分として反射される光の強度を I_d とすると、その反射光を直線偏光子を通して観察した場合の強度は、直線偏光子の回転方向によらず $I = (I_d)/2$ となる [?]。

鏡面反射成分については 2.5 節で述べたように、偏光度は入射角と屈折率に依存する。

しかし、今回は光源となる自然光を偏光子によって完全に偏光させることにより、鏡面反射成分は完全に直線偏光していると仮定する。また、拡散反射成分は光源が偏光であったとしても、内部でのランダムな反射のため非偏光であると仮定する。

つまり、偏光子を通して反射光を観察したとき、光強度最大のときを I_{max}、光強度最小のときを I_{min} とすると、次式が得られる。

$$(I_{max}) = (\text{鏡面反射成分}) + (\text{拡散反射成分})/2 \tag{2.11}$$

$$(I_{min}) = (\text{拡散反射成分})/2 \tag{2.12}$$

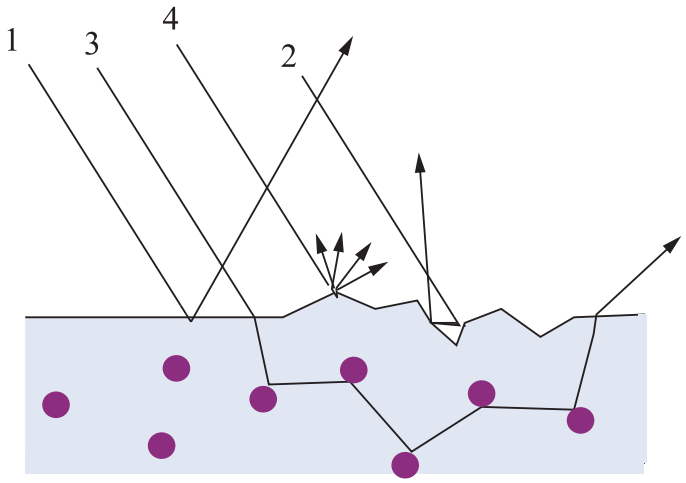
よって、次式のように反射成分を分離することが出来る。

$$(\text{鏡面反射成分}) = (I_{max}) - (I_{min}) \tag{2.13}$$

$$(\text{拡散反射成分}) = (I_{min}) * 2 \tag{2.14}$$

2.7 おわりに

本章では、反射成分の分離を行うために必要である光学的原理を示した。具体的には、光の反射、屈折の公式、偏光、反射成分の分離の原理について述べた。



1 : 鏡面反射成分
2 : 表面で 2 回以上反射された成分
3 : 媒質中を透過してくる成分
4 : 微細面での回折光成分

図2.1 反射の仕組み

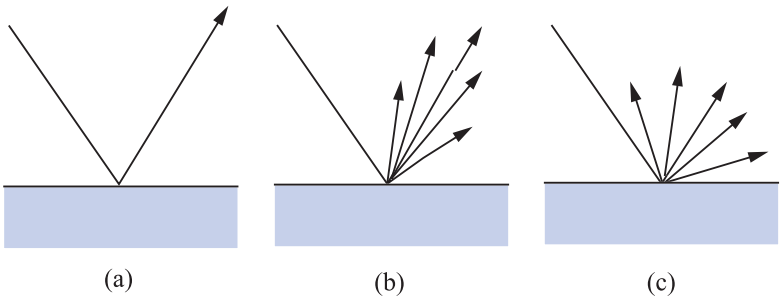


図2.2 表面粗さに依存した反射の様子

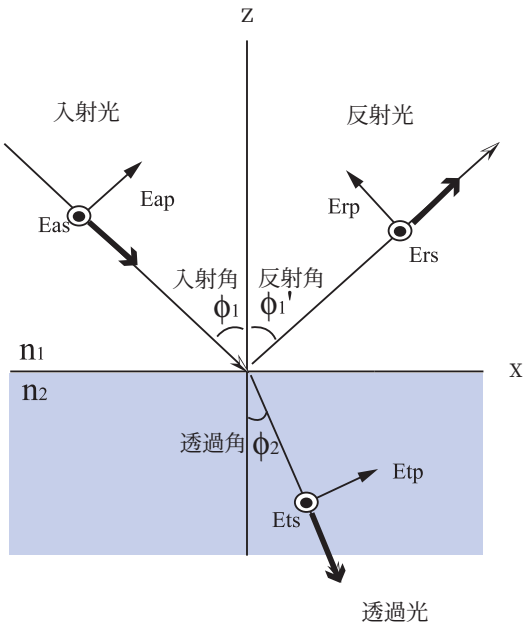


図2.3 フレネル反射

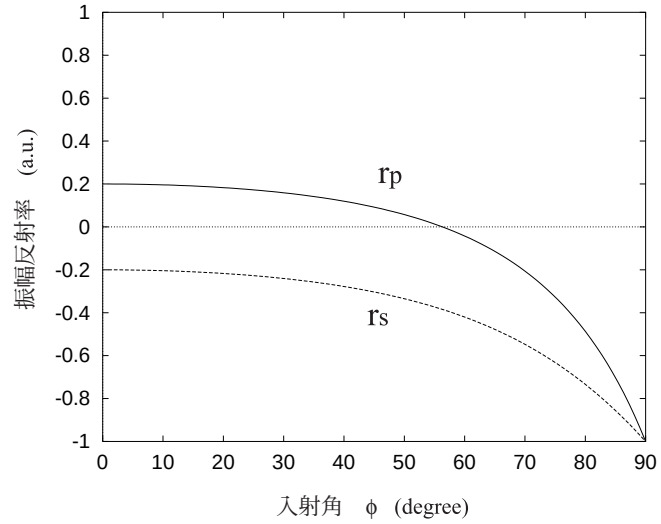
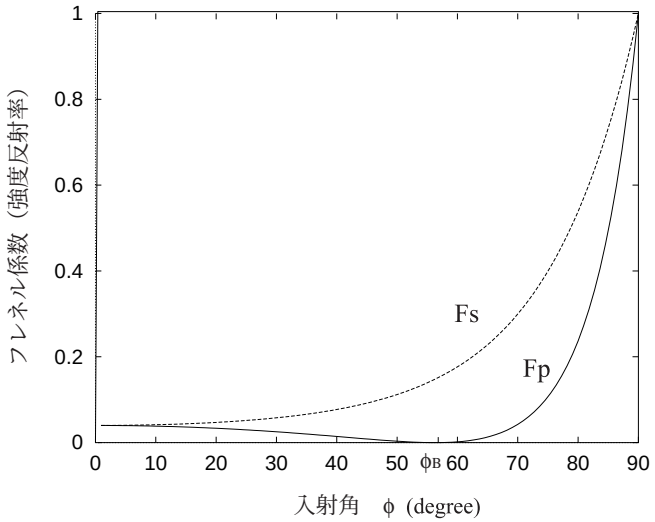


図2.4 振幅反射率の入射角特性



ϕ_B：ブリュースタ角

図2.5 フレネル反射係数（強度反射率）の入射角特性

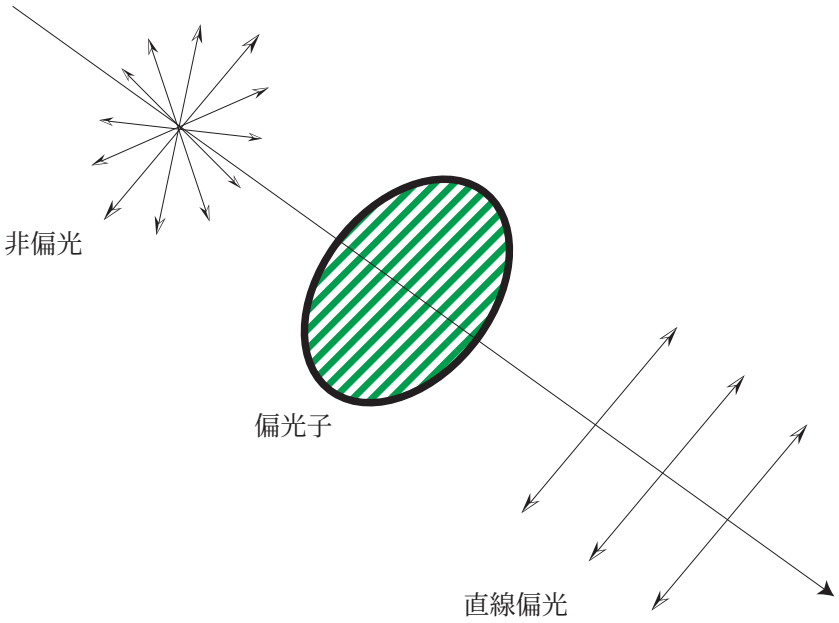


図2.6　偏光

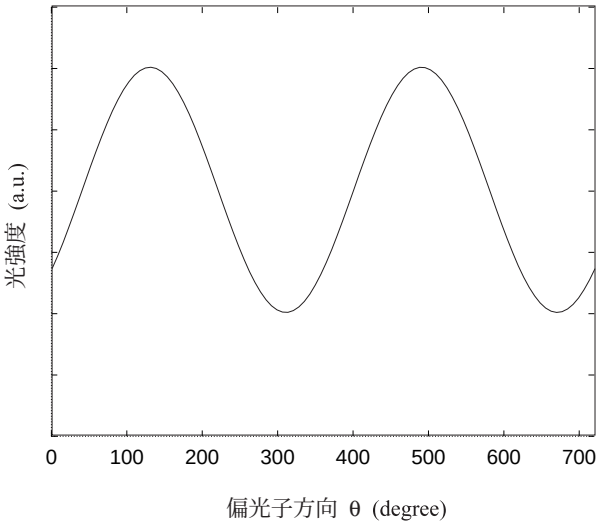


図2.7 反射光の偏光

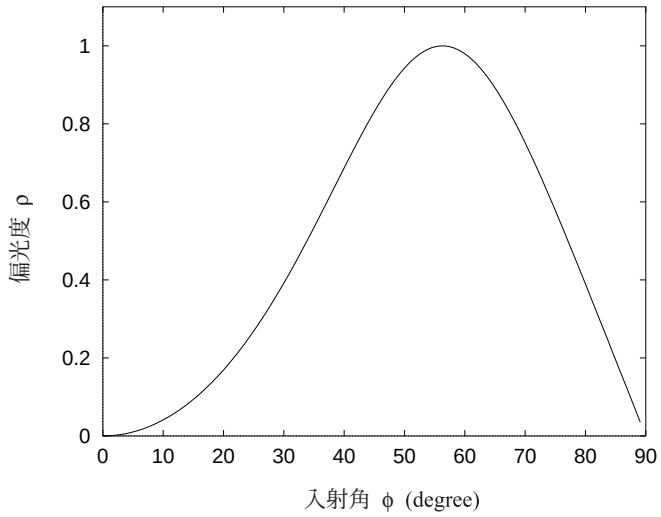


図2.8 偏光度の入射角依存性 (n=1.5)

第3章　反射モデル

3.1　はじめに

本章では、今回用いる鏡面反射成分を生成するために必要な Torrance-Sparrow 反射モデル [?] や、その他代表的な反射モデルを説明する。ここで説明する反射モデルとは、2 章で述べた主な 2 つの反射成分 (鏡面反射、拡散反射) の光強度をさまざまな幾何学的、物理学的パラメータを使って表現したものである。

3.2　Phong モデル

Phong モデル [?] は、物体の反射特性を数式で表現した最初のモデルである。非常に簡潔に表現されているが、コンピュータビジョンにおいては広く受け入れられている。

Phong モデルでは、拡散反射成分は次式のように、入射角の余弦と入射光の強さに比例するとしている。

$$I_d = I_i k_d \cos\alpha = I_i k_d (L \cdot N) \tag{3.1}$$

ここで、I_d は拡散反射光の強さであり、I_i は入射光の強さであり、k_d は拡散反射率であり、α は入射角であり、L は光源方向ベクトルであり、N は表面法線ベクトルである。(図 3.1)

また、Phong モデルでは次式のように、鏡面反射光の強さは入射角の余弦の n 乗と鏡面反射率に比例するとしている。ここで新たな定数 n はハイライトの特性を示す。

また、厳密には鏡面反射率は光の波長によってことなるが、一定値 W としている。

$$S = I_i W \cos^n \gamma \tag{3.2}$$

ここで、S は、鏡面反射光の強さを表し、I_i は入射光の強さ、W は鏡面反射率である。また、n は ハイライトの特性を示す定数である。n =0 のときは完全に鏡面反射を示す。γ は光源の反射ベクトルと視線ベクトルのなす角を表す。

また、環境光については一定で、$I_a k_a$ としている。

そして、拡散、鏡面反射、環境光の 3 要素を考慮した場合の光の強さは次式のように表現している。

$$I = k_d \cos\alpha + I_i W \cos^n \gamma + I_a k_a \tag{3.3}$$

ここで、I は視点に入射する光の強さで、k_d は拡散反射率、α は入射角である。また、I_a は環境光の強さを表し、k_a を環境光定数としている。

3.3 Torrance-Sparrow モデル

反射特性を数式化した Phong モデルが提案されて以来、最初の効果的な反射モデルが Torrance-Sparrow モデルである。

Torrance-Sparrow モデルは Blinn が Torrance と Sparrow が提案した物体表面モデルをもとに作った反射モデルである。反射係数や幾何学的要因による光の減衰といったパラメータも入っており、Phong モデルよりも正確に鏡面反射を表現している。

Torrance-Sparrow モデルでは、まず物体表面を構成する微小面の分布を次式に示すように正規確率分布により近似する。このとき、$\alpha = cos^{-1}(N \cdot H)$ であり、b は定数、σ は表面の粗さを表わす係数である。

$$P(\alpha) = be^{-\alpha^2/2*\sigma^2} \tag{3.4}$$

$\alpha = cos^{-1}(N \cdot H)$ で，b は定数，g は表面の粗さパラメータである．

さらに、幾何学的要因（微小面による遮蔽や陰影など）による光の減衰は、次式により表わされる。ただし、V は視点方向、L は光源方向、H は L と V の角度の 2 等分線である。

$$G = min\{1, \frac{2(N \cdot H)(N \cdot V)}{(V \cdot H)}, \frac{2(N \cdot H)(N \cdot L)}{(V \cdot H)}\} \tag{3.5}$$

最後に、反射光自体の減衰を表わすために、$F(\theta, \eta, \lambda)$ が用いられる。ここで、$\theta = cos^{-1}(N \cdot L)$ は入射角、λ は波長、η は屈折率である。

以上の各要素により、Torrance-Sparrow 反射モデルの鏡面反射成分は次式のように記述される。

$$R(N,V,L,g,\eta,\lambda) = \frac{F(N,L,\eta,\lambda)P(N,V,L,g)G(N,V,L)}{(N\cdot V)} \tag{3.6}$$

ここで、反射係数 F は入射角がほぼ一定であればほとんど定数と見ることが出来、さらに幾何学的要因による光の減衰 G は平面と仮定できる板などの物体においては 1 と仮定できる。

これらのことから、Torrance-Sparrow モデルは次式のように表現できる。

$$I = I_d + k\frac{e^{\frac{-\alpha^2}{2\sigma^2}}}{\cos\theta} \tag{3.7}$$

ここで、I_d は拡散反射成分を表し、I は鏡面反射輝度を表し、k は定数であり、σ は物体の粗さを表すパラメータ (粗さ定数) であり、α は入射角と視点方向のなす角の 2 等分線が表面法線となす角で、θ は、視点方向と表面法線方向がなす角である。(図 3.2)

3.4 おわりに

本章では、今回鏡面反射成分を生成するために必要な Torrance-Sparrow 反射モデルと、代表的な反射モデルである Phong モデルについて述べた。

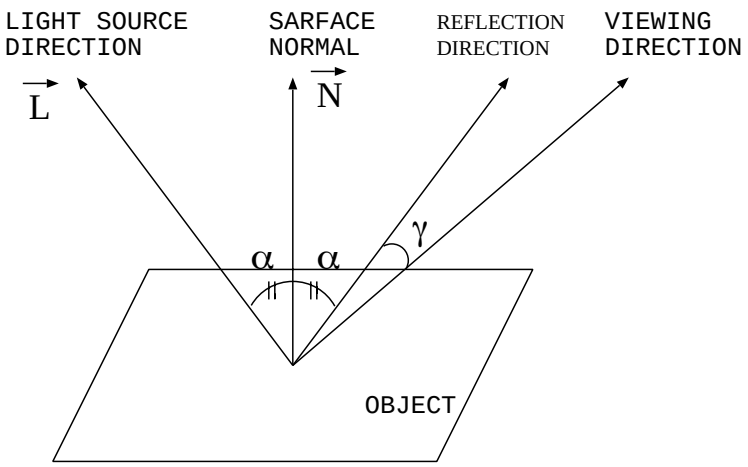


図 3.1: Phong Model

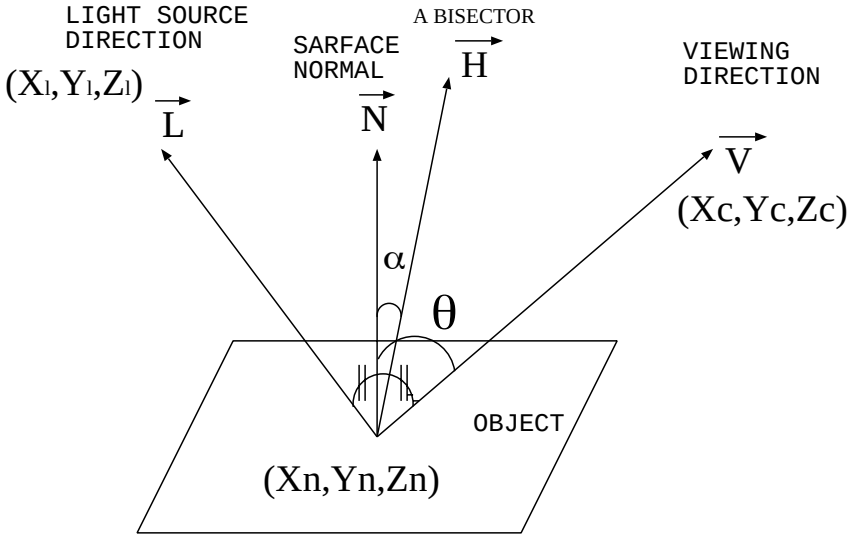


図 3.2: Torrance-Sparrow Model

第4章　実画像計測からのパラメータ推定

4.1　はじめに

本章では、2、3 章で述べた原理を踏まえ、鏡面反射成分を生成する反射モデルのパラメータの推定の原理を述べる。

4.2　パラメータ推定

Torrance-Sparrow 反射モデルは次式のように表現されることは 3 章で述べた。

$$I = I_d + k\frac{e^{\frac{-\alpha^2}{2*\sigma^2}}}{\cos\theta} \tag{4.1}$$

ここで、I は輝度値、I_d は拡散反射輝度を表し、I は鏡面反射輝度を表し、k は定数であり、σ は物体の粗さを表すパラメータ (粗さ定数) であり、α は、入射角と視点方向のなす角の 2 等分線が表面法線となす角で、θ は、表面法線方向と視点方向のなす角である。

本章では式 (4.1) で表現される、Torrance-Sparrow 反射モデルの 2 項目 (鏡面反射成分の項) のパラメータの決定法を述べる。

式 (4.1) で表現される反射モデルの 2 項目のパラメータ推定するために、まず、鏡面反射成分を取り出す必要がある。

反射成分の分離法は 2 章で述べたとおりである。

次に、各ピクセルにおけるを θ、α 求める方法を述べる。今回は θ、α を光源位置、視点位置、各ピクセルの 3 次元座標から求める。(各ピクセルの 3 次元座標の決定法は巻末の付録に述べてある。)

ここでは、画像中の 2 次元座標がわかっていれば、実際の 3 次元座標でどのような座標をとるかがわかるものとする。

図 4.1 において、(X_c, Y_c, Z_c)、(X_n, Y_n, Z_n)、(X_l, Y_l, Z_l) はそれぞれ、カメラ、物体のある点 (ピクセル)、光源の 3 次元座標である。

θ は (X_c, Y_c, Z_c) と (X_n, Y_n, Z_n) の間の角なので、この図から、

$$\cos\theta = \frac{Z_c - Z_n}{\sqrt{(X_c - X_n)^2 + (Y_c - Y_n)^2 + (Z_c - Z_n)^2}} \tag{4.2}$$

となり、θ が求められる。

次に α について述べる。α は照明ベクトル $\overrightarrow{L}$ とカメラベクトル $\overrightarrow{C}$ の 2 等分ベクトル $\overrightarrow{D}$ と物体の表面法線ベクトルとなす角である。

よって、$\overrightarrow{L}$、$\overrightarrow{C}$、$\overrightarrow{D}$ はそれぞれ式 (3.5) のように表現することが出来る。

$$\overrightarrow{L} = (X_l - X_n, Y_l - Y_n, Z_l - Z_n) \tag{4.3}$$

$$\overrightarrow{C} = (X_c - X_n, Y_c - Y_n, Z_c - Z_n) \tag{4.4}$$

$$\begin{aligned}\overrightarrow{D} &= (\overrightarrow{L} + \overrightarrow{C})/2 \\ &= (X_l - X_n, Y_l - Y_n, Z_l - Z_n) + (X_c - X_n, Y_c - Y_n, Z_c - Z_n)/2 \\ &= (X_l + X_c - 2X_n, Y_l + Y_c - 2Y_n, Z_l + Z_c - 2Z_n)/2\end{aligned} \tag{4.5}$$

また、表面法線方向 $\overrightarrow{n}$ は、平面物体を仮定しているため常に (0,0,1) である。

よって、α はベクトルの内積から次式によって導き出せる。

$$\cos\alpha = \frac{Z_l + Z_c - 2Z_n}{\sqrt{(X_l + X_c - 2X_n)^2 + (Y_l + Y_c - 2Y_n)^2 + (Z_l + Z_c - 2Z_n)^2}} \tag{4.6}$$

以上のように各ピクセルにおける θ、α を求めることが出来る。

最後に、粗さ定数 σ、定数 k の推定法を述べる。

今、鏡面反射成分は次式のように表現されている。

$$I_s = k\frac{e^{\frac{-\alpha^2}{2\sigma^2}}}{\cos\theta} \tag{4.7}$$

ここで、I_s は鏡面反射輝度値である。

そして、これまでのところで、各ピクセルにおける θ、α、鏡面反射成分のみの画像から I_s を知ることが出来る。

ここで、残りのパラメータ σ、k を最小 2 乗法による直線フィッティングによって決定するため、以下のように上式を線形化した。

$$\ln k = \ln I_s + \ln \cos\theta + \frac{-\alpha^2}{2\sigma^2} \tag{4.8}$$

ここで、以下のように X、Y を定めた。

$$X = \alpha^2/2 \tag{4.9}$$

$$Y = \ln I_s + \ln \cos\theta \tag{4.10}$$

よって最終的には次式のような形になる。

$$Y = -\frac{1}{\sigma^2}X + lnk \tag{4.11}$$

ここで、X、Y は α、θ、I_s の値から求まるため、X、Y をそれぞれ軸とした点がプロットできる。この点の集合を最小 2 乗法によって直線近似することにより、傾きと切片から σ と k がそれぞれ求めることが出来る。

4.3 おわりに

本章では、反射成分を表現している Torrance-Sparrow モデルのパラメータを決定する原理について述べた。具体的には物体、光源、視点の 3 次元座標から各ピクセルにおける θ 、α を求め、さらに、鏡面反射成分の画素輝度値と各ピクセルにおける θ、α から最小 2 乗法を用いて、粗さ定数 σ と定数 k を決定する原理について述べた。

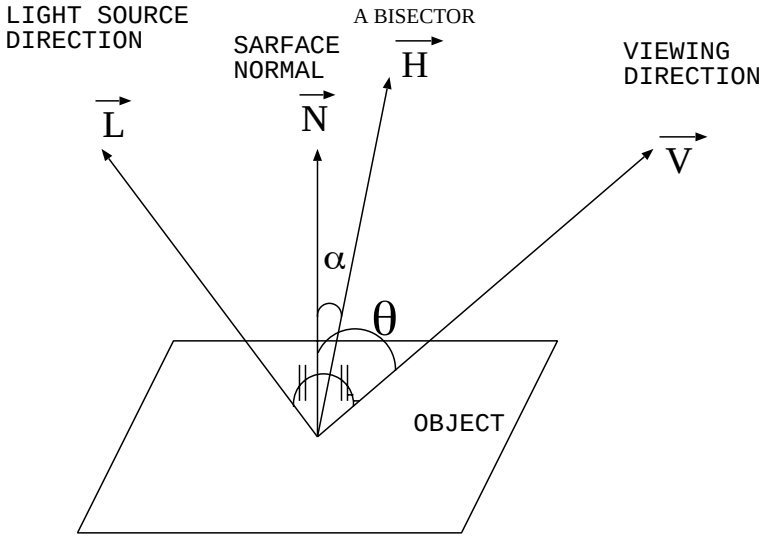


図 4.1: Torrance-Sparrow Model

第5章　実験及び考察

5.1　はじめに

本章では、前章までで述べた、偏光と反射モデルとパラメタ推定の原理を踏まえ、鏡面反射成分の生成法について提案する。本章では、実験方法と実験装置について述べる。そして、この実験装置を用いた結果を示し、検討を行う。

5.2　実験装置

図 5-1 のような実験装置において実験を行った。

また、表面法線方向 $\vec{n}$ が $(0,0,1)$ となるように 3 次元座標を決めた。

ここで、カメラの位置を (X_c, Y_c, Z_c) とし、光源の位置を (X_l, Y_l, Z_l) とした。また、物体の 3 次元位置を (X_n, Y_n, Z_n) とした。また、$\vec{n}$ は表面法線方向 $(0,0,1)$ を示し、物体 (X_n, Y_n, Z_n) から光源 (X_l, Y_l, Z_l) へのベクトルを照明ベクトル $\vec{L}$ とし、物体 (X_n, Y_n, Z_n) からカメラ (X_c, Y_c, Z_c) へのベクトルをカメラベクトル $\vec{C}$ とした。

5.3　写真の撮影、反射成分の分離

図 5.1 のような実験装置を用い、平面で均一な物質と仮定できる板を撮影した。撮影には SONY の CCD カメラ (DXC-9000) を用いた。

今回行う撮影では全て、10 枚連続して撮影し、その平均を取り、画像を平滑化した。

次に、カメラ側の偏光版をまわしながら、鏡面反射成分が最大になる時と、最小になる時をを撮影した。(図 5.2-5.3)

ここで撮影された写真のうち、最も明るく写っている画像を極大画像と呼び、最も暗く写っている画像を極小画像と呼ぶことにする。

ここで同時に幾何的キャリブレーション (付録 B 参照) を行うために図 5.4 のような座標用の紙も撮影した。

さらに、偏光版をまわした際の偏光板とカメラのレンズの干渉による影響を考慮して、画像の補正用 (付録 A:光学的キャリブレーション参照) の白い紙も同様に偏光版をまわしながら撮影した。(図 5.5-5.6)

2 章でも触れたとおり、

(極小画像) = (拡散反射)/2

であり、

(極大画像) = (鏡面反射) + (拡散反射)/2

であるため、

(極大画像) − (極小画像) = (鏡面反射成分)

であることが言える。

実験ではこの計算をプログラム (付録 E-1 separate.cpp 参照) によって行った。

5.4 θ、α の決定

3 章で述べたように鏡面反射成分を生成するためには、各画像のピクセルにおいての角度 θ と角度 α、物体の粗さ定数、 定数を知る必要がある。

そこで、まず、画像中の各ピクセルにおいての θ、α の決定法を述べる。今回は θ、α を幾何学的に求める。

つまり、光源とカメラと物体の 3 次元位置からそれぞれの角度を求める。ここで、世界座標 (3 次元) と画像座標 (2 次元) の対応を知るために、trai の幾何的キャリブレーション手法を用いた。(付録 B:幾何的キャリブレーション参照) この手法を用いることにより、実際の 3 次元座標において、画像中の 2 次元座標がどのような座標をとるかを知ることが出来る。この幾何的キャリブレーションのために、図 5.4 のような紙の一番左下の点を原点 $(0,0,0)$ とし、画面に向かって右方向を x 軸、画面に向かって上方向を y 軸とし、高さ方向を z 軸とした。さらに、1 座標を 1mm としてカメラと光源の 3 次元位置を実測した。また、キャリブレーションに必要な数点の組 (3 次元座標と 2 次元座標の組) をそれぞれ実測した。

4 章でも述べたように各ピクセル、光源、視点の 3 次元座標から

$$\cos\theta = \frac{Z_c - Z_n}{\sqrt{(X_c - X_n)^2 + (Y_c - Y_n)^2 + (Z_c - Z_n)^2}} \tag{5.1}$$

となることがわかり、θが求められる。

また、αは照明ベクトル $\overrightarrow{L}$ とカメラベクトル $\overrightarrow{C}$ の2等分ベクトル $\overrightarrow{D}$ と物体の表面法線ベクトルとなす角である。

よって、ベクトルの内積から次のような式が導き出せる。

$$\cos\alpha = \frac{Z_l + Z_c - 2Z_n}{\sqrt{(X_l + X_c - 2X_n)^2 + (Y_l + Y_c - 2Y_n)^2 + (Z_l + Z_c - 2Z_n)^2}} \tag{5.2}$$

以上のようにθ、αを求めることが出来る。

実験では、この計算をプログラム (付録 E-2 kakudo.cpp 参照) によって行った。

5.5 粗さ定数、定数の推定

前節で鏡面反射成分 I が取り出せ、前々節で各ピクセルの α、θ が求まった。

3章でも述べたように、鏡面反射モデルは

$$I = k\frac{e^{\frac{-\alpha^2}{\sigma^2}}}{\cos\theta} \tag{5.3}$$

である。

実験では、4章の原理で述べたような (X, Y) の点の組を出力するプログラム (付録 E-3 parameta.cpp 参照) を用いて (X, Y) の点の組を出力し、MicroSoft Excel によりグラフを作り、近似直線から傾き、切片を求め k、σ を推定した。

5.6 鏡面反射成分と画像の生成

前節までで、それぞれのピクセルについての α、θ、そして、粗さ定数 σ、定数 k が求まった。Torrance-Sparrow 反射モデルに、それぞれの数値を各ピクセルについて代入し、鏡面反射輝度 I を求めることが出来る。そして、偏光によって取り出した拡散反射成分は、2章で述べたように実際の半分の画素値である。つまり、求める画像を生成するためには次式で表現されるような計算が必要である。

(求める画像) = (取り出した拡散反射成分) $*$ 2 + (生成した鏡面反射成分) (5.4)

実験では、この計算をプログラム (付録 E-4 seisei.cpp 参照) によって行い、画像を生成した。

5.7 実験結果

図 5.7-5.8 は光学的キャリブレーションの結果である。
図 5.9 は分離した鏡面反射成分である。
図 5.10 は 5.5 節で述べた、パラメタ推定用の rgb 分布である。
図 5.11 は生成した画像である。

5.8 考察

誤差について

生成した画像が元の画像に対して、正しい画像であるか、また、元の画像から推定したパラメタが妥当であるかどうかを検討するために、生成した画像と元の画像のある一列においての画素値を比較する。

図 5.12-5.14 はそれぞれ、赤、緑、青 (r,g,b) 成分ついて元の画像と生成した画像のある一列についての画素輝度値を比較したものである。

図から、おおよそ生成した画像と元の画像の画素輝度値は一致しているといえる。

また、全体としてみたときに、どのくらい生成した画像と元の画像の画素値の間に差が生じているかを表現するために次式で表現するような、1 ピクセルあたりの画素値の差の平均を計算した。表 5.1 は赤、青、緑成分についての 1 ピクセルあたりの画素値の差の平均を示している。

$$(1\text{ピクセルあたりの画素値の差の平均}) = \frac{\sum |I_{mot}[x][y] - I_{sei}[x][y]|}{N} \tag{5.5}$$

ここで、$I_{mot}[x][y]$、$I_{sei}[x][y]$ はそれぞれ、画像座標 (x, y) における、もとの画像の画素値、生成した画像の画素値で、N は総ピクセル数である。

表 5.1: 赤、青、緑成分についての 1 ピクセルあたりの画素値の差の平均

赤成分の 1 ピクセルあたりの画素値の差の平均	3.962554
緑成分の 1 ピクセルあたりの画素値の差の平均	2.920598
青成分の 1 ピクセルあたりの画素値の差の平均	4.072663

表から、1 ピクセルあたりの画素値の差の平均はおおよそ 2 から 4 であるといえる。また、画素値のずれの平均が 2 から 4 というのはかなり小さいといえ、パラメタが正確に求めることが出来たといえる。

誤差の要因として考えられるのは次の点である。

- 物体の 3 次元位置を決定する際に、光源位置とカメラ位置を実測によって測った時の誤差。
- ホワイトバランスの誤差 (例えば、白色の物体を撮影した時には rgb の比は 1:1:1 になるはずだがカメラのホワイトバランスが少しずれるという可能性がある。)
- 偏光板は無色であるという仮定のため。(もし色がついていれば、当然全体的にその色が乗る。)
- 拡散反射成分が微小であるが一部偏光したという可能性。(もし一部偏光したら、鏡面反射のみ取り出したはずのところに拡散反射成分も乗ってしまい、推定したパラメタに誤差が生じる。)

どれも微小で無視できるという仮定であったが、少しずつ誤差が集まったと考えられる。

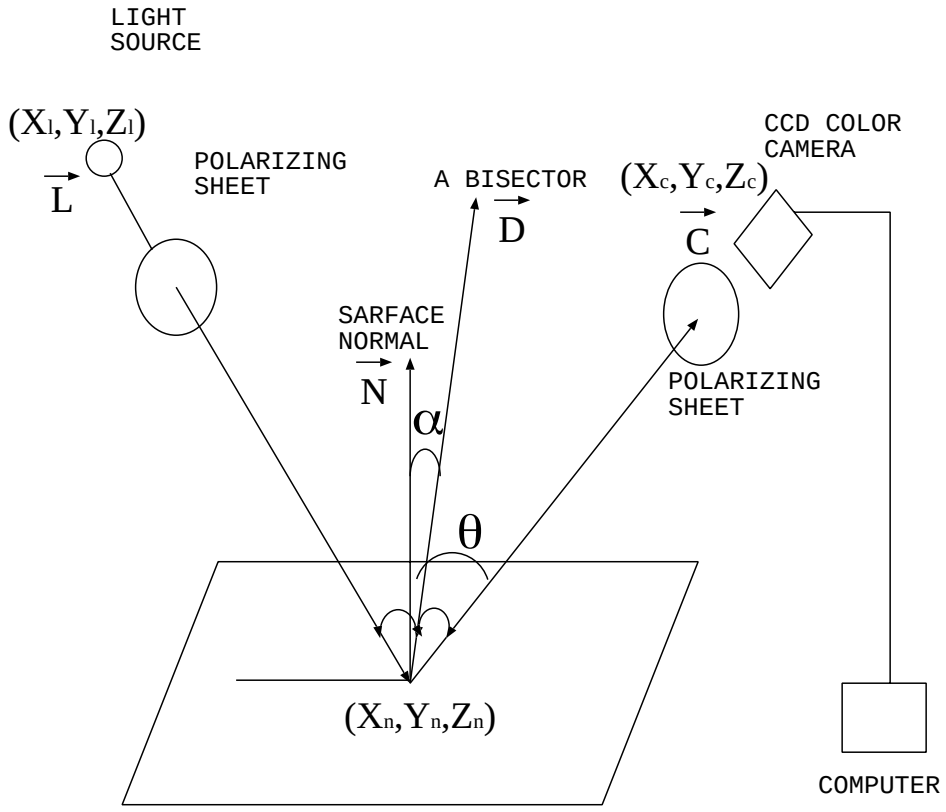

LIGHT
SOURCE
(X_l, Y_l, Z_l)
$\vec{L}$
POLARIZING
SHEET
A BISECTOR
$\vec{D}$
(X_c, Y_c, Z_c)
CCD COLOR
CAMERA
$\vec{C}$
SARFACE
NORMAL
$\vec{N}$
α
POLARIZING
SHEET
θ
(X_n, Y_n, Z_n)
COMPUTER

図 5.1: 実験装置

図 5.2: 極大画像

図 5.3: 極小画像

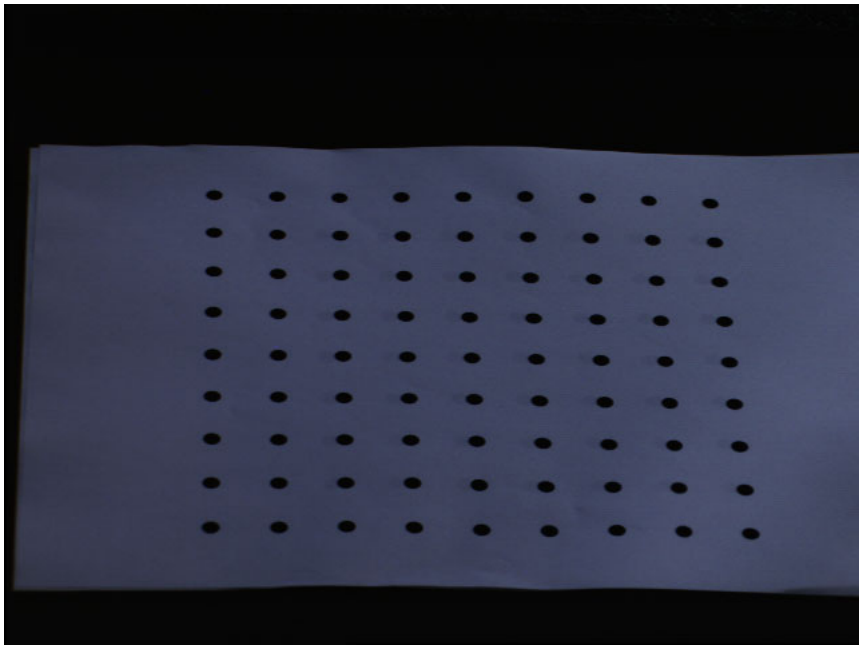


図 5.4: 幾何的キャリブレーション用画像の一例

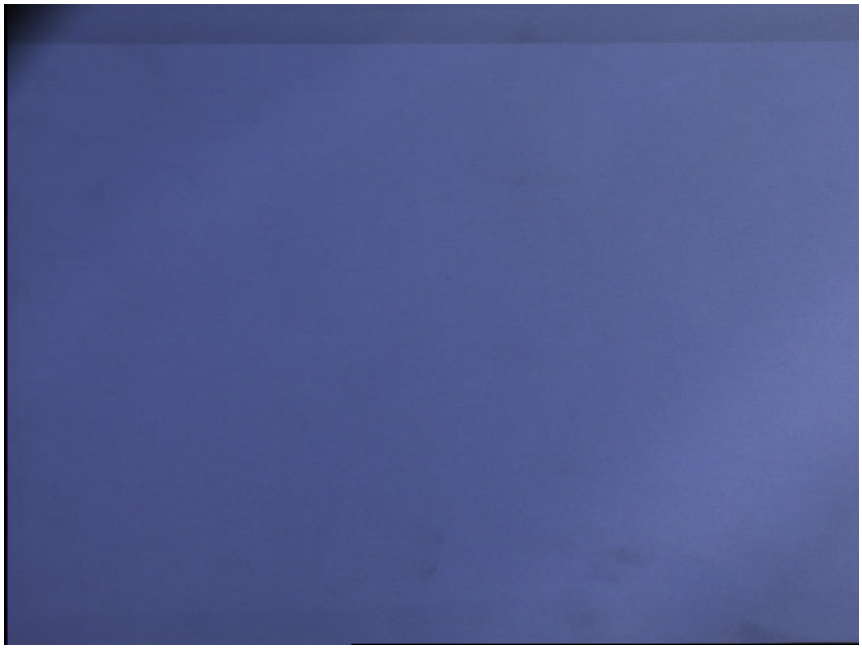


図 5.5: 光学的キャリブレーション用画像の一例

図 5.6: 光学的キャリブレーション用画像の一例

図 5.7: 光学的キャリブレーション結果画像 1

図 5.8: 光学的キャリブレーション結果画像 2

図 5.9: 分離した鏡面反射成分

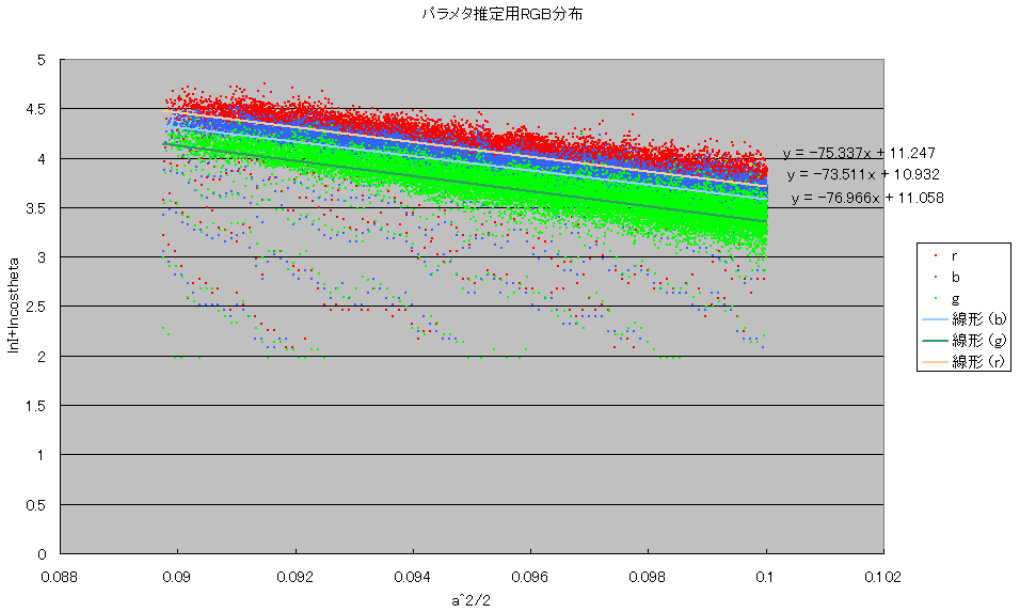


図 5.10: パラメタ推定用 rgb の分布

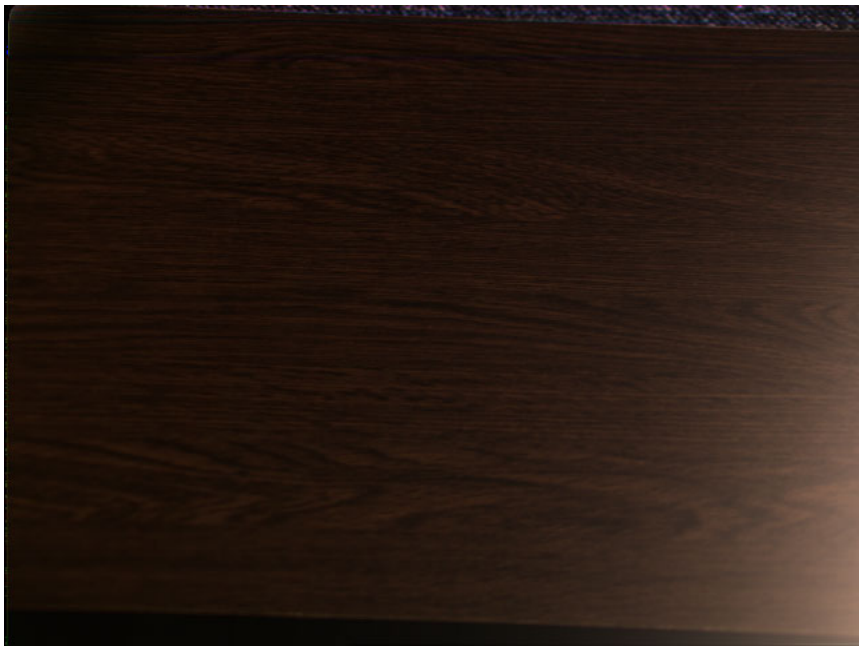


図 5.11: 生成した画像

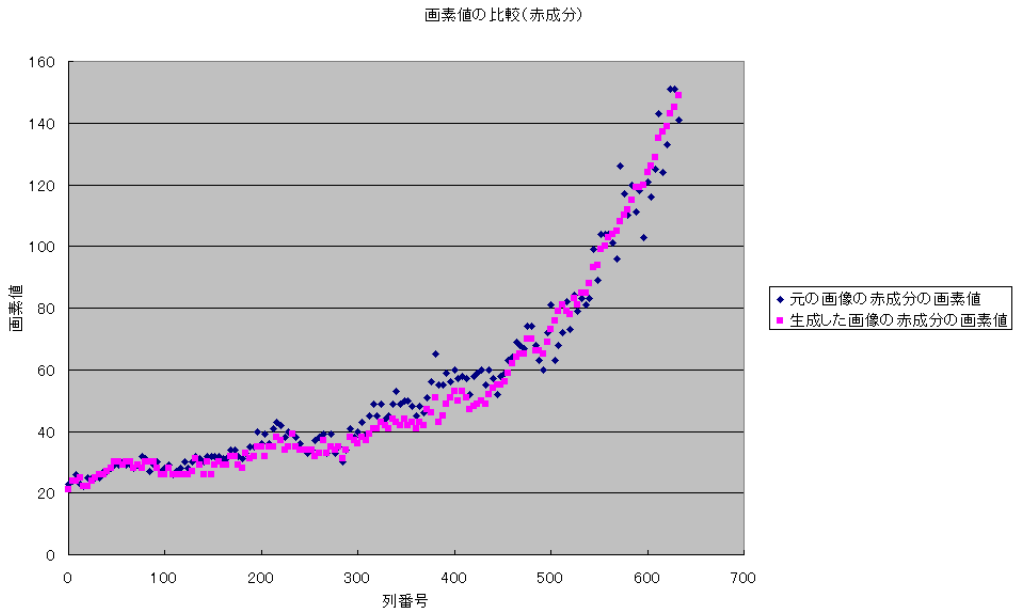


図 5.12: 画素値の比較 (赤成分)

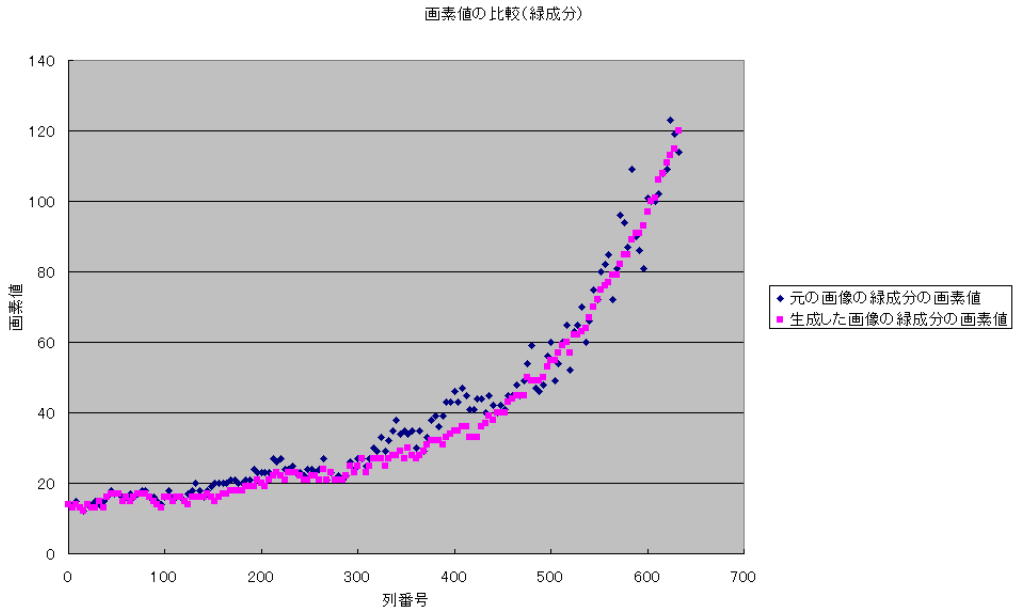


図 5.13: 画素値の比較 (緑成分)

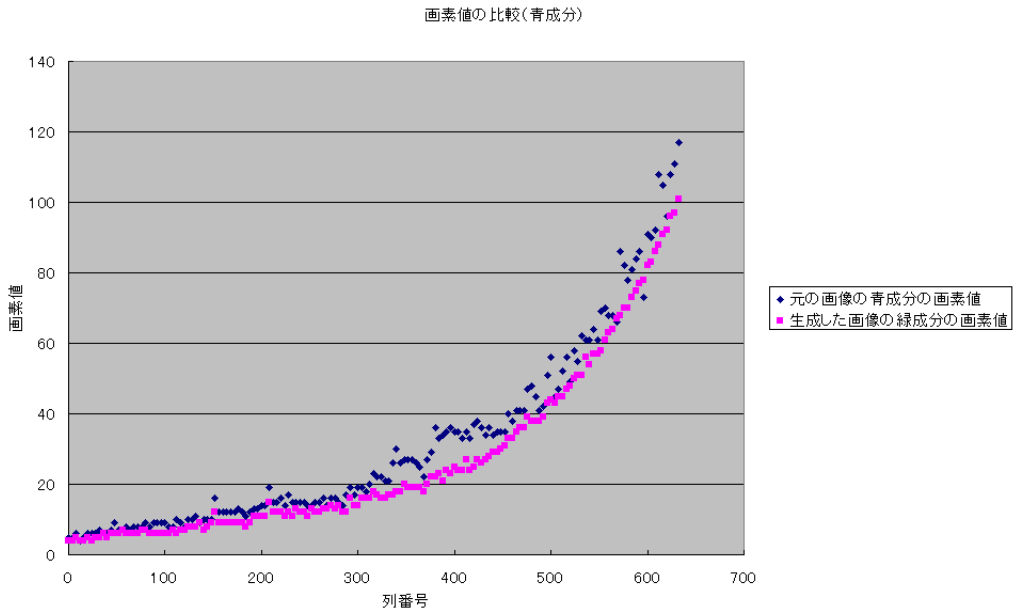


図 5.14: 画素値の比較 (青成分)

第6章　結論と今後の課題

6.1　結論

板などの平面物体において、光源、視点の 3 次元位置が既知の場合、偏光を用いて反射成分を分離し、Torrance-Sparrow 反射モデルにおける、反射パラメタ k、σ を実画像から正確に推定でき、正確に鏡面反射を生成することが出来た。

6.2　今後の課題

今後の課題としては以下の点があげられる。

- 板以外での物体 (表面法線方向が一定でない物体) での適用方法
- なめらかではない物体での適用方法
- 金属や透明物体といった特殊な反射特性を示す物体での適用方法

謝辞

はじめに、せっかく中西・斎藤研究室に入れて頂いたにもかかわらず、私のわがままにより東京大学生産技術研究所第 3 部池内研究室で研究をすることに対して賛成を頂き、いろいろと便宜を図って頂いた、斎藤博昭先生に心より感謝致します。

東京大学生産技術研究所第 3 部池内研究室では、私にはもったいないほどの研究環境を与えて頂き、さらに、研究について無知な私を、1 からご指導頂きました池内克史教授、佐藤洋一助教授に深くお礼申し上げます。また、研究室のすばらしい雰囲気を作り、分からないことだらけの私に、貴重な時間を裂いてまで様々な助言を頂いた諸先輩方、本当にありがとうございました。特に、高橋徹先輩、高橋拓二先輩、佐藤いまり先輩には、日頃から熱心にご指導、励ましのお言葉を頂きまして、言葉では表現できないほど感謝しています。先輩の存在がなかったら、この卒業論文は書けませんでした。かなりの時間を使わせてしまって本当にすいませんでした。

中西・斎藤研究室の諸先輩方と会うことは、めっきり減ってしまいましたが、プレゼミ・発表練習等多くのことでお世話になりました。ありがとうございました。B 4 のみんな、頭のいい人ばかりでいい刺激をありがとう。

そして、遠くから様々なことで気遣ってくれた母親と兄、そして、大きな心の支えになってくれた友人達にこの場を借りて御礼を言いたいと思います。

最後に、直接ご指導頂くことは少なかったのですが、去年の秋、急逝された中西正和先生には、たいへんお世話になりました。心よりご冥福お祈り申し上げます。

みんな、本当にありがとうございました。

2001 年
春

参 考 文 献

[1] 佐藤洋一, “コンピュータビジョンにおける最新動向：物理ベースドビジョン”, interLab, pp40-44, 1999.8

[2] 斎藤めぐみ, “偏光解析に基づく透明物体の形状モデリング”, 慶應義塾大学修士論文, 1999

[3] Y.sato, M.D.Wheeler,and K.Ikeuchi, “Object shape and reflectance modeling from observation”, Comp Graphics Proceedings, 379, 1997

[4] Shouji Tominaga, Norihiro Tanaka, “Estimating Reflection Parameters from a single Color Image”,

[5] L.B.Wolf, T.E Boult, “Constraining object features using a polarizationreflectance model”, IEEE Trans. Patt. Anal. Machine Intell, 13, 167, 1991

[6] 大津元一, “現代光科学”, 朝倉書店, 1994

[7] Shree K.Nayar, K. Ikeuchi, and T.Kanade, “Surface Reflection: Physical and Geometrical Perspectives”, IEEE Trance. Patt. Anal. Mach. Intell. 1990

[8] 浅田尚紀, “コンピュータビジョン　技術評論と将来展望”, 新技術コミュニケーションズ, pp37-41, 1998

[9] ALAN WATT, “3D COMPUTER GRAPHICS”,1994

付録

幾何的キャリブレーション
光学的キャリブレーション
外部仕様
内部仕様
ソースプログラム

付録 A 幾何的キャリブレーション

A-1　Tsaiの幾何学キャリブレーション

ここでは、画像座標に対応する、3次元座標の求める方法として、Tsaiのキャリブレーション手法 [?] を幾何学キャリブレーションとして説明する。

A-2　世界座標と画像座標の関係

図A.1に示すように、基準となる世界座標 $O_w - x_w - y_w - z_w$ における点 P の座標を (x_w, y_w, z_w) と表し、カメラ座標 $O - x - y - z$ における点 P の座標を (x, y, z) と表す。ただし、O はレンズ中心を表し、z 軸はレンズの光軸に一致するように設定する。

次に、$x - y$ 平面に平行で z 座標が f の位置に画像中心が O_i、座標軸が $X - Y$ の画像平面を考える。理想的なピンホールカメラの場合には、点 P の座標 (x, y, z) は画像平面上では (X_u, Y_u) と表せるが、レンズの幾何学的なひずみにより実際には画像平面上では (X_d, Y_d) の位置に対応するものとする。そして、(X_d, Y_d) のディジタル画像上での離散化された座標を (X_f, Y_f) と表す。

点 P の世界座標 (x_w, y_w, z_w) とその点の画像座標 (X_f, Y_f) の関係は以下の手順で導くことが出来る。

Step 1 (x_w, y_w, z_w) から (x, y, z) への変換:回転行列 R と平行移動ベクトル T を用いて表す。

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = T + R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \tag{A.1}$$

ただし、

$$R = \begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}, T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} \tag{A.2}$$

Step 2 透視変換による (x, y, z) から (X_u, Y_u) への変換:焦点距離 f を用いて表す。

$$X_u = f\frac{x}{z}, Y_u = f\frac{y}{z} \tag{A.3}$$

Step 3 (X_u, Y_u) から (X_d, Y_d) への変換:

$$X_d + D_x = X_u, Y_d + D_y = Y_u \tag{A.4}$$

ただし、D_x、D_y はレンズ半径方向のひずみ係数 k_1、k_2 を用いて次のように表す。

$$D_x = X_d(k_1 r^2 + k_2 r^4) \tag{A.5}$$

$$D_y = Y_d(k_1 r^2 + k_2 r^4) \tag{A.6}$$

$$r = \sqrt{X_d^2 + Y_d^2} \tag{A.7}$$

Step 4 (X_d, Y_d) から (X_f, Y_f) への変換:

$$X_f = s_x d'^{-1}_x X_d + C_x \tag{A.8}$$

$$Y_f = d_y^{-1} Y_d + C_y \tag{A.9}$$

ただし、s_x はスケール係数、(C_x, C_y) はディジタル画像上での原点座標、d_x、d_y はそれぞれ X 方向、Y 方向の CCD 素子の間隔を表す。なお、d'_x は X 方向の CCD 素子数 N_{cx} と 1 走査線のサンプル数 N_{fx} を用いて d_x を補正したものである。

$$d'_x = d_x \frac{N_{cx}}{N_{fx}} \tag{A.10}$$

以上の関係を整理すると、

$$X = X_f - C_x, Y = Y_f - C_y \tag{A.11}$$

を用いて、

$$\begin{aligned} & s_x^{-1} d'_x X + s_x^{-1} d'_x X (k_1 r^2 + k_2 r^4) \\ & = f \frac{r_1 x_w + r_2 y_w + r_3 z_w + T_x}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \end{aligned} \tag{A.12}$$

$$\begin{aligned} & d_y Y + d_y Y (k_1 r^2 + k_2 r^4) \\ & = f \frac{r_4 x_w + r_5 y_w + r_6 z_w + T_y}{r_7 x_w + r_8 y_w + r_9 z_w + T_z} \end{aligned} \tag{A.13}$$

が得られる。ただし、

$$r = \sqrt{(s_x^{-1} d'_x X)^2 + (d_y Y)^2} \tag{A.14}$$

である。

A-3　キャリブレーション手順

キャリブレーションは、多数の点の世界座標 (x_w, y_w, z_w) とそれらの点に対応する画像座標 (X_f, Y_f) の組が与えられたときに、外部パラメータ

としての回転行列 R と平行移動ベクトル T、内部パラメータとして焦点距離 f、レンズひずみ係数 k_1、k_2、スケール係数 s_x、画像原点 (C_x, C_y) の全部で 12 個のパラメータを求める問題になる。以下では、s_x を既知とした場合のキャリブレーション手順について述べる。

Step 1 同一平面上に存在する N 個のキャリブレーション点を撮影した画像から点 P_i の画像座標 (X_i, Y_i) を求める。

Step 2 カメラおよび A/D 変換器の仕様から N_{cx}、N_{fx}、D'_x、d_y を求める。

Step 3 画像の中央を (X_{di}, Y_{di}) とする。

Step 4 N 点の (X_{di}, Y_{di}) を求める。

$$X_{di} = s_x^{-1} d'_x (X_{fi} - C_x) \tag{A.15}$$

$$Y_{di} = d_y (Y_{fi} - C_y) \tag{A.16}$$

Step 5 (x_{wi}, y_{wi}, z_{wi}) と (X_{di}, Y_{di}) の組から $T_y^{-1} r_1$、$T_y^{-1} r_2$、$T_y^{-1} T_x$、$T_y^{-1} r_4$、$T_y^{-1} r_5$ を未知数とする線型方程式を解く。

$$\begin{bmatrix} Y_{di} x_{wi} \\ Y_{di} y_{wi} \\ Y_{di} \\ -Xdi x_{wi} \\ -Xdi y_{wi} \end{bmatrix}^T \begin{bmatrix} T_y^{-1} r_1 \\ T_y^{-1} r_2 \\ T_y^{-1} T_x \\ T_y^{-1} r_4 \\ T_y^{-1} r_5 \end{bmatrix} = Xdi \tag{A.17}$$

この式は、式 (3)、(4) から

$$X_{dy} - Y_{dx} = 0 \tag{A.18}$$

を導き、さらに $z_w = 0$ を代入することによって得られる。

Step 6 $T_y^{-1}r_1$、$T_y^{-1}r_2$、$T_y^{-1}T_x$、$T_y^{-1}r_4$、$T_y^{-1}r_5$ から T_y^2 を求める。まず、

$$C = \begin{bmatrix} r'_1 & r'_2 \\ r'_4 & r'_5 \end{bmatrix} = \begin{bmatrix} T_y^{-1}r_1 & T_y^{-1}r_2 \\ T_y^{-1}r_4 & T_y^{-1}r_5 \end{bmatrix} \tag{A.19}$$

とする。行列 C の行または列の要素が 0 でない場合は

$$T_y^2 = \frac{S_r - \sqrt{S_r^2 - 4(r'_1 r'_5 - r'_4 r'_2)^2}}{2(r'_1 r'_5 - r'_4 r'_2)^2} \tag{A.20}$$

を求める、ただし $S_r = r'^2_1 + r'^2_2 + r'^2_3 + r'^2_4 + r'^2_5$ とする。一方、行列 C の行または列の要素が 0 の場合は、

$$T_y^2 = (r'^2_i + r'^2_j)^{-1} \tag{A.21}$$

を求める。ただし、$r'_i r'_j$ は行列 C の 0 でない行または列の要素を表す。

Step 7 画像原点 (C_x, C_y) から十分離れた点 (X_{fi}, Y_{fi}) とその対応する点の世界座標 (x_{wi}, y_{wi}, z_{wi}) から T_y の符号を決定する。まず、T_y の符号を正として

$$\begin{aligned} r_1 &= (T_y^{-1}r_1)T_y \\ r_2 &= (T_y^{-1}r_2)T_y \\ r_4 &= (T_y^{-1}r_4)T_y \\ r_5 &= (T_y^{-1}r_5)T_y \\ T_x &= (T_y^{-1}T_x)T_y \\ x &= r_1 x_w + r_2 y_w + T_x \\ y &= r_4 x_w + r_5 y_w + T_y \end{aligned} \tag{A.22}$$

を求め、x と X の符号が等しく且つ y と Y の符号が等しいなら T_y の符号を正、それ以外では T_y の符号を負とする。

Step 8 r_1、r_2、r_4、r_5から回転行列Rを決定する。

$$R = \begin{bmatrix} r_1 & r_2 & \sqrt{1-r_1^2-r_2^2} \\ r_4 & r_5 & s\sqrt{1-r_4^2-r_5^2} \\ r_7 & r_8 & r9 \end{bmatrix} \tag{A.23}$$

ただし、$s = -sgn(r_1 r_4 + r_2 r_5)$である。$r_7$、$r_8$、$r_9$は$R$が直行行列であるという性質を用いて求める。なお、Step9で求める焦点距離fが負となる場合は、

$$R = \begin{bmatrix} r_1 & r_2 & -\sqrt{1-r_1^2-r_2^2} \\ r_4 & r_5 & -s\sqrt{1-r_4^2-r_5^2} \\ -r_7 & -r_8 & r9 \end{bmatrix} \tag{A.24}$$

を用いる。

Step 9 レンズひずみを無視$(k_1 = k_2 = 0)$としてfとT_zの初期値を求める。N点のキャリブレーション点を用いてfとT_zを未知数とする線型方程式を解く。

$$\begin{bmatrix} y_i - d_y Y_i \end{bmatrix} \begin{bmatrix} f \\ T_z \end{bmatrix} = w_i d_y Y_i \tag{A.25}$$

ただし、

$$\begin{aligned} y_i &= r_4 x_{wi} + r_5 y_{wi} + T_y \\ w_I &= r_7 x_{wi} + r_8 y_{wi} \end{aligned} \tag{A.26}$$

Step 10 Step9で求めたfとT_zおよび$k_1 = k_2 = 0$を初期値として、式(12)を非線型最適化問題として解き、f、T_z、k_1、k_2を求める。

以上の方法により、未知数を計算し、世界座標と画像座標の対応を決定する。

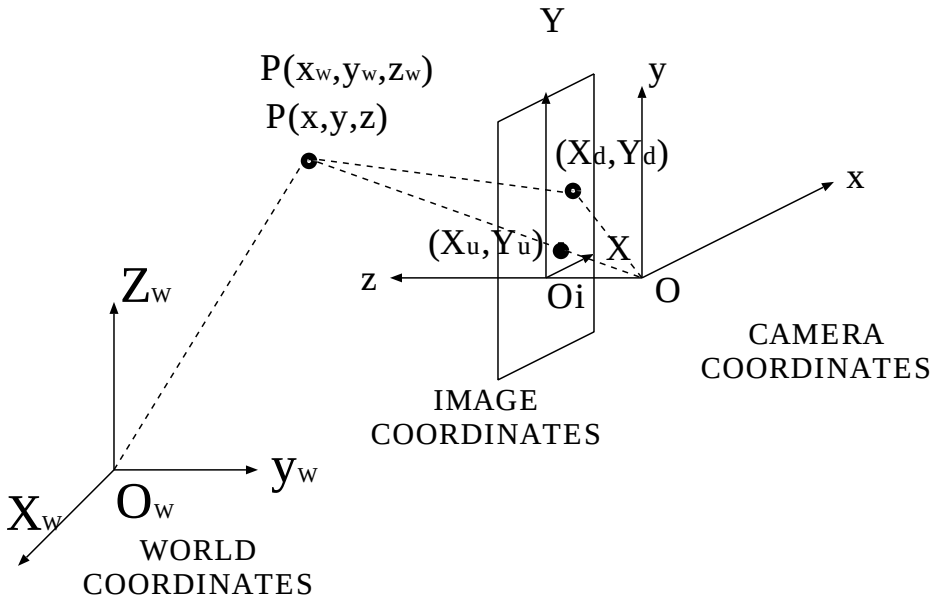


図 A.1: 世界座標と画像座標の関係

付録 B 光学的キャリブレーション

B-1　光学的キャリブレーション

まず、光学的キャリブレーションとは偏光板とカメラのレンズの光の干渉を補正することである。

例えば、反射成分が拡散反射成分のみであるという仮定できる物体 (具体的にはざらざらした画用紙など) を偏光板を回しながら撮影する。2 章でも述べたように、拡散反射成分というのは視点、光源が一定ならば常に一定であるはずのため、偏光板をまわしても画像の画素輝度値は一定のはずである。ところが、実際には偏光板とカメラのレンズの光学的な干渉のために画素輝度値が変化してしまう。

本研究で用いた、干渉による画素輝度値を変化を補正する手法を述べる。

B-2　キャリブレーション手法

まず、反射成分が拡散反射成分のみであると仮定できる物体を、実際の研究で行うのと同じ実験環境下で同じ偏光板の角度の対応をとって撮影する。これをキャリブレーション用画像という。このとき、最初に撮った画像の偏光板の角度を 0 度とする。さらに、キャリブレーション用画像と、実際に研究を行う物体を撮影した原画像の画像座標 (x, y) の画素輝度値をそれぞれ $C_0[x][y]$、$R_0[x][y]$ とし、n 度偏光板を回したときの画素輝度値を $C_n[x][y]$、$R_n[x][y]$ とする。

ここで、キャリブレーション用画像において n 度回したときに画像座標 (x, y) のときの画素輝度値は $C_n[x][y]/C_0[x][y]$ 倍となってしまうため、以下の式で表すように補正する。

$$H_n[x][y] = R_n[x][y]\frac{C_0[x][y]}{C_n[x][y]} \tag{B.1}$$

ただし、$H_n[x][y]$は補正後の画素輝度値を表す。

この変換をすべてのピクセルにおいて行い、偏光板とカメラのレンズの干渉を補正する。

付録 C 外部仕様

C-1　separate.cpp

C-1.1　実行

Windows 上で『separate.exe』を開き実行する。

C-1.2　入力ファイル

- 反射成分

  鏡面反射成分と拡散反射成分が含まれる (画像) ビットマップファイル

- 拡散反射成分

  拡散反射成分のみ含まれる (画像) ビットマップファイル

C-1.3　出力ファイル

- 鏡面反射成分

  鏡面反射成分のみ含まれる (画像) ビットマップファイル

C-2　kakudo.cpp

C-2.1　実行

Windows 上で『kakdo.exe』を開き実行する。

C-2.2　入力ファイル

- 各ピクセルの 3 次元座標

  各ピクセルの 3 次元座標が入っているテキストファイル (表 C.1)

C-2.3　出力ファイル

- 各ピクセルにおける角度

  各ピクセルにおける角度 α，θ の入っているテキストファイル (表 C.2)

C-3　parameta.cpp

C-3.1　実行

Windows 上で『parameta.exe』を開き実行する。

C-3.2　入力ファイル

- 鏡面反射成分

  separate.cpp で分離した鏡面反射成分のみの (画像) ビットマップファイル

- 各ピクセルにおける角度

  kakudo.cpp で決定した各ピクセルの角度 α，θ の入っているテキストファイル (表 C.2)

C-3.3　出力ファイル

- パラメタ決定のための値

  各ピクセルの $\alpha^2/2$、$lnI + In\cos\theta$ が入っているテキストファイル (表 C.3)

表 C.1: 各ピクセルの 3 次元座標が入っているテキストファイルの例

-67.520702 259.752880
-67.094175 259.777850
-66.667515 259.802787
-66.240722 259.827691
-65.813797 259.852562
-65.386740 259.877400
-64.959552 259.902205
-64.532233 259.926977
-64.104783 259.951715
-63.677203 259.976421
-63.249493 260.001093
-62.821653 260.025733
-62.393683 260.050339
-61.965585 260.074912
-61.537358 260.099452
-61.109003 260.123958
-60.680519 260.148432
-60.251908 260.172872
-59.823170 260.197279
-59.394305 260.221653
-58.965314 260.245994
-58.536196 260.270301

表 C.2: 各ピクセルにおける角度α, θの入っているテキストファイルの例

0.628032 0.632593
0.627837 0.632591
0.627643 0.632590
0.627448 0.632589
0.627254 0.632587
0.627059 0.632586
0.626864 0.632584
0.626669 0.632583
0.626474 0.632581
0.626279 0.632580
0.626084 0.632578
0.625888 0.632577
0.625693 0.632575
0.625497 0.632573
0.625302 0.632571
0.625106 0.632570
0.624910 0.632568
0.624714 0.632566
0.624518 0.632564
0.624322 0.632562
0.624126 0.632560
0.623930 0.632558

表 C.3: 各ピクセルの $\alpha^2/2$、$lnI + Incos\,\theta$ が入っているテキストファイルの例

0.092144 2.171728
0.091933 2.084789
0.091828 2.084824
0.091723 2.171872
0.091618 2.251950
0.091512 2.251986
0.091407 2.326131
0.091302 2.395160
0.091197 2.395196
0.091092 2.520396
0.090987 2.631658
0.090882 2.459844
0.090778 2.459881
0.090673 2.683061
0.090569 2.631804
0.090464 2.731925
0.090360 2.778481
0.090255 2.731999
0.090151 2.577885
0.090047 2.906426
0.089942 3.054883
0.089838 3.121612

C-4　seisei.cpp

C-4.1　実行

Windows 上で『seisei.exe』を開き実行する。

C-4.2　入力ファイル

- 各ピクセルにおける角度

  kakudo.cpp で決定した各ピクセルの角度 α，θ の入っているテキストファイル (表 C.2)

- 拡散反射成分

  拡散反射成分のみの (画像) ビットマップファイル

C-4.3　出力ファイル

- 生成した画像

  反射モデルに当てはめ、生成した (画像) ビットマップファイル

C-5　hyouka.cpp

C-5.1　実行

Windows 上で『hyouka.exe』を開き実行する。

C-5.2　入力ファイル

- 元の画像

  鏡面反射成分と拡散反射成分が含まれる (画像) ビットマップファイル

- 生成した画像

  seisei.cpp で生成した (画像) ビットマップファイル

C-5.3　出力ファイル

- ある一列における元の画像と、生成した画像の画素輝度値

  ある一列における元の画像と、生成した画像の画素輝度値の入ったテキストファイル (表 C.4)

表 C.4: ある一列における元の画像と、生成した画像の画素輝度値の入ったテキストファイルの例

4 25 21
8 24 24
12 24 24
16 25 25
20 24 22
24 24 22
28 25 24
32 25 25
36 24 26
40 24 26
44 25 27
48 25 28
52 25 30
56 24 30
60 23 29
64 25 30
68 24 30
72 25 28
76 25 29
80 24 28
84 24 30
88 24 30
92 23 30
96 23 28
100 24 26
104 24 26
108 24 28
112 25 26
116 24 26
120 25 26
124 23 26
128 24 26

付録 D 内部仕様

D-1 separate.cpp

D-1.1 主な変数・配列・構造体

BITMAPFILEHEADER	ビットマップのファイルヘッダー
BITMAPINFOHEADER	ビットマップのインフォヘッダー
FILE *fp_min	拡散反射成分のみの画像へのファイルポインタ
FILE *fp_max	2つの反射成分を含む画像へのファイルポインタ
FILE *fp_sar_ref	取り出した鏡面反射成分の画像へのファイルポインタ
unsigned char *min_r	拡散反射成分の赤成分
unsigned char *min_g	拡散反射成分の緑成分
unsigned char *min_b	拡散反射成分の青成分
unsigned char *max_r	2つの反射成分の赤成分
unsigned char *max_g	2つの反射成分の緑成分
unsigned char *max_b	2つの反射成分の青成分
unsigned char *sar_ref_r	鏡面反射成分の赤成分
unsigned char *sar_ref_g	鏡面反射成分の緑成分
unsigned char *sar_ref_b	鏡面反射成分の青成分

D-1.2 主要関数

```
void file_headear_write(FILE *fp,BITMAPFILEHEADER *head)
```

引数　書き込むファイル名のファイルポインタ，BITMAPFILE-HEADER へのポインタ

返戻値　なし

機能　書き込みたいビットマップファイルのファイルヘッダーを書く

```
void info_header_write(FILE *fp,BITMAPINFOHEADER *head)
```

引数　書き込むファイル名のファイルポインタ，BITMAPIN-FOHEADER へのポインタ

返戻値　なし

機能　書き込みたいビットマップファイルのインフォヘッダー書く

D-2 kakudo.cpp

D-2.1 主な変数・配列

FILE *fp_tmp	3 次元座標の入ったファイルへのファイルポインタ
FILE *fp_out	角度の入ったファイルへのファイルポインタ
COL	ピクセルの横幅
ROW	ピクセルの縦幅
double **X	fp_tmp から得られた 3 次元座標の X 成分を指すポインタ
double **Y	fp_tmp から得られた 3 次元座標の Y 成分を指すポインタ
float LIGHT_X	光源の 3 次元 X 座標
float LIGHT_Y	光源の 3 次元 Y 座標
float LIGHT_Z	光源の 3 次元 Z 座標
float CAMERA_X	カメラの 3 次元 X 座標
float CAMERA_Y	カメラの 3 次元 Y 座標
float CAMERA_Z	カメラの 3 次元 Z 座標
double **camera_vector_x	各ピクセルからカメラへのベクトルの X 成分
double **camera_vector_y	各ピクセルからカメラへのベクトルの Y 成分
double **camera_vector_z	各ピクセルからカメラへのベクトルの Z 成分
double **light_vector_x	各ピクセルから光源へのベクトルの X 成分
double **light_vector_y	各ピクセルから光源へのベクトルの Y 成分
double **light_vector_z	各ピクセルから光源へのベクトルの Z 成分
double **nitoubun_vector_x	カメラと光源の 2 等分ベクトルの X 成分
double **nitoubun_vector_y	カメラと光源の 2 等分ベクトルの Y 成分
double **nitoubun_vector_z	カメラと光源の 2 等分ベクトルの Z 成分
double **sita	各ピクセルの角度 θ
double **arufa	各ピクセルの角度 α

D-3 parameta.cpp

D-3.1 主な変数・配列・構造体

BITMAPFILEHEADER	ビットマップのファイルヘッダー
BITMAPINFOHEADER	ビットマップのインフォヘッダー
FILE *fp_ref	取り出した鏡面反射成分へのファイルポインタ
FILE *fp_parameta_r	出力用ファイルへのファイルポインタ
FILE *fp_parameta_g	出力用ファイルへのファイルポインタ
FILE *fp_parameta_b	出力用ファイルへのファイルポインタ
FILE *fp_out	角度の入っているファイルへのファイルポインタ
FILE *fp_sar_ref	取り出した鏡面反射成分の画像へのファイルポインタ
double **sita	各ピクセルの角度 θ
double **arufa	各ピクセルの角度 α
double *X_r	赤成分のパラメタ推定用 X 成分の値をいれる
double *X_g	緑成分のパラメタ推定用 X 成分の値をいれる
double *X_b	青成分のパラメタ推定用 X 成分の値をいれる
double *Y_r	赤成分のパラメタ推定用 Y 成分の値をいれる
double *Y_g	緑成分のパラメタ推定用 Y 成分の値をいれる
double *Y_b	青成分のパラメタ推定用 Y 成分の値をいれる
unsigned char *ref_r	鏡面反射成分の赤成分
unsigned char *ref_g	鏡面反射成分の緑成分
unsigned char *ref_b	鏡面反射成分の青成分

D-4　seisei.cpp

D-4.1　主な変数・配列・構造体

BITMAPFILEHEADER	ビットマップのファイルヘッダー
BITMAPINFOHEADER	ビットマップのインフォヘッダー
FILE *fp_s_min	拡散反射成分のみの画像へのファイルポインタ
FILE *fp_s_ref	生成した鏡面反射成分の画像へのファイルポインタ
FILE *fp_s_seisei	生成した画像へのファイルポインタ
FILE *fp_out	角度の入っているファイルへのファイルポインタ
unsigned char *s_min_r	拡散反射成分の赤成分
unsigned char *s_min_g	拡散反射成分の緑成分
unsigned char *s_min_b	拡散反射成分の青成分
unsigned char *s_ref_r	生成した鏡面反射成分の赤成分
unsigned char *s_ref_g	生成した鏡面反射成分の緑成分
unsigned char *s_ref_b	生成した鏡面反射成分の青成分
unsigned char *seisei_r	生成した画像の赤成分
unsigned char *seisei_g	生成した画像の緑成分
unsigned char *seisei_b	生成した画像の青成分
double **sita	各ピクセルの角度 θ
double **arufa	各ピクセルの角度 α

D-4.2　主要関数

```
void file_headear_write(FILE *fp,BITMAPFILEHEADER *head)
```

引数　書き込むファイル名のファイルポインタ，BITMAPFILE-HEADER へのポインタ
返戻値　なし
機能　書き込みたいビットマップファイルのファイルヘッダーを書く

`void info_header_write(FILE *fp,BITMAPINFOHEADER *head)`

引数　書き込むファイル名のファイルポインタ，BITMAPIN-FOHEADER へのポインタ
返戻値　なし
機能　書き込みたいビットマップファイルのインフォヘッダー書く

D-5　hyouka.cpp

D-5.1　主な変数・配列

`FILE *fp_s_seisei`	生成した画像へのファイルポインタ
`FILE *fp_hyouka_r`	評価する赤成分の入った出力ファイルへのポインタ
`FILE *fp_hyouka_g`	評価する緑成分の入った出力ファイルへのポインタ
`FILE *fp_hyouka_b`	評価する青成分の入った出力ファイルへのポインタ
`unsigned char *moto_r`	もとの画像の赤成分
`unsigned char *moto_g`	もとの画像の緑成分
`unsigned char *moto_b`	もとの画像の青成分
`unsigned char *seisei_r`	生成した画像の赤成分
`unsigned char *seisei_g`	生成した画像の緑成分
`unsigned char *seisei_b`	生成した画像の青成分

付録 E ソースコード

E-1　separate.cpp

```
////////////////////////////////////////////
//２つの画像 (ビットマップ) から
//その差分の画像を生成
////////////////////////////////////////////
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/*BITMAP FILE HEADER の宣言*/
typedef struct tagBITMAPFILEHEADER {

        unsigned short bfType;
unsigned long  bfSize;
unsigned short bfReserved1;
unsigned short bfReserved2;
unsigned long  bfOffBits;

}BITMAPFILEHEADER;

/*BITMAP INFO HEADER の宣言*/
typedef struct tagBITMAPINFOHEADER{

        unsigned long  biSize;
signed long    biWidth;
signed long    biHeight;
unsigned short biPlanes;
unsigned short biBitCount;
unsigned long  biCompression;
unsigned long  biSizeImage;
signed long    biXPelsPerMeter;
signed long    biYPelsPerMeter;
unsigned long  biClrUsed;
unsigned long  biClrImportant;

} BITMAPINFOHEADER;
```

```
void file_header_write(FILE*,BITMAPFILEHEADER *);//BITMAP FILE HEADER を
書き込む関数

void info_header_write(FILE*,BITMAPINFOHEADER *);//BITMAP INFO HEADER
を書き込む関数

void file_n_byte_output(FILE*,unsigned long,int);//出力のための関数

main()
{
FILE *fp_min;//minを読み込む際のファイルポインタ
FILE *fp_max;//maxを読み込む際のファイルポインタ
FILE *fp_sar_ref;//生成した画像のファイルポインタ
FILE *fp_min_test;//minが読めているかどうかテストするためのファイルポインタ
FILE *fp_max_test;//maxが読めているかどうかテストするためのファイルポインタ

BITMAPFILEHEADER fileheader;
BITMAPINFOHEADER infoheader;
BITMAPFILEHEADER fileheader2;
BITMAPINFOHEADER infoheader2;

int i=0; //カウンタ用
int j=0; //カウンタ用
int k=0; //カウンタ用
        int m=0; //カウンタ用
int n=0; //カウンタ用
int kaisuu=0; //カウンタ用
int satiration=0; //カウンタ用


unsigned char *max;//ここにまず取り込む（極大）
unsigned char *min;//ここにも取り込む（極小）
        unsigned char *sar_ref;//（極大）ー（極小）＝　（鏡面反射成分）

unsigned char *min_r;
unsigned char *min_g;
unsigned char *min_b;

unsigned char *max_r;
unsigned char *max_g;
unsigned char *max_b;

unsigned char *sar_ref_r;
unsigned char *sar_ref_g;
unsigned char *sar_ref_b;

// unsigned char xy[480][640];//生成した画像情報の実体部分を入れる配列
// unsigned char xy_max_test[480][640];//取り込んだmaxの画像情報の実体部
```

分を入れる配列
// unsigned char xy_min_test[480][640];//取り込んだ min の画像情報の実体部
分を入れる配列

```
max = (unsigned char*)malloc(sizeof(unsigned char)*921600);
min = (unsigned char*)malloc(sizeof(unsigned char)*921600);
sar_ref = (unsigned char*)malloc(sizeof(unsigned char)*921600);

min_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
min_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
min_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);

max_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
max_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
max_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);

sar_ref_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
sar_ref_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
sar_ref_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);

/*１枚目の写真の読み込み*/

if((fp_min=fopen("result1.bmp","rb"))==NULL){
printf("File open error 1 \n");
return 0;
}

/*１枚目の写真のヘッダーの読み込み*/
if(fread(&fileheader.bfType,2,1,fp_min)!=1){
printf("read error1");
}

printf("fileheader.bfType=%u\n",fileheader.bfType);

if(fread(&fileheader.bfSize,4,1,fp_min)!=1){
printf("read error2");
}

printf("fileheader.bfSize=%u\n",fileheader.bfSize);

if(fread(&fileheader.bfReserved1,2,1,fp_min)!=1){
printf("read error3");
}

printf("fileheader.bfReserved1=%u\n",fileheader.bfReserved1);

if(fread(&fileheader.bfReserved2,2,1,fp_min)!=1){
printf("read error4");
}
```

```
printf("fileheader.bfReserved2=%u\n",fileheader.bfReserved2);

if(fread(&fileheader.bfOffBits,4,1,fp_min)!=1){
printf("read error5");
}

printf("fileheader.bfOffBits=%u\n",fileheader.bfOffBits);

if(fread(&infoheader.biSize,4,1,fp_min)!=1){
  printf("read error6");
}

printf("infoheader.biSize=%u\n",infoheader.biSize);

if(fread(&infoheader.biWidth,4,1,fp_min)!=1){
printf("read error7");
}

printf("infoheader.biWidth=%u\n",infoheader.biWidth);

if(fread(&infoheader.biHeight,4,1,fp_min)!=1){
printf("read error8");
}

printf("infoheader.biHeight=%u\n",infoheader.biHeight);

if(fread(&infoheader.biPlanes,2,1,fp_min)!=1){
printf("read error9");
}

printf("infoheader.biPlanes=%u\n",infoheader.biPlanes);

if(fread(&infoheader.biBitCount,2,1,fp_min)!=1){
printf("read error10");
}

printf("infoheader.biBitCount=%d\n",infoheader.biBitCount);

if(fread(&infoheader.biCompression,4,1,fp_min)!=1){
printf("read error11");
}

printf("infoheader.biCompression=%u\n",infoheader.biCompression);

if(fread(&infoheader.biSizeImage,4,1,fp_min)!=1){
printf("read error12");
}
```

```
printf("infoheader.biSizeImage=%u\n",infoheader.biSizeImage);

if(fread(&infoheader.biXPelsPerMeter,4,1,fp_min)!=1){
printf("read error13");
}

printf("infoheader.biXPelsPerMeter=%u\n",infoheader.biXPelsPerMeter);

if(fread(&infoheader.biYPelsPerMeter,4,1,fp_min)!=1){
printf("read error14");
}

printf("infoheader.biYPelsPerMeter=%u\n",infoheader.biYPelsPerMeter);


if(fread(&infoheader.biClrUsed,4,1,fp_min)!=1){
printf("read error15");
}

printf("infoheader.biClrUsed=%u\n",infoheader.biClrUsed);

if(fread(&infoheader.biClrImportant,4,1,fp_min)!=1){
printf("read error16");
}

printf("infoheader.biClrImportant=%u\n",infoheader.biClrImportant);


/*１枚目の写真の中身の読み込み*/
i=0; //カウンタの初期化
while((fread(&min[i],sizeof(unsigned char),1,fp_min))==1){
i++;
}

i=0;    //カウンタの初期化
for(k=0;k<307200;k++){

min_b[k]=min[i];
i++;
min_g[k]=min[i];
i++;
min_r[k]=min[i];
i++;

}

/*出力用ファイル  min_test  の OPEN*/
if((fp_min_test = fopen("min_test2.bmp", "wb")) == NULL){
```

```
printf("FILE min_test.bmp cant't open.\n");

return 0;

    }

/*出力用ファイル min_test のヘッダーへの書き込み*/
file_header_write(fp_min_test,&fileheader);
info_header_write(fp_min_test,&infoheader);

/*出力用ファイル min_test の中身へ書き込み*/
for(i=0;i<307200;i++)
{
if((fwrite(&min_b[i],sizeof(unsigned char),1,fp_min_test))!=1){
printf("min_test.bmp write error\n");
}

if((fwrite(&min_g[i],sizeof(unsigned char),1,fp_min_test))!=1){
printf("min_test.bmp write error\n");
}

if((fwrite(&min_r[i],sizeof(unsigned char),1,fp_min_test))!=1){
printf("min_test.bmp write error\n");
}

}

printf("b=%u\n",min_b[44430]);
printf("g=%u\n",min_g[44430]);
printf("r=%u\n",min_r[44430]);

/*2 枚目の写真の読み込み*/
if((fp_max=fopen("result0.bmp","rb"))==NULL){
printf("File open error 2 \n");
return 0;
}

/*2 枚目の写真のヘッダーの読み込み*/
if(fread(&fileheader2.bfType,2,1,fp_max)!=1){
printf("read error17");
}

printf("fileheader2.bfType=%u\n",fileheader2.bfType);

if(fread(&fileheader2.bfSize,4,1,fp_max)!=1){
printf("read erro18");
}

printf("fileheader2.bfSize=%u\n",fileheader2.bfSize);
```

```
if(fread(&fileheader2.bfReserved1,2,1,fp_max)!=1){
printf("read error19");
}

printf("fileheader2.bfReserved1=%u\n",fileheader2.bfReserved1);

if(fread(&fileheader2.bfReserved2,2,1,fp_max)!=1){
printf("read error20");
}

printf("fileheader2.bfReserved2=%u\n",fileheader2.bfReserved2);

if(fread(&fileheader2.bfOffBits,4,1,fp_max)!=1){
printf("read erro21");
}

printf("fileheader2.bfOffBits=%u\n",fileheader2.bfOffBits);

if(fread(&infoheader2.biSize,4,1,fp_max)!=1){
printf("read erro22");
}

printf("infoheader2.biSize=%u\n",infoheader2.biSize);

if(fread(&infoheader2.biWidth,4,1,fp_max)!=1){
printf("read error23");
}

printf("infoheader2.biWidth=%u\n",infoheader2.biWidth);

if(fread(&infoheader2.biHeight,4,1,fp_max)!=1){
printf("read error24");
}

printf("infoheader2.biHeight=%u\n",infoheader2.biHeight);

if(fread(&infoheader2.biPlanes,2,1,fp_max)!=1){
printf("read error25");
}

printf("infoheader2.biPlanes=%u\n",infoheader2.biPlanes);

if(fread(&infoheader2.biBitCount,2,1,fp_max)!=1){
printf("read error26");
}

printf("infoheader2.biBitCount=%d\n",infoheader2.biBitCount);
```

```
if(fread(&infoheader2.biCompression,4,1,fp_max)!=1){
printf("read error27");
}

printf("infoheader2.biCompression=%u\n",infoheader2.biCompression);

if(fread(&infoheader2.biSizeImage,4,1,fp_max)!=1){
printf("read error28");
}

printf("infoheader2.biSizeImage=%u\n",infoheader2.biSizeImage);

if(fread(&infoheader2.biXPelsPerMeter,4,1,fp_max)!=1){
printf("read error29");
}

printf("infoheader2.biXPelsPerMeter=%u\n",infoheader2.biXPelsPerMeter);

if(fread(&infoheader2.biYPelsPerMeter,4,1,fp_max)!=1){
printf("read error30");
}

printf("infoheader2.biYPelsPerMeter=%u\n",infoheader2.biYPelsPerMeter);


if(fread(&infoheader2.biClrUsed,4,1,fp_max)!=1){
printf("read error31");
}

printf("infoheader2.biClrUsed=%u\n",infoheader2.biClrUsed);

if(fread(&infoheader2.biClrImportant,4,1,fp_max)!=1){
printf("read error32");
}

printf("infoheader2.biClrImportant=%u\n",infoheader2.biClrImportant);

/*2 枚目の写真の中身の読み込み*/

i=0;//カウンタの初期化
while((fread(&max[i],sizeof(unsigned char),1,fp_max))==1){
i++;
}

i=0;    //カウンタの初期化
for(k=0;k<307200;k++){

max_b[k]=max[i];
i++;
```

```
max_g[k]=max[i];
i++;
max_r[k]=max[i];
i++;

}

/*出力用ファイル　max_test　の OPEN*/
if((fp_max_test = fopen("max_test2.bmp", "wb")) == NULL){

printf("FILE max_test.bmp cant't open.\n");

return 0;

    }

/*出力用ファイル max_test のヘッダーへの書き込み*/
file_header_write(fp_max_test,&fileheader2);
info_header_write(fp_max_test,&infoheader2);

/*出力用ファイル max_test の中身へ書き込み*/
for(i=0;i<307200;i++)
{
if((fwrite(&max_b[i],sizeof(unsigned char),1,fp_max_test))!=1){
printf("max_test.bmp write error\n");
}

if((fwrite(&max_g[i],sizeof(unsigned char),1,fp_max_test))!=1){
printf("max_test.bmp write error\n");
}

if((fwrite(&max_r[i],sizeof(unsigned char),1,fp_max_test))!=1){
printf("max_test.bmp write error\n");
}

}


/*ここで鏡面反射成分を取り出す。*/
for(i=0;i<307200*3;i++){

if(max[i]<min[i]){
sar_ref[i]=0;
kaisuu++;
}

else {sar_ref[i] = max[i]-min[i];}

}
```

```
printf("min>maxは [%d] 回です。\n",kaisuu);
printf("サチレーションは [%d] 回です。\n",satiration);

i=0;    //カウンタの初期化

for(k=0;k<307200;k++){

sar_ref_b[k]=sar_ref[i];
i++;
sar_ref_g[k]=sar_ref[i];
i++;
sar_ref_r[k]=sar_ref[i];
i++;
}

/*出力用ファイル  sar_ref の OPEN*/
if((fp_sar_ref = fopen("sar_ref2.bmp", "wb")) == NULL){

printf("FILE sar_ref.bmp cant't open.\n");

return 0;

    }

/*出力用ファイル sar_ref_filename のヘッダーへの書き込み*/
file_header_write(fp_sar_ref,&fileheader);
info_header_write(fp_sar_ref,&infoheader);

/*出力用ファイル sar_ref の中身へ書き込み*/
for(i=0;i<307200;i++)
{
if((fwrite(&sar_ref_b[i],sizeof(unsigned char),1,fp_sar_ref))!=1){
printf("sar_ref_b.bmp write error\n");
}

if((fwrite(&sar_ref_g[i],sizeof(unsigned char),1,fp_sar_ref))!=1){
printf("sar_ref_g.bmp write error\n");
}

if((fwrite(&sar_ref_r[i],sizeof(unsigned char),1,fp_sar_ref))!=1){
printf("sar_ref_r.bmp write error\n");
}

}


for(i=0;i<307200;i++)
{
```

```
if((fwrite(&max_b[i],sizeof(unsigned char),1,fp_max_test))!=1){
printf("max_test.bmp write error\n");
}

if((fwrite(&max_g[i],sizeof(unsigned char),1,fp_max_test))!=1){
printf("max_test.bmp write error\n");
}

if((fwrite(&max_r[i],sizeof(unsigned char),1,fp_max_test))!=1){
printf("max_test.bmp write error\n");
}

}

fclose(fp_min);
fclose(fp_min_test);
fclose(fp_max);
fclose(fp_max_test);
fclose(fp_sar_ref);

return 0;
}

void file_header_write(FILE *fp,BITMAPFILEHEADER *head)

{

    file_n_byte_output(fp,head->bfType,2);

    file_n_byte_output(fp,head->bfSize,4);

    file_n_byte_output(fp,head->bfReserved1,2);

    file_n_byte_output(fp,head->bfReserved2,2);

    file_n_byte_output(fp,head->bfOffBits,4);

}



void info_header_write(FILE *fp,BITMAPINFOHEADER *head)

{

    file_n_byte_output(fp,head->biSize,4);

    file_n_byte_output(fp,head->biWidth,4);
```

```
    file_n_byte_output(fp,head->biHeight,4);
    file_n_byte_output(fp,head->biPlanes,2);
    file_n_byte_output(fp,head->biBitCount,2);
    file_n_byte_output(fp,head->biCompression,4);
    file_n_byte_output(fp,head->biSizeImage,4);
    file_n_byte_output(fp,head->biXPelsPerMeter,4);
    file_n_byte_output(fp,head->biYPelsPerMeter,4);
    file_n_byte_output(fp,head->biClrUsed,4);
    file_n_byte_output(fp,head->biClrImportant,4);
}

void file_n_byte_output(FILE *fp,unsigned long number, int mojisuu)
{
    int i;
for (i = 0; i < mojisuu; i++){
fputc((int)(number % 256), fp);
number = number / 256;
    }
}
```

E-2　kakudo.cpp

```
//////////////////////////////////////////
//tmp ファイルからワールド座標を読み込んで,
//各ピクセルにおけるα、θを求める。
//out にα、θを出力する。
//////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define COL 640 //ピクセルの縦の長さ
#define ROW 480 //ピクセルの横の長さ

int main(){

FILE *fp_tmp; //ワールド座標の入ったファイルへのポインタ
FILE *fp_out; //出力用ファイル out へのファイルポインタ。α、θの順で書き出さ
れる。

int i=0; //カウンタ用
int j=0; //カウンタ用

double **X; //渡されるワールド X 座標を入れる
double **Y; //渡されるワールド Y 座標を入れる

X = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
X[i] = (double*)malloc(sizeof(double)*COL);

Y = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
Y[i] = (double*)malloc(sizeof(double)*COL);

double **camera_vector_x; //カメラの方向ベクトルｘ成分
double **camera_vector_y; //カメラの方向ベクトルｙ成分
double **camera_vector_z; //カメラの方向ベクトルｚ成分

camera_vector_x = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
camera_vector_x[i] = (double*)malloc(sizeof(double)*COL);

camera_vector_y = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
camera_vector_y[i] = (double*)malloc(sizeof(double)*COL);

camera_vector_z = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
```

```
camera_vector_z[i] = (double*)malloc(sizeof(double)*COL);

double **light_vector_x; //照明方向ベクトルｘ成分
double **light_vector_y; //照明方向ベクトルｙ成分
double **light_vector_z; //照明方向ベクトルｚ成分

light_vector_x = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
light_vector_x[i] = (double*)malloc(sizeof(double)*COL);

light_vector_y = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
light_vector_y[i] = (double*)malloc(sizeof(double)*COL);

light_vector_z = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
light_vector_z[i] = (double*)malloc(sizeof(double)*COL);

double **light_vector; //照明ベクトルの絶対値（大きさ）
double **light_vector_xy; //照明ベクトルの z=0 のときの絶対値（θ計算用）

light_vector = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
light_vector[i] = (double*)malloc(sizeof(double)*COL);

light_vector_xy = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
light_vector_xy[i] = (double*)malloc(sizeof(double)*COL);

double **nitoubun_vector_x; //カメラ方向と照明方向の二等分線ベクトルｘ成分
double **nitoubun_vector_y; //カメラ方向と照明方向の二等分線ベクトルｙ成分
double **nitoubun_vector_z; //カメラ方向と照明方向の二等分線ベクトルｚ成分
double **nitoubun_vector; //二等分線ベクトルの絶対値（絶対値）

nitoubun_vector_x = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
nitoubun_vector_x[i] = (double*)malloc(sizeof(double)*COL);

nitoubun_vector_y = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
nitoubun_vector_y[i] = (double*)malloc(sizeof(double)*COL);

nitoubun_vector_z = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
nitoubun_vector_z[i] = (double*)malloc(sizeof(double)*COL);

nitoubun_vector = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
nitoubun_vector[i] = (double*)malloc(sizeof(double)*COL);
```

```
double LIGHT_X=0.0; //光源のワールド X 座標
double LIGHT_Y=-1089.0; //光源のワールド Y 座標
double LIGHT_Z=990.0; //光源のワールド Z 座標

double CAMERA_X=1473.0; //カメラのワールド X 座標
double CAMERA_Y=790.0; //カメラのワールド Y 座標
double CAMERA_Z=1495.0; //カメラのワールド Z 座標

double sarface_normal_x=0.0; //板の表面法線ベクトルｘ成分
double sarface_normal_y=0.0; //板の表面法線ベクトルｙ成分
double sarface_normal_z=1.0; //板の表面法線ベクトルｚ成分

double **sita; //各ピクセルのθ
double **arufa; //各ピクセルのα

sita = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
sita[i] = (double*)malloc(sizeof(double)*COL);

arufa = (double**)malloc(sizeof(double*)*ROW);
for( i=0 ; i < ROW ; i++)
arufa[i] = (double*)malloc(sizeof(double)*COL);


/*入力ファイル tmp.txt のオープン*/
if((fp_tmp=fopen("tmp.txt","r"))==NULL){
printf("File tmp.txt can't open. error! \n");
return 0;
}

printf("Now LOADING tmp.txt …… \n");

/*tmp.txt からワールド座標を X,Y の順に２つづつ値を読み込み、X[i][j],Y[i][j]
にそれぞれ入れる*/
for(i=0;i<480;i++){
for(j=0;j<640;j++){

fscanf(fp_tmp,"%lf %lf",X[i]+j,Y[i]+j);

}
}

printf("Loaded tmp.txt\n");
fclose(fp_tmp);

/*出力用ファイル out.txt のオープン*/
if((fp_out=fopen("out.txt","w+"))==NULL){
printf("File out.txt can't open. error! \n");
```

```
return 0;
}

printf("Now Calculating···· \n");

for(i=0;i<480;i++){
for(j=0;j<640;j++){

/*照明ベクトルの計算*/
light_vector_x[i][j] = LIGHT_X - X[i][j];
light_vector_y[i][j] = LIGHT_Y - Y[i][j];
light_vector_z[i][j] = LIGHT_Z ;

/*カメラベクトルの計算*/
camera_vector_x[i][j] = CAMERA_X - X[i][j];
camera_vector_y[i][j] = CAMERA_Y - Y[i][j];
camera_vector_z[i][j] = CAMERA_Z;

/*カメラ方向と照明方向の二等分線の計算*/
nitoubun_vector_x[i][j] = (light_vector_x[i][j] + camera_vector_x[i][j])/2;
nitoubun_vector_y[i][j] = (light_vector_y[i][j] + camera_vector_y[i][j])/2;
nitoubun_vector_z[i][j] = (light_vector_z[i][j] + camera_vector_z[i][j])/2;

/*θの計算*/
/*照明ベクトル light_vector[][] の絶対値計算*/
light_vector[i][j] = sqrt((light_vector_x[i][j] *
 light_vector_x[i][j]) + (light_vector_y[i][j] * light_vector_y[i][j])
 + (light_vector_z[i][j] * light_vector_z[i][j]));

/*照明ベクトルのｚ成分０の絶対値（θを求めるため）*/
light_vector_xy[i][j] = sqrt((light_vector_x[i][j] *
 light_vector_x[i][j]) + (light_vector_y[i][j] * light_vector_y[i][j]));

sita[i][j] = acos(light_vector_xy[i][j]/light_vector[i][j]);

/*αの計算*/
/*二等分線ベクトルの絶対値計算*/
nitoubun_vector[i][j] = sqrt((nitoubun_vector_x[i][j]*
nitoubun_vector_x[i][j]) + (nitoubun_vector_y[i][j]*nitoubun_vector_y[i][j])
 + (nitoubun_vector_z[i][j]*nitoubun_vector_z[i][j]));

arufa[i][j] = acos((nitoubun_vector_z[i][j])/
nitoubun_vector[i][j]);

/*出力用ファイル out.txt へ書き出す。*/
fprintf(fp_out,"%f %f\n",arufa[i][j],sita[i][j]);

}
}
```

```
printf("light_vector_x[0][0]=%lf\n",light_vector_x[0][0]);
printf("light_vector_y[0][0]=%lf\n",light_vector_y[0][0]);
printf("light_vector_z[0][0]=%lf\n",light_vector_z[0][0]);
printf("light_vector[0][0]=%lf\n",light_vector[0][0]);

printf("camera_vector_x[0][0]=%lf\n",camera_vector_x[0][0]);
printf("camera_vector_y[0][0]=%lf\n",camera_vector_y[0][0]);
printf("camera_vector_z[0][0]=%lf\n",camera_vector_z[0][0]);

printf("nitoubun_vector_x[0][0]=%lf\n",nitoubun_vector_x[0][0]);
printf("nitoubun_vector_y[0][0]=%lf\n",nitoubun_vector_y[0][0]);
printf("nitoubun_vector_z[0][0]=%lf\n",nitoubun_vector_z[0][0]);

printf("nitoubun_vector=%lf\n",nitoubun_vector[0][0]);

fclose(fp_out);
return 0;
}
```

E-3　parameta.cpp

```
////////////////////////////////////////////////////
//鏡面反射を示すいくつかのピクセルから
//
//lnI+lncos[sita],arufa^2/2の2つをとり
//
//出力ファイルparameta.txtに出力。
//
////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*BITMAP FILE HEADERの宣言*/
typedef struct tagBITMAPFILEHEADER {

        unsigned short bfTY_rpe;
unsigned long  bfSize;
unsigned short bfReserved1;
unsigned short bfReserved2;
unsigned long  bfOffBits;

}BITMAPFILEHEADER;

/*BITMAP INFO HEADERの宣言*/
typedef struct tagBITMAPINFOHEADER{

        unsigned long  biSize;
signed long    biWidth;
signed long    biHeight;
unsigned short biPlanes;
unsigned short biBitCount;
unsigned long  biCompression;
unsigned long  biSizeImage;
signed long    biX_rPelsPerMeter;
signed long    biY_rPelsPerMeter;
unsigned long  biClrUsed;
unsigned long  biClrImportant;

} BITMAPINFOHEADER;

void file_header_write(FILE*,BITMAPFILEHEADER *);//BITMAP FILE HEADER を
書き込む関数

void info_header_write(FILE*,BITMAPINFOHEADER *);//BITMAP INFO HEADER
を書き込む関数

void file_n_bY_rte_output(FILE*,unsigned long,int);//出力のための関数
```

```
main()
{
BITMAPFILEHEADER fileheader;
BITMAPINFOHEADER infoheader;

FILE *fp_ref; //鏡面反射成分を取り出すための sar_ref2.bmp へのファイルポイン
タ

FILE *fp_parameta_r; //ファイル parameta_r.txt へのファイルポインタ
FILE *fp_parameta_g; //ファイル parameta_g.txt へのファイルポインタ
FILE *fp_parameta_b; //ファイル parameta_b.txt へのファイルポインタ

FILE *fp_out; //sita arufa の入ってる

int i=0; //カウンタ用
int j=0; //カウンタ用
int k=0; //カウンタ用
int l=0;
int m=0;

/* α θ の宣言（領域確保）*/
double **sita; //各ピクセルの θ
double **arufa; //各ピクセルの α

sita = (double**)malloc(sizeof(double*)*480);
for( i=0 ; i < 480 ; i++)
sita[i] = (double*)malloc(sizeof(double)*640);

arufa = (double**)malloc(sizeof(double*)*480);
for( i=0 ; i < 480 ; i++)
arufa[i] = (double*)malloc(sizeof(double)*640);

/*計算結果 lnI+lncos[sita],(arufa^2)/2 の入れ子*/

double *X_r;
double *Y_r;
X_r=(double *)malloc(sizeof(double)*307200);
Y_r=(double *)malloc(sizeof(double)*307200);


double *X_g;
double *Y_g;
X_g=(double *)malloc(sizeof(double)*307200);
Y_g=(double *)malloc(sizeof(double)*307200);

double *X_b;
double *Y_b;
X_b=(double *)malloc(sizeof(double)*307200);
```

```
Y_b=(double *)malloc(sizeof(double)*307200);

/*取り込む鏡面反射成分の宣言（領域確保）*/
unsigned char *ref_r;
unsigned char *ref_g;
unsigned char *ref_b;
unsigned char *ref;

ref_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
ref_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
ref_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);
ref=(unsigned char*)malloc(sizeof(unsigned char)*307200*3);

/*鏡面反射の取り込み（２次元配列）*/
unsigned char **ref_r2;
unsigned char **ref_g2;
unsigned char **ref_b2;

ref_r2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
ref_r2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

ref_g2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
ref_g2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

ref_b2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
ref_b2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

/*１枚目の写真（鏡面反射成分画像）の読み込み*/

if((fp_ref=fopen("sar_ref2.bmp","rb"))==NULL){
printf("File open error 1 \n");
return 0;
}

/*１枚目の写真のヘッダーの読み込み*/
if(fread(&fileheader.bfTY_rpe,2,1,fp_ref)!=1){
printf("read error1");
}

printf("fileheader.bfTY_rpe=%u\n",fileheader.bfTY_rpe);

if(fread(&fileheader.bfSize,4,1,fp_ref)!=1){
printf("read error2");
}

printf("fileheader.bfSize=%u\n",fileheader.bfSize);
```

```
if(fread(&fileheader.bfReserved1,2,1,fp_ref)!=1){
printf("read error3");
}

printf("fileheader.bfReserved1=%u\n",fileheader.bfReserved1);

if(fread(&fileheader.bfReserved2,2,1,fp_ref)!=1){
printf("read error4");
}

printf("fileheader.bfReserved2=%u\n",fileheader.bfReserved2);

if(fread(&fileheader.bfOffBits,4,1,fp_ref)!=1){
printf("read error5");
}

printf("fileheader.bfOffBits=%u\n",fileheader.bfOffBits);

if(fread(&infoheader.biSize,4,1,fp_ref)!=1){
  printf("read error6");
}

printf("infoheader.biSize=%u\n",infoheader.biSize);

if(fread(&infoheader.biWidth,4,1,fp_ref)!=1){
printf("read error7");
}

printf("infoheader.biWidth=%u\n",infoheader.biWidth);

if(fread(&infoheader.biHeight,4,1,fp_ref)!=1){
printf("read error8");
}

printf("infoheader.biHeight=%u\n",infoheader.biHeight);

if(fread(&infoheader.biPlanes,2,1,fp_ref)!=1){
printf("read error9");
}

printf("infoheader.biPlanes=%u\n",infoheader.biPlanes);

if(fread(&infoheader.biBitCount,2,1,fp_ref)!=1){
printf("read error10");
}

printf("infoheader.biBitCount=%d\n",infoheader.biBitCount);
```

```
if(fread(&infoheader.biCompression,4,1,fp_ref)!=1){
printf("read error11");
}

printf("infoheader.biCompression=%u\n",infoheader.biCompression);

if(fread(&infoheader.biSizeImage,4,1,fp_ref)!=1){
printf("read error12");
}

printf("infoheader.biSizeImage=%u\n",infoheader.biSizeImage);

if(fread(&infoheader.biX_rPelsPerMeter,4,1,fp_ref)!=1){
printf("read error13");
}

printf("infoheader.biX_rPelsPerMeter=%u\n",infoheader.biX_rPelsPerMeter);

if(fread(&infoheader.biY_rPelsPerMeter,4,1,fp_ref)!=1){
printf("read error14");
}

printf("infoheader.biY_rPelsPerMeter=%u\n",infoheader.biY_rPelsPerMeter);


if(fread(&infoheader.biClrUsed,4,1,fp_ref)!=1){
printf("read error15");
}

printf("infoheader.biClrUsed=%u\n",infoheader.biClrUsed);

if(fread(&infoheader.biClrImportant,4,1,fp_ref)!=1){
printf("read error16");
}

printf("infoheader.biClrImportant=%u\n",infoheader.biClrImportant);


/*１枚目の写真の中身の読み込み*/
i=0; //カウンタの初期化
while((fread(&ref[i],sizeof(unsigned char),1,fp_ref))==1){
i++;
}

i=0;    //カウンタの初期化
for(k=0;k<307200;k++){

ref_b[k]=ref[i];
i++;
```

```
ref_g[k]=ref[i];
i++;
ref_r[k]=ref[i];
i++;

}

k=0;//カウンタの初期化

/*取り込んだ鏡面反射成分を 2 次元配列に入れ替える*/
for(i=0;i<480;i++){
for(j=0;j<640;j++){

ref_b2[i][j]=ref_b[k];
ref_g2[i][j]=ref_g[k];
ref_r2[i][j]=ref_r[k];

k++;

}
}

printf("fajopdijpkjgpfjkp\n");



/* α と θ を out.txt から読み込む*/

if((fp_out=fopen("out1.txt","r"))==NULL){
printf("File out.txt can't open. error! \n");
return 0;
}

for(i=0;i<480;i++){
for(j=0;j<640;j++){

fscanf(fp_out,"%lf %lf",arufa[479-i]+j,sita[479-i]+j);

}
}

/*出力用ファイル parameta2.txt のオープン*/

if((fp_parameta_r=fopen("parameta_r6.txt","w+"))==NULL){
printf("File parameta6_r.txt can't open. error! \n");
return 0;
}

/*出力用ファイル parameta_g2.txt のオープン*/
```

```
if((fp_parameta_g=fopen("parameta_g6.txt","w+"))==NULL){
printf("File parameta_g.txt can't open. error! \n");
return 0;
}

/*出力用ファイル parameta_b.txt のオープン*/

if((fp_parameta_b=fopen("parameta_b6.txt","w+"))==NULL){
printf("File parameta_b.txt can't open. error! \n");
return 0;
}



//カウンタの初期化
k=0;
l=0;
m=0;

/*lnI+ln[cos(sita)],(arufa^2)/2 の計算*/

for(i=0;i<480;i++){
for(j=0;j<640;j++){

if(ref_r2[i][j]<240 && ref_r2[i][j]>30/* && arufa[i][j]>0.01*/)

{
Y_r[k]=log(ref_r2[i][j])+log(cos(sita[i][j]));
X_r[k]=(arufa[i][j]*arufa[i][j])/2;
k++;
}


if(ref_g2[i][j]<240 && ref_g2[i][j]>30/*&& arufa[i][j]>0.01*/)

{
Y_g[l]=log(ref_g2[i][j])+log(cos(sita[i][j]));
X_g[l]=(arufa[i][j]*arufa[i][j])/2;
l++;
}

if(ref_b2[i][j]<240 && ref_b2[i][j]>30/*&& arufa[i][j]>0.01*/)

{
Y_b[m]=log(ref_b2[i][j])+log(cos(sita[i][j]));
X_b[m]=(arufa[i][j]*arufa[i][j])/2;
m++;
}
```

```
}
}



i=0;//カウンタの初期化
/*出力ファイル parameta_r.txt への書き込み*/
for(i=0;i<k;i++){

if(X_r[i]>0.05 && X_r[i]<0.2 && Y_r[i]<7.0 && Y_r[i]>1.0){
fprintf(fp_parameta_r,"%lf %lf\n",X_r[i],Y_r[i]);
}
}

i=0;//カウンタの初期化
/*出力ファイル parameta_g.txt への書き込み*/
for(i=0;i<l;i++){
if(X_g[i]>0.05 && X_g[i]<0.2 && Y_g[i]<7.0 && Y_g[i]>1.0){
fprintf(fp_parameta_g,"%lf %lf\n",X_g[i],Y_g[i]);
}
}

i=0;//カウンタの初期化 r
/*出力ファイル parameta_b.txt への書き込み*/
for(i=0;i<m;i++){
if(X_b[i]>0.05 && X_b[i]<0.2 && Y_b[i]<7.0 && Y_b[i]>1.0){
fprintf(fp_parameta_b,"%lf %lf\n",X_b[i],Y_b[i]);
}
}

fclose(fp_ref);
fclose(fp_out);
fclose(fp_parameta_r);
fclose(fp_parameta_g);
fclose(fp_parameta_b);

return 0;
}
```

E-4　seisei.cpp

```
////////////////////////////////////
//粗さ定数、反射定数を入れて
//鏡面反射成分と拡散反射成分＋鏡面反射成分の
//画像を出力
////////////////////////////////////
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

/*BITMAP FILE HEADERの宣言*/
typedef struct tagBITMAPFILEHEADER {

    unsigned short bfType;

    unsigned long  bfSize;

    unsigned short bfReserved1;

    unsigned short bfReserved2;

    unsigned long  bfOffBits;

}BITMAPFILEHEADER;

/*BITMAP INFO HEADERの宣言*/
typedef struct tagBITMAPINFOHEADER{

    unsigned long  biSize;

    signed long    biWidth;

    signed long    biHeight;

    unsigned short biPlanes;

    unsigned short biBitCount;

    unsigned long  biCompression;

    unsigned long  biSizeImage;

    signed long    biXPelsPerMeter;

    signed long    biYPelsPerMeter;

    unsigned long  biClrUsed;
```

```
    unsigned long  biClrImportant;

} BITMAPINFOHEADER;

void file_header_write(FILE*,BITMAPFILEHEADER *);//BITMAP FILE HEADER を
書き込む関数

void info_header_write(FILE*,BITMAPINFOHEADER *);//BITMAP INFO HEADER
を書き込む関数

void file_n_byte_output(FILE*,unsigned long,int);//出力のための関数

main()
{

BITMAPFILEHEADER fileheader; //fileheader 構造体へのポインタ
BITMAPINFOHEADER infoheader; //infoheader 構造体へのポインタ

FILE *fp_s_min;          //min（拡散反射成分）を読み込む際のファイルポインタ
FILE *fp_s_min_test; //min（拡散反射成分）を書き出す際のファイルポインタ
FILE *fp_s_ref;      //鏡面反射成分を書き出す際のファイルポインタ
FILE *fp_seisei;     //生成した画像を書き出す際のファイルポインタ
FILE *fp_out; //各ピクセルのθ、αが入っているファイル out.txt へのファイルポ
インタ
FILE *fp_out2; //テスト用

int i=0; //カウンタ用
int j=0; //カウンタ用
int k=0; //カウンタ用
int ss=0;

double arasa=0.11262;
double teisuu_r=exp(11.247);
double teisuu_g=exp(11.058);
double teisuu_b=exp(10.932);

unsigned char *s_min; //拡散反射成分を取り込む（極小）

unsigned char *s_min_r;       //
unsigned char *s_min_g;       //拡散反射成分
unsigned char *s_min_b;       //

unsigned char *s_ref_r;       //
unsigned char *s_ref_g;       //生成した鏡面反射成分
unsigned char *s_ref_b;       //

/* unsigned char ref_r[480][640];//鏡面反射成分生成のためにつくった
unsigned char ref_g[480][640];//便宜的な２次元配列
```

```
unsigned char ref_b[480][640];//
*/
unsigned char **ref_r;//鏡面反射成分生成のためにつくった
unsigned char **ref_g;//鏡面反射成分生成のためにつくった
unsigned char **ref_b;//鏡面反射成分生成のためにつくった

ref_r = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
ref_r[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

ref_g = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
ref_g[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

ref_b = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
ref_b[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

unsigned char *seisei_r;      //
unsigned char *seisei_g;      //拡散反射＋生成した鏡面反射成分
unsigned char *seisei_b;      //

s_min_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
s_min_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
s_min_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);

s_ref_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
s_ref_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
s_ref_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);

seisei_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
seisei_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
seisei_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);

s_min = (unsigned char*)malloc(sizeof(unsigned char)*307200*3);

double **sita; //各ピクセルのθ
double **arufa; //各ピクセルのα

sita = (double**)malloc(sizeof(double*)*480);
for( i=0 ; i < 480 ; i++)
sita[i] = (double*)malloc(sizeof(double)*640);

arufa = (double**)malloc(sizeof(double*)*480);
for( i=0 ; i < 480 ; i++)
arufa[i] = (double*)malloc(sizeof(double)*640);

/*１枚目の写真の読み込み*/
if((fp_s_min=fopen("result1.bmp","rb"))==NULL){
```

```
printf("File open error 1 \n");
return 0;
}

/*１枚目の写真のヘッダーの読み込み*/
if(fread(&fileheader.bfType,2,1,fp_s_min)!=1){
printf("read error1");
}

printf("fileheader.bfType=%u\n",fileheader.bfType);

if(fread(&fileheader.bfSize,4,1,fp_s_min)!=1){
printf("read error2");
}

printf("fileheader.bfSize=%u\n",fileheader.bfSize);

if(fread(&fileheader.bfReserved1,2,1,fp_s_min)!=1){
printf("read error3");
}

printf("fileheader.bfReserved1=%u\n",fileheader.bfReserved1);

if(fread(&fileheader.bfReserved2,2,1,fp_s_min)!=1){
printf("read error4");
}

printf("fileheader.bfReserved2=%u\n",fileheader.bfReserved2);

if(fread(&fileheader.bfOffBits,4,1,fp_s_min)!=1){
printf("read error5");
}

printf("fileheader.bfOffBits=%u\n",fileheader.bfOffBits);

if(fread(&infoheader.biSize,4,1,fp_s_min)!=1){
  printf("read error6");
}

printf("infoheader.biSize=%u\n",infoheader.biSize);

if(fread(&infoheader.biWidth,4,1,fp_s_min)!=1){
printf("read error7");
}

printf("infoheader.biWidth=%u\n",infoheader.biWidth);

if(fread(&infoheader.biHeight,4,1,fp_s_min)!=1){
printf("read error8");
```

```
}

printf("infoheader.biHeight=%u\n",infoheader.biHeight);

if(fread(&infoheader.biPlanes,2,1,fp_s_min)!=1){
printf("read error9");
}

printf("infoheader.biPlanes=%u\n",infoheader.biPlanes);

if(fread(&infoheader.biBitCount,2,1,fp_s_min)!=1){
printf("read error10");
}

printf("infoheader.biBitCount=%d\n",infoheader.biBitCount);

if(fread(&infoheader.biCompression,4,1,fp_s_min)!=1){
printf("read error11");
}

printf("infoheader.biCompression=%u\n",infoheader.biCompression);

if(fread(&infoheader.biSizeImage,4,1,fp_s_min)!=1){
printf("read error12");
}

printf("infoheader.biSizeImage=%u\n",infoheader.biSizeImage);

if(fread(&infoheader.biXPelsPerMeter,4,1,fp_s_min)!=1){
printf("read error13");
}

printf("infoheader.biXPelsPerMeter=%u\n",infoheader.biXPelsPerMeter);

if(fread(&infoheader.biYPelsPerMeter,4,1,fp_s_min)!=1){
printf("read error14");
}

printf("infoheader.biYPelsPerMeter=%u\n",infoheader.biYPelsPerMeter);


if(fread(&infoheader.biClrUsed,4,1,fp_s_min)!=1){
printf("read error15");
}

printf("infoheader.biClrUsed=%u\n",infoheader.biClrUsed);

if(fread(&infoheader.biClrImportant,4,1,fp_s_min)!=1){
printf("read error16");
```

```
}

printf("infoheader.biClrImportant=%u\n",infoheader.biClrImportant);


/*１枚目の写真の中身の読み込み*/
i=0; //カウンタの初期化
while((fread(&s_min[i],sizeof(unsigned char),1,fp_s_min))==1){
i++;
}

i=0;    //カウンタの初期化
for(k=0;k<307200;k++){

s_min_b[k]=s_min[i];
i++;
s_min_g[k]=s_min[i];
i++;
s_min_r[k]=s_min[i];
i++;

}


/*出力用ファイル  min_test  の OPEN*/
if((fp_s_min_test = fopen("s_min_test.bmp", "wb")) == NULL){

printf("FILE s_min_test.bmp cant't open.\n");

return 0;

    }

/*出力用ファイル s_min_test のヘッダーへの書き込み*/
file_header_write(fp_s_min_test,&fileheader);
info_header_write(fp_s_min_test,&infoheader);

/*出力用ファイル min_test の中身へ書き込み*/
for(i=0;i<307200;i++)
{
if((fwrite(&s_min_b[i],sizeof(unsigned char),1,fp_s_min_test))!=1){
printf("s_min_test.bmp write error\n");
}

if((fwrite(&s_min_g[i],sizeof(unsigned char),1,fp_s_min_test))!=1){
printf("s_min_test.bmp write error\n");
}

if((fwrite(&s_min_r[i],sizeof(unsigned char),1,fp_s_min_test))!=1){
```

```
printf("s_min_test.bmp write error\n");
}

}


/*αとθをout.txtから読み込む*/

if((fp_out=fopen("out1.txt","r"))==NULL){
printf("File out.txt can't open. error! \n");
return 0;
}

for(i=0;i<480;i++){
for(j=0;j<640;j++){

fscanf(fp_out,"%lf %lf",arufa[479-i]+j,sita[479-i]+j);

}
}


//鏡面反射成分の生成

for(i=0;i<480;i++){
for(j=0;j<640;j++){

ref_r[i][j]=(teisuu_r/cos(sita[i][j]))*exp((-arufa[i][j]*arufa[i][j])/(2*arasa*arasa));
ref_g[i][j]=(teisuu_g/cos(sita[i][j]))*exp((-arufa[i][j]*arufa[i][j])/(2*arasa*arasa));
ref_b[i][j]=(teisuu_b/cos(sita[i][j]))*exp((-arufa[i][j]*arufa[i][j])/(2*arasa*arasa));
}
}



/*ref_r ref_g ref_b  に値が入っているかどうか確認のためout2.txtに書き出し
た*/
if((fp_out2=fopen("out2.txt","w+"))==NULL){
printf("File out2.txt can't open. error! \n");
return 0;
}

for( j=0 ; j<100 ;j++){
fprintf(fp_out2,"%u %u %u\n",ref_r[0][j],ref_g[0][j],ref_b[0][j]);
}


k=0;//カウンタの初期化
for(i=0;i<480;i++){
```

```
for(j=0;j<640;j++){

if(ref_r[i][j]>255){s_ref_r[k]=255;}

if(ref_g[i][j]>255){s_ref_g[k]=255;}

if(ref_b[i][j]>255){s_ref_b[k]=255;}

else{

s_ref_r[k]=ref_r[i][j];
s_ref_g[k]=ref_g[i][j];
s_ref_b[k]=ref_b[i][j];

}

k++;

}
}

/*出力用ファイル　s_ref（生成した鏡面反射の成分）　の OPEN*/
if((fp_s_ref = fopen("s_ref_b.bmp", "wb")) == NULL){

printf("FILE ref_test.bmp cant't open.\n");

return 0;

    }


for(i = 0; i < 480; i++){

for(j = 0; j < 640; j++){

if((fwrite(&ref[i][j],sizeof(unsigned char),1,fp_s_ref))!=1){
printf("s_ref.bmp write error\n");
}
}

}


/*出力用ファイル s_ref（生成した鏡面反射の成分） の中身への書き込み*/
for(i=0;i<307200;i++)
{
if((fwrite(&s_ref_b[i],sizeof(unsigned char),1,fp_s_ref))!=1){
printf("s_ref_b.bmp write error\n");
}
```

```
if((fwrite(&s_ref_g[i],sizeof(unsigned char),1,fp_s_ref))!=1){
printf("s_ref_g.bmp write error\n");
}

if((fwrite(&s_ref_r[i],sizeof(unsigned char),1,fp_s_ref))!=1){
printf("s_ref_r.bmp write error\n");
}

}


        //合成した画像 seisei  の生成

/*rgb それぞれ   s_ref[]+s_min[] を seisei[] に入れる*/
for(i = 0; i < 307200; i++){

seisei_r[i]=s_ref_r[i]+s_min_r[i];
seisei_g[i]=s_ref_g[i]+s_min_g[i];
seisei_b[i]=s_ref_b[i]+s_min_b[i];

}

/*出力用ファイル   seisei  の OPEN*/
if((fp_seisei = fopen("seisei_r.bmp", "wb")) == NULL){

printf("FILE seisei.bmp cant't open.\n");

return 0;

    }

/*出力用ファイル seisei のヘッダーへの書き込み*/
file_header_write(fp_seisei,&fileheader);
info_header_write(fp_seisei,&infoheader);

for(i=0;i<307200;i++)
{
if((fwrite(&seisei_b[i],sizeof(unsigned char),1,fp_seisei))!=1){
printf("seisei_b.bmp write error\n");
}

if((fwrite(&seisei_g[i],sizeof(unsigned char),1,fp_seisei))!=1){
printf("seisei_g.bmp write error\n");
}

if((fwrite(&seisei_r[i],sizeof(unsigned char),1,fp_seisei))!=1){
printf("seisei_r.bmp write error\n");
}
```

```
}

fclose (fp_s_min);
fclose (fp_s_min_test);
fclose (fp_s_ref);
fclose (fp_seisei);
fclose (fp_out);
fclose (fp_out2);

return 0;
}


void file_header_write(FILE *fp,BITMAPFILEHEADER *head)

{

    file_n_byte_output(fp,head->bfType,2);

    file_n_byte_output(fp,head->bfSize,4);

    file_n_byte_output(fp,head->bfReserved1,2);

    file_n_byte_output(fp,head->bfReserved2,2);

    file_n_byte_output(fp,head->bfOffBits,4);

}

void info_header_write(FILE *fp,BITMAPINFOHEADER *head)

{

    file_n_byte_output(fp,head->biSize,4);

    file_n_byte_output(fp,head->biWidth,4);

    file_n_byte_output(fp,head->biHeight,4);

    file_n_byte_output(fp,head->biPlanes,2);

    file_n_byte_output(fp,head->biBitCount,2);

    file_n_byte_output(fp,head->biCompression,4);

    file_n_byte_output(fp,head->biSizeImage,4);

    file_n_byte_output(fp,head->biXPelsPerMeter,4);
```

```
    file_n_byte_output(fp,head->biYPelsPerMeter,4);

    file_n_byte_output(fp,head->biClrUsed,4);

    file_n_byte_output(fp,head->biClrImportant,4);

}

void file_n_byte_output(FILE *fp,unsigned long number, int mojisuu)

{

    int i;

for (i = 0; i < mojisuu; i++){

fputc((int)(number % 256), fp);

number = number / 256;

    }
}
```

E-5　hyouka.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
////////////////////////////
//２つの画像のある一列
//における画素値を出力
////////////////////////////
/*BITMAP FILE HEADERの宣言*/
typedef struct tagBITMAPFILEHEADER {

        unsigned short bfType;
unsigned long  bfSize;
unsigned short bfReserved1;
unsigned short bfReserved2;
unsigned long  bfOffBits;

}BITMAPFILEHEADER;

/*BITMAP INFO HEADERの宣言*/
typedef struct tagBITMAPINFOHEADER{

        unsigned long  biSize;
signed long    biWidth;
signed long    biHeight;
unsigned short biPlanes;
unsigned short biBitCount;
unsigned long  biCompression;
unsigned long  biSizeImage;
signed long    biXPelsPerMeter;
signed long    biYPelsPerMeter;
unsigned long  biClrUsed;
unsigned long  biClrImportant;

} BITMAPINFOHEADER;

void file_header_write(FILE*,BITMAPFILEHEADER *);//BITMAP FILE HEADER を
書き込む関数

void info_header_write(FILE*,BITMAPINFOHEADER *);//BITMAP INFO HEADER
を書き込む関数

void file_n_byte_output(FILE*,unsigned long,int);//出力のための関数

main()
{
BITMAPFILEHEADER fileheader;
BITMAPINFOHEADER infoheader;
```

```
FILE *fp_moto; //元の画像を取り出すための result0.bmp へのファイルポインタ

FILE *fp_seisei; //生成した画像へのファイルポインタ
FILE *fp_hyouka_r; //各ピクセルの角度（y、画素）が入っているファイル hyouka.txt
へのファイルポインタ
FILE *fp_hyouka_g;
FILE *fp_hyouka_b;

int i=0; //カウンタ用
int j=0; //カウンタ用
int k=0; //カウンタ用
int m=0; //カウンタ用

/*取り込む元の画像の宣言（領域確保）*/
unsigned char *moto_r;
unsigned char *moto_g;
unsigned char *moto_b;
unsigned char *moto;

moto_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
moto_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
moto_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);
moto=(unsigned char*)malloc(sizeof(unsigned char)*307200*3);

/*鏡面反射の取り込み（２次元配列）*/
unsigned char **moto_r2;
unsigned char **moto_g2;
unsigned char **moto_b2;

moto_r2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
moto_r2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

moto_g2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
moto_g2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

moto_b2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
moto_b2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

/*取り込む生成画像の宣言（領域確保）*/
unsigned char *seisei_r;
unsigned char *seisei_g;
unsigned char *seisei_b;
unsigned char *seisei;

seisei_r=(unsigned char*)malloc(sizeof(unsigned char)*307200);
```

```
seisei_g=(unsigned char*)malloc(sizeof(unsigned char)*307200);
seisei_b=(unsigned char*)malloc(sizeof(unsigned char)*307200);
seisei=(unsigned char*)malloc(sizeof(unsigned char)*307200*3);

/*生成画像の取り込み（２次元配列）*/
unsigned char **seisei_r2;
unsigned char **seisei_g2;
unsigned char **seisei_b2;

seisei_r2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
seisei_r2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

seisei_g2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
seisei_g2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

seisei_b2 = (unsigned char**)malloc(sizeof(unsigned char*)*480);
for( i=0 ; i < 480 ; i++)
seisei_b2[i] = (unsigned char*)malloc(sizeof(unsigned char)*640);

/*１枚目の写真（もとの画像）の読み込み*/

if((fp_moto=fopen("sar_ref2.bmp","rb"))==NULL){
printf("File open error 1 \n");
return 0;
}

/*１枚目の写真のヘッダーの読み込み*/
if(fread(&fileheader.bfType,2,1,fp_moto)!=1){
printf("read error1");
}

printf("fileheader.bfType=%u\n",fileheader.bfType);

if(fread(&fileheader.bfSize,4,1,fp_moto)!=1){
printf("read error2");
}

printf("fileheader.bfSize=%u\n",fileheader.bfSize);

if(fread(&fileheader.bfReserved1,2,1,fp_moto)!=1){
printf("read error3");
}

printf("fileheader.bfReserved1=%u\n",fileheader.bfReserved1);

if(fread(&fileheader.bfReserved2,2,1,fp_moto)!=1){
printf("read error4");
```

```
}

printf("fileheader.bfReserved2=%u\n",fileheader.bfReserved2);

if(fread(&fileheader.bfOffBits,4,1,fp_moto)!=1){
printf("read error5");
}

printf("fileheader.bfOffBits=%u\n",fileheader.bfOffBits);

if(fread(&infoheader.biSize,4,1,fp_moto)!=1){
  printf("read error6");
}

printf("infoheader.biSize=%u\n",infoheader.biSize);

if(fread(&infoheader.biWidth,4,1,fp_moto)!=1){
printf("read error7");
}

printf("infoheader.biWidth=%u\n",infoheader.biWidth);

if(fread(&infoheader.biHeight,4,1,fp_moto)!=1){
printf("read error8");
}

printf("infoheader.biHeight=%u\n",infoheader.biHeight);

if(fread(&infoheader.biPlanes,2,1,fp_moto)!=1){
printf("read error9");
}

printf("infoheader.biPlanes=%u\n",infoheader.biPlanes);

if(fread(&infoheader.biBitCount,2,1,fp_moto)!=1){
printf("read error10");
}

printf("infoheader.biBitCount=%d\n",infoheader.biBitCount);

if(fread(&infoheader.biCompression,4,1,fp_moto)!=1){
printf("read error11");
}

printf("infoheader.biCompression=%u\n",infoheader.biCompression);

if(fread(&infoheader.biSizeImage,4,1,fp_moto)!=1){
printf("read error12");
}
```

```
printf("infoheader.biSizeImage=%u\n",infoheader.biSizeImage);

if(fread(&infoheader.biXPelsPerMeter,4,1,fp_moto)!=1){
printf("read error13");
}

printf("infoheader.biXPelsPerMeter=%u\n",infoheader.biXPelsPerMeter);

if(fread(&infoheader.biYPelsPerMeter,4,1,fp_moto)!=1){
printf("read error14");
}

printf("infoheader.biYPelsPerMeter=%u\n",infoheader.biYPelsPerMeter);


if(fread(&infoheader.biClrUsed,4,1,fp_moto)!=1){
printf("read error15");
}

printf("infoheader.biClrUsed=%u\n",infoheader.biClrUsed);

if(fread(&infoheader.biClrImportant,4,1,fp_moto)!=1){
printf("read error16");
}

printf("infoheader.biClrImportant=%u\n",infoheader.biClrImportant);


/*１枚目の写真の中身の読み込み*/
i=0; //カウンタの初期化
while((fread(&moto[i],sizeof(unsigned char),1,fp_moto))==1){
i++;
}

i=0;    //カウンタの初期化
for(k=0;k<307200;k++){

moto_b[k]=moto[i];
i++;
moto_g[k]=moto[i];
i++;
moto_r[k]=moto[i];
i++;

}

k=0;//カウンタの初期化
```

```
/*取り込んだ元の画像を 2 次元配列に入れ替える*/
for(i=0;i<480;i++){
for(j=0;j<640;j++){

moto_b2[i][j]=moto_b[k];
moto_g2[i][j]=moto_g[k];
moto_r2[i][j]=moto_r[k];

k++;

}
}

/*２枚目の写真（生成画像）の読み込み*/

if((fp_seisei=fopen("seisei_r.bmp","rb"))==NULL){
printf("File open error 2 \n");
return 0;
}

/*２枚目の写真のヘッダーの読み込み*/
if(fread(&fileheader.bfType,2,1,fp_seisei)!=1){
printf("read error1");
}

printf("fileheader.bfType=%u\n",fileheader.bfType);

if(fread(&fileheader.bfSize,4,1,fp_seisei)!=1){
printf("read error2");
}

printf("fileheader.bfSize=%u\n",fileheader.bfSize);

if(fread(&fileheader.bfReserved1,2,1,fp_seisei)!=1){
printf("read error3");
}

printf("fileheader.bfReserved1=%u\n",fileheader.bfReserved1);

if(fread(&fileheader.bfReserved2,2,1,fp_seisei)!=1){
printf("read error4");
}

printf("fileheader.bfReserved2=%u\n",fileheader.bfReserved2);

if(fread(&fileheader.bfOffBits,4,1,fp_seisei)!=1){
printf("read error5");
}
```

```
printf("fileheader.bfOffBits=%u\n",fileheader.bfOffBits);

if(fread(&infoheader.biSize,4,1,fp_seisei)!=1){
  printf("read error6");
}

printf("infoheader.biSize=%u\n",infoheader.biSize);

if(fread(&infoheader.biWidth,4,1,fp_seisei)!=1){
printf("read error7");
}

printf("infoheader.biWidth=%u\n",infoheader.biWidth);

if(fread(&infoheader.biHeight,4,1,fp_seisei)!=1){
printf("read error8");
}

printf("infoheader.biHeight=%u\n",infoheader.biHeight);

if(fread(&infoheader.biPlanes,2,1,fp_seisei)!=1){
printf("read error9");
}

printf("infoheader.biPlanes=%u\n",infoheader.biPlanes);

if(fread(&infoheader.biBitCount,2,1,fp_seisei)!=1){
printf("read error10");
}

printf("infoheader.biBitCount=%d\n",infoheader.biBitCount);

if(fread(&infoheader.biCompression,4,1,fp_seisei)!=1){
printf("read error11");
}

printf("infoheader.biCompression=%u\n",infoheader.biCompression);

if(fread(&infoheader.biSizeImage,4,1,fp_seisei)!=1){
printf("read error12");
}

printf("infoheader.biSizeImage=%u\n",infoheader.biSizeImage);

if(fread(&infoheader.biXPelsPerMeter,4,1,fp_seisei)!=1){
printf("read error13");
}

printf("infoheader.biXPelsPerMeter=%u\n",infoheader.biXPelsPerMeter);
```

```
if(fread(&infoheader.biYPelsPerMeter,4,1,fp_seisei)!=1){
printf("read error14");
}

printf("infoheader.biYPelsPerMeter=%u\n",infoheader.biYPelsPerMeter);


if(fread(&infoheader.biClrUsed,4,1,fp_seisei)!=1){
printf("read error15");
}

printf("infoheader.biClrUsed=%u\n",infoheader.biClrUsed);

if(fread(&infoheader.biClrImportant,4,1,fp_seisei)!=1){
printf("read error16");
}

printf("infoheader.biClrImportant=%u\n",infoheader.biClrImportant);




/*２枚目の写真の中身の読み込み*/
i=0; //カウンタの初期化
while((fread(&seisei[i],sizeof(unsigned char),1,fp_seisei))==1){
i++;
}


i=0;    //カウンタの初期化
for(k=0;k<307200;k++){

seisei_b[k]=seisei[i];
i++;
seisei_g[k]=seisei[i];
i++;
seisei_r[k]=seisei[i];
i++;

}

k=0;//カウンタの初期化

/*取り込んだ生成画像を 2 次元配列に入れ替える*/
for(i=0;i<480;i++){
for(j=0;j<640;j++){

seisei_b2[i][j]=seisei_b[k];
```

```
seisei_g2[i][j]=seisei_g[k];
seisei_r2[i][j]=seisei_r[k];

k++;

}
}

/*出力用ファイル hyouka_teisuu.txt のオープン*/

if((fp_hyouka_r=fopen("hyouka_r2.txt","w+"))==NULL){
printf("File hyouka_r2.txt can't open. error! \n");
return 0;
}

/*出力用ファイル hyouka_.txt のオープン*/

if((fp_hyouka_g=fopen("hyouka_g2.txt","w+"))==NULL){
printf("File hyouka_g2.txt can't open. error! \n");
return 0;
}

/*出力用ファイル hyouka_teisuu.txt のオープン*/

if((fp_hyouka_b=fopen("hyouka_b2.txt","w+"))==NULL){
printf("File hyouka_b2.txt can't open. error! \n");
return 0;
}


for(j=0;j<640;j++){
fprintf(fp_hyouka_r,"%u %u\n",moto_r2[40][j],seisei_r2[40][j]);
j=j+3;
}

for(j=0;j<640;j++){
fprintf(fp_hyouka_g,"%u %u\n",moto_g2[40][j],seisei_g2[40][j]);
j=j+3;
}

for(j=0;j<640;j++){
fprintf(fp_hyouka_b,"%u %u\n",moto_b2[40][j],seisei_b2[40][j]);
j=j+3;
}


fclose(fp_hyouka_r);
fclose(fp_hyouka_g);
fclose(fp_hyouka_b);
```

```
fclose(fp_moto);
fclose(fp_seisei);

return 0;



}
```