Optical_simulator author AngularSpectrumMTD

最終更新日 2020/08/22

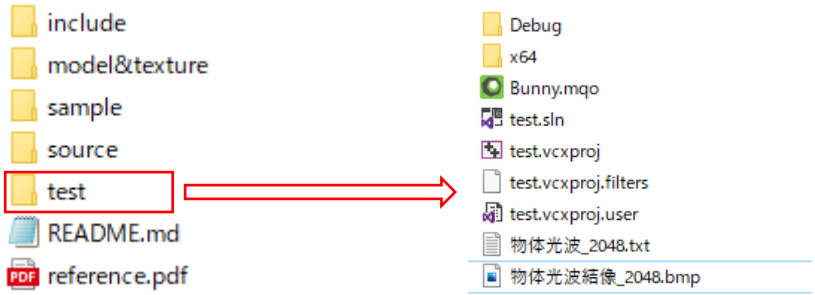
前書き

このプログラムは，角スペクトル法(Angular Spectrum Method)と呼ばれるヘルムホルツ方程式の解析解を計算する手法により，自由空間内における回折伝搬計算を行うことを目的としたものです.
レンズによる光の収斂・結像に始まり，メルセンヌツイスタより生成される乱数を用いた光の散乱を行えます.
また，三次元映像技術であるホログラフィによる成果物・ホログラムを計算機によって計算するコンピュータホログラフィにおいて，物体からの光波(物体光波)を計算する手法を取り入れています．この手法はポリゴン法と呼ばれるものであり，3DCG モデルのポリゴンを有限広さの面光源として捉えることで物体光波を計算します．二次元分布として光波分布を扱うので，テクスチャの反映も容易であり，フラット・スムースシェーディングを本プログラムで行いその結果を光波として計算することが可能です.
実際，ホログラムの作製には干渉縞が必要ですが，干渉縞をガラス乾板などに描画する機器は大変高価であり，生成しても描画する機会を得ることが困難であろうと考えたので現在は意図してつけていません.
そのため，光波だけが現在計算できる状況です.

外部ヘッダ

stb_image.h: stb ライブラリ(http://nothings.org/stb/)
stb_image_write.h: stb ライブラリ(http://nothings.org/stb/)
include/sample/source フォルダを sln と同一の階層にコピーし，更に上記ヘッダオンリライブラリを include フォルダ直下に配置してください.

動作時構成など



以上の様にしてください.
モデルファイルはサンプルコードの初期指定パスがプロジェクトのある階層なので，そのまま動作させる場合は上図右の様にしてください.
また，SDL チェックは外してください.

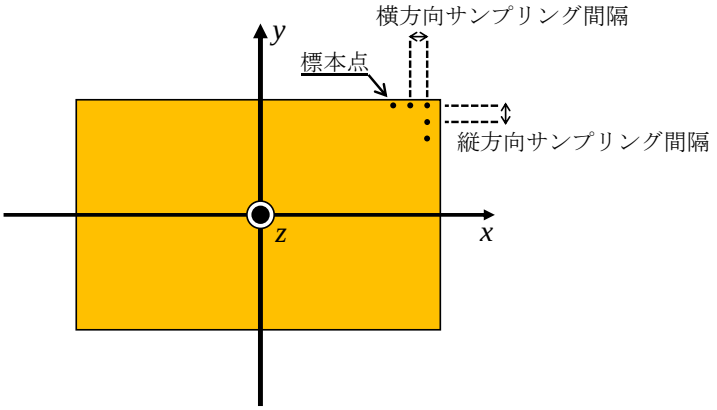
クラス構成

このプログラムは以下 4 つのクラスで構成されています.

- WaveFront: 基本的な光波計算の機能を集約したクラス
- ImagingWaveFront: WaveFront クラスの子クラス 光波の結像再生を行う
- Model: mqo モデルから物体光波を計算するクラス
- Image: BMP 画像の読み込みを行うためのクラス

WaveFront

WaveFront クラスは単一の波長をもった光波を二次元信号としてサンプリングしたデータを扱うクラスです．デフォルトコンストラクタは後に消します．
このクラスから生成されるオブジェクトはそれぞれ固有のローカル座標を持ちます．
また軸とサンプリング間隔などは以下のようになっています．



また画像出力における形式指定及び，補間法の指定・csv ファイル出力時の軸指定に関して以下の enum を持ちます．

出力 Out: REAL,IMAGE,AMPLITUDE,INTENSITY,PHASE
→実部形式/虚部形式/振幅形式/強度形式/位相形式
補間法 Interp: NEAREST_NEIGHBOR,BILINEAR,BICUBIC
→最近傍法/バイリニア法/バイキュービック法
csvファイル出力時の軸 Axis: X_AXIS,Y_AXIS
→x 軸/y 軸

[コンストラクタ]
WaveFront(int nx, int ny, double px, double py, double lambda)
引数: 横ピクセル数・縦ピクセル数・横サンプリング間隔・縦サンプリング間隔・波長
戻り値: WaveFront オブジェクト
WaveFront(const WaveFront& a)
引数: WaveFront オブジェクト
戻り値: WaveFront オブジェクト

[基本的な関数]
void dispTotalTime()
引数: (なし)
戻り値: (なし)
補足:対象の WaveFront オブジェクトが今まで行ってきた FFT/補間/乱数生成の計算時間

を表示します．(単位 ms)

int GetNx() const

引数: (なし)

戻り値: 横ピクセル数

int GetNy() const

引数: (なし)

戻り値: 縦ピクセル数

int GetN() const

引数: (なし)

戻り値: 総ピクセル数

double GetPx() const

引数: (なし)

戻り値: 横サンプリング間隔

double GetPy() const

引数: (なし)

戻り値: 縦サンプリング間隔

double GetLambda() const

引数: (なし)

戻り値: 波長

vec3 GetOrigin() const

引数: (なし)

戻り値: ローカル座標中心

bool GetSpace() const

引数: (なし)

戻り値: 光波の存在する空間状態フラグ．true: 実空間 false: 周波数空間

vec3 GetNormal() const

引数: (なし)

戻り値: ローカル座標z軸方向法線ベクトル

double GetWidth() const

引数: (なし)

戻り値: 光波の横方向サイズ

double GetHeight() const

引数: (なし)

戻り値: 光波の縦方向サイズ

void SetNx(const unsigned int nx)

引数: 横ピクセル数

戻り値: (なし)

void SetNy(const unsigned int ny)

引数: 縦ピクセル数

戻り値: (なし)

void SetPx(const double px)

引数: 横サンプリング間隔

戻り値: (なし)

void SetPy(const double py)

引数: 縦サンプリング間隔

戻り値: (なし)

void SetLambda(const double lambda)

引数: 波長

戻り値: (なし)

void SetSpace(const bool flug)

引数: 光波の存在する空間状態フラグ

戻り値: (なし)

void SetPixel(const unsigned int i, const unsigned int j, const std::complex<double> val)

引数: 横インデックス/縦インデックス/挿入したい値

戻り値: (なし)

void SetReal(const unsigned int i, const unsigned int j, const double val)

引数: 横インデックス/縦インデックス/挿入したい値

戻り値: (なし)

void SetImage(const unsigned int i, const unsigned int j, const double val)

引数: 横インデックス/縦インデックス/挿入したい値

戻り値: (なし)

void SetOrigin(const vec3 &vec)

引数: 挿入したいローカル座標中心

戻り値: (なし)

void SetNormal(const vec3 &vec)

引数: 挿入したい法線ベクトル

戻り値: (なし)

int idxij(const int i, const int j)

引数: 横インデックス/縦インデックス

戻り値: 要素(i, j)の存在するインデックス

void CopyParam(const WaveFront& wave)

引数: パラメータをコピーしたい WaveFront オブジェクト
戻り値: (なし)
void Init()
引数: (なし)
戻り値: (なし)
説明:標本点数を再設定した場合これをコールしてメモリを再取得
unsigned int xtoi(double x)const
引数: 横インデックスに変換したい横方向実座標
戻り値: 対応する横インデックス
unsigned int ytoj(double y)const
引数: 横インデックスに変換したい縦方向実座標
戻り値: 対応する縦インデックス
double itox(unsigned int i)const
引数:横方向実座標に変換したい横インデックス
戻り値: 対応する横方向実座標
double jtoy(unsigned int j)const
引数: 縦方向実座標に変換したい縦インデックス
戻り値: 対応する縦方向実座標
complex<double> GetPixel(unsigned int i, unsigned int j)const
引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)の複素数
double GetReal(unsigned int i, unsigned int j)const
引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)の複素数実部
double GetImage(unsigned int i, unsigned int j)const
引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)の複素数虚部
double GetAmplitude(unsigned int i, unsigned int j)const
引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)の複素数振幅
double GetIntensity(unsigned int i, unsigned int j)const
引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)の複素数強度
double GetPhase(unsigned int i, unsigned int j)const
引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)の複素数位相

double ComputeEnergy()const

引数: (なし)

戻り値: 光波分布のエネルギー

double ComputeMaxAmplitude()const

引数: (なし)

戻り値: 最大振幅．正規化などに使うがNormalize()関数があるので呼ぶことは少ない.

void Clear()

引数: (なし)

戻り値: 光波分布のゼロクリア

void DispMat(mat3 &mat)const

引数: 確認したい行列

戻り値: (なし)

void DispVec(vec3 &vec)const

引数: 確認したいベクトル

戻り値: (なし)

void DispParam()const

引数: (なし)

戻り値: (なし)

unsigned int nearPow2(int n)

引数: 2 のべき乗に丸めたい整数

戻り値: 2のべき乗に丸められた整数

WaveFront& AllSet(double val)

引数: 全ピクセルに設定する実数値

戻り値: 指定した実数値を設定された自身への参照

WaveFront& MultiplyPlaneWave(double u, double v, double phase = 0.0)

引数: 自身に乗算したい平面波の波動ベクトル u 成分/v 成分/初期位相

戻り値: 平面波の乗算された自身への参照

WaveFront& Add(const WaveFront &source)

引数: ローカル座標を考慮した加算を行いたい WaveFront オブジェクト

戻り値: 加算の行われた自身への参照

WaveFront& TransformforBrainImage()

引数: (なし)

戻り値: 縦横それぞれ反転させた自身への参照

[画像出力・ロード周りの関数]

void SaveBmp(const char* filename, Out type)

引数: 出力したい 画像ファイル名/出力タイプ

戻り値: (なし)

void Normalize()

引数: (なし)

戻り値: (なし)

WaveFront& LoadBmp(const char* filename)

引数: WaveFront オブジェクトとして読み込みたい BMP 画像ファイル名

戻り値: BMPファイルを読み込んだ自身への参照

[開口関数]

void GenerateCirc(double r)

引数: セットしたい円形開口の半径

戻り値: 円形開口のセットされた自身への参照

void GenerateRect(double wx, double wy)

引数: セットしたい矩形開口の 横幅/縦幅

戻り値: 矩形開口のセットされた自身への参照

void GenerateGaussian(double r, double n)

引数: セットしたいガウシアン分布の 1/e 幅/次数

戻り値: ガウシアン分布のセットされた自身への参照

[補間値計算周りの関数]

complex<double> GetInterpolatedValueNEAREST_NEIGHBOR(double u, double v)const

引数: 補間対象座標の 横方向座標/縦方向座標

戻り値: 最近傍補間された複素数振幅値

complex<double> GetInterpolatedValueBILINEAR(double u, double v)const

引数: 補間対象座標の 横方向座標/縦方向座標

戻り値: バイリニア補間された複素数振幅値

complex<double> GetInterpolatedValueBICUBIC(double u, double v)const

引数: 補間対象座標の 横方向座標/縦方向座標

戻り値: バイキュービック補間された複素数振幅値

complex<double> GetInterpolatedValue(double u, double v, Interp interp)const

引数: 補間対象座標の 横方向座標/縦方向座標/補間法

戻り値: 指定した補間法で補間された複素数振幅値

[FFT周りの関数]

void swap()

引数: (なし)

戻り値: (なし)

補足:FFTに用いられる象限変換を行う

void fft1D(std::unique_ptr <std::complex<double>[]>& x, int n, int func)
引数: 1DFFT を行いたい　データ/データ数/順変換・逆変換フラグ(-1: 順変換 1:逆変換)
戻り値: (なし)
void fft2D(int func)
引数: 順変換・逆変換フラグ(-1: 順変換 1:逆変換)
戻り値: (なし)
補足:スペクトルから複素振幅に戻す場合，fft2D(1)を呼んだ後，総ピクセル数で除算してください
void pitchtrans()
引数: (なし)
戻り値: (なし)
補足: 自身のサンプリング間隔を実空間での間隔・周波数空間での間隔へ互いに変換します

[角スペクトル法周りの関数]
void generateFRF(double distance)
引数: 伝搬距離
戻り値: (なし)
補足: 角スペクトル法に用いられる周波数応答関数を生成する．WaveFrontオブジェクトが周波数空間に存在することが前提
void bandlimit(double uband, double vband)
引数: 制限帯域　横幅/縦幅
戻り値: (なし)
void AsmProp(const double R)
引数: ASM で伝搬させたい距離
戻り値: (なし)
補足:実装自体は帯域制限角スペクトル法
WaveFront& ShiftedAsmProp(const WaveFront& source)
引数: シフテッド ASM で伝搬させたい WaveFront オブジェクト
戻り値: 伝搬結果を持つ自身への参照
補足: サンプリング間隔・サンプル数が一致している必要がある
WaveFront& ShiftedAsmPropAdd(const WaveFront& source)
引数: シフテッド ASM で伝搬したのち加算させたい WaveFront オブジェクト
戻り値: 伝搬結果を持つ自身への参照
補足: サンプリング間隔・サンプル数が一致している必要がある
WaveFront& ShiftedAsmPropEx(const WaveFront& source)

引数: シフテッド ASM で伝搬させたい WaveFront オブジェクト
戻り値: 伝搬結果を持つ自身への参照
補足: サンプリング間隔・サンプル数が一致している必要がない

WaveFront& ShiftedAsmPropAddEx(const WaveFront& source)
引数: シフテッド ASM で伝搬したのち加算させたい WaveFront オブジェクト
戻り値: 伝搬結果を持つ自身への参照
補足: サンプリング間隔・サンプル数が一致している必要がない

void AsmPropInFourierSpace(const double R)
引数: ASM で伝搬させたい距離
戻り値: 伝搬結果を持つ自身への参照
補足: 周波数空間内での伝搬のみ行うため，結果は手波数空間に存在

void Embed()
引数: (なし)
戻り値: (なし)
補足: WaveFrontオブジェクトのピクセル数を縦横倍にする

void Extract()
引数: (なし)
戻り値: (なし)
補足: WaveFrontオブジェクトのピクセル数を縦横半分にする

void ExactAsmProp(const double R)
引数: (なし)
戻り値: (なし)
補足: 4倍拡張ASMを用いたエイリアシングのない厳密解を得る

double GetWpow2(double u, double v)
引数: 波動ベクトル u 成分/v 成分
戻り値: w成分の2乗値
補足: エヴァネッセント領域の判定に使用

mat3 GetRotMat(const vec3 &v) const
引数: 変換元のベクトル
戻り値: 自身の法線ベクトル(ローカル座標z軸方向)への座標変換行列

mat3 GetRotMat(const vec3& global, const vec3& local)
引数: 変換元のベクトル/変換先のベクトル
戻り値: 座標変換行列

mat3 L2G()
引数: (なし)
戻り値: ローカル座標とグローバル座標の間の座標変換行列

vec3 GetUnitVector_alongX()
引数: (なし)
戻り値: ローカル座標横方向の単位ベクトルのグローバル座標
vec3 GetUnitVector_alongY()
引数: (なし)
戻り値: ローカル座標縦方向の単位ベクトルのグローバル座標
void RotInFourierSpace(WaveFront& reference, Interp interp, vec3 *carrier = nullptr)
引数: 伝搬先 WaveFront オブジェクト/補間法/キャリア周波数を保存するポインタ
戻り値: (なし)
補足: 周波数空間内での伝搬のみ行うため，結果は手波数空間に存在
void TiltedAsmProp(WaveFront& reference, Interp interp, vec3* carrier = nullptr)
引数: 伝搬先 WaveFront オブジェクト/補間法/キャリア周波数を保存するポインタ
戻り値: (なし)

[乱数化周りの関数]
WaveFront& ModRandomphase()
引数: (なし)
戻り値: 位相を乱数化しすることで散乱光となった自身への参照
double getrandom(double min, double max)
引数: 得る乱数の 最小値/最大値
戻り値: メルセンヌツイスタで生成された乱数

[csv出力周りの関数]
void SaveAsCsv(const char* fname, Axis axis, int ij)
引数: 出力したい csv ファイル名/出力軸/出力列へのインデックス
戻り値: (なし)
補足: 軸を決め，その軸に沿った指定インデックスの示すデータ列の座標/実部/虚部/振幅/位相/強度をcsv形式で出力します

[レンズ周りの関数]
WaveFront& SetQuadraticPhase(const double f)
引数: レンズの焦点距離
戻り値: 指定した焦点距離に収斂させる光波となった自身への参照
補足: 呼び出したWaveFrontオブジェクト自身がレンズの機能を持ちます
WaveFront& MultiplyQuadraticPhase(const double f)
引数: レンズの焦点距離

戻り値: 指定した焦点距離に収斂する光波となった自身への参照
補足: 呼び出したWaveFrontオブジェクトが指定した焦点距離のレンズを通過した場合の光波になります

[光波形式出力・ロード周りの関数]
void SaveAsWaveFront(const char* filename)
引数: 光波形式で出力する際の ファイル名
戻り値: (なし)
補足: txt形式で光波のパラメータと複素振幅分布を実部・虚部の組み合わせて出力します
WaveFront& LoadAsWaveFront(const char* filename)
引数: 読み込みたい光波形式の ファイル名
戻り値: 読み込んだデータを持つ自身への参照

[オペレータ]
WaveFront& operator+=(double val)
全要素の実部に指定した値を加算
WaveFront& operator-=(double val)
全要素の実部から指定した値を減算
WaveFront& operator*=(double val)
全要素の複素数に指定した値を乗算
WaveFront& operator/=(double val)
全要素の複素数を指定した値で除算
WaveFront& operator+=(const WaveFront& val)
引数のWaveFrontオブジェクトの各要素を自身の同じ位置の要素に加算
補足:全パラメータが一致している必要がある
WaveFront& operator-=(const WaveFront& val)
引数のWaveFrontオブジェクトの各要素を自身の同じ位置の要素から減算
補足:全パラメータが一致している必要がある
WaveFront& operator*=(const WaveFront& val)
引数のWaveFrontオブジェクトの各要素を自身の同じ位置の要素に乗算
補足:全パラメータが一致している必要がある
WaveFront& operator/=(const WaveFront& val)
引数のWaveFrontオブジェクトの各要素を自身の同じ位置の要素で除算
補足:全パラメータが一致している必要がある
WaveFront& operator =(const WaveFront& val)
引数のWaveFrontオブジェクトの各要素を自身の同じ位置の要素に代入

補足:全パラメータが一致している必要がある

ImagingWaveFront

ImagingWaveFront クラスは WaveFront オブジェクトを結像再生するためのクラスです. WaveFront を継承し，関数をオーバーライドしています.
デフォルトは直径 2cm かつ瞳の直径 4mm を想定しています. また, この時結像結果は 4096×4096 ピクセルで出力されます.

[コンストラクタ]

ImagingWaveFront(void)
引数: (なし)
戻り値: ImagingWaveFrontオブジェクト
補足:デフォルト値として前述のパラメータが入力されます

ImagingWaveFront(double deye, double dpupil, vec3 vp, const WaveFront image)
引数: 眼球の直径/瞳の直径/視点の座標/結像画像に変換する WaveFront オブジェクト
戻り値: ImagingWaveFrontオブジェクト

[基本的な関数]

WaveFront& SetOrigin(vec3 p)
引数: 視点の座標
戻り値: WaveFrontオブジェクト自身への参照

void SetImagingDistance(const double dd)
引数: 眼球の直径(ここでは結像距離であるためこの名称にしている)
戻り値: (なし)

void SetPupilDiameter(const double dd)
引数: 瞳の直径
戻り値: (なし)

double GetImagingDistance(void)
引数: (なし)
戻り値: 眼球の直径

double GetPupilDiameter(void)
引数: (なし)
戻り値: 瞳の直径

void SetEyeParam()
引数: (なし)

戻り値: (なし)
補足:眼球の直径2.4cm 瞳の直径6mmとなるパラメタを設定する(より現実的なパラメタと考えられる). 内部でWaveFront::Init()を呼んでいる.

[結像周りの関数]
void Imaging(vec3 p)
引数: 結像を行いたい注視点
戻り値: (なし)
補足:結像を行ってその結果をメンバであるWaveFrontオブジェクトに格納する
void View(const WaveFront& wf, const vec3 &p)
引数: 結像を行いたい WaveFront オブジェクト/注視点
戻り値: (なし)
補足:内部でImagingを呼び出す. 結像の際はこちらを呼び出す
void SetEye(WaveFront & eye, const vec3 &p)
引数: 瞳関数を書き込みたい WaveFront オブジェクト/注視点
戻り値: (なし)

Image

Image クラスは画像を扱うためのクラスです. 基本触ることはない補助的なクラスです. デフォルトで 1024×1024 ピクセルの画像を想定しています.

[コンストラクタ]
Image()
引数: (なし)
戻り値: Imageオブジェクト
補足:デフォルト値として前述のパラメータが入力されます
Image(int W, int H)
引数: 画像の 横ピクセル数/縦ピクセル数
戻り値: Imageオブジェクト

[基本的な関数]
int GetWidth() const
引数: (なし)
戻り値: 画像横ピクセル数
int GetHeight() const

引数: (なし)
戻り値: 画像縦ピクセル数

```
void* GetPixels() const
```

引数: (なし)
戻り値: 画像データの実体のポインタ

```
int idxij(const int i, const int j) const
```

引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)の存在するインデックス

```
void Write(int i, int j, double R, double G, double B, bool read = true)
```

引数: 横インデックス/縦インデックス/R 値/G 値/B 値/読み込みフラグ(true: 読み込み false: 書き出し)
戻り値: (なし)

```
vec3 Load(int i, int j)
```

引数: 横インデックス/縦インデックス
戻り値: 要素(i, j)のRGB値
[画像読み込み周りの関数]

```
Image* Read_Bmp(const char* filename)
```

引数: 読み込みたい BMP 形式の ファイル名
戻り値: 読み込んだデータを持つ自身へのポインタ

Model(+BoundingBox)

Model クラスは物体光波をするためのクラスです.
三角ポリゴン及び 24bitBMP テクスチャのみのサポートです. スペキュラーなども不可能です. 技術は既に存在します. 興味あれば是非調べてみて下さい.
BoundingBox に始まり他のクラスや構造体を使用しますが, ここでは BoundingBox のみ焦点を当てます. 他はコードを参照してください. 通常の光波計算には WaveFront クラスで事足ります.
またシェーディング及び, バインディングボックスへモデルをフィットさせる際に基準にする方向・隠面消去法の指定に関して以下の enum を持ちます.
物体光波計算時には想定領域全域を対象とする全光波・ポリゴンに基づいた回折範囲で定められる部分光波が使用されます. 詳しくは「計算機合成ホログラム・物体光波・隠面消去・ポリゴン法・サブモデル分割・スイッチバック法」などで検索してみて下さい.

出力 Shader: FLAT,SMOOTH
→フラットシェーダ/スムースシェーダ

基準方向 Direction: DWIDTH,DHEIGHT,DDEPTH
→横/縦/奥行
隠面消去法 Shield: SHILHOUETTE,EXACT
→シルエット法/完全な隠面消去法

隠面消去法はシルエット法が高速ですが，近似手法であり，隠面消去が完全ではありません．
後者は完全に隠面消去を再現できます．

[BoundingBox]

[コンストラクタ]

BoundingBox()
引数: (なし)
戻り値: BoundinBoxオブジェクト
補足:全てが0で初期化されていますのでこのままでは使い物になりません

BoundingBox(vec3 min, vec3 max)
引数: 最小座標値を要素に持つベクトル/最大座標値を要素に持つベクトル
戻り値: BoundinBoxオブジェクト

BoundingBox(double w, double h, double d, vec3 center)
引数: 横幅/高さ/奥行/中心
戻り値: BoundinBoxオブジェクト

[基本的な関数]

double GetWidth() const
引数: (なし)
戻り値: 横幅

double GetHeight() const
引数: (なし)
戻り値: 高さ

double GetDepth() const
引数: (なし)
戻り値: 奥行

[Model]

[コンストラクタ]

Model ()
引数: (なし)

戻り値: Modelオブジェクト
補足:何もしていないので初期化するか別のコンストラクタを使用してください
Model (const Model & Model)
引数: Model オブジェクト
戻り値: Modelオブジェクト
Model(const char* FileName, vec3 emv, Shader shade, BoundingBox bb, Direction dir, bool surface)
引数: MQO ファイル名/環境光ベクトル
戻り値: Modelオブジェクト

[基本的な関数]
void dispTotalTime()
引数: (なし)
戻り値: (なし)
補足:対象のModelオブジェクトがAddObjectField()で行ってきたFFT/補間/乱数生成/その他の処理時間を表示します．(単位ms)．AddObjectField()の最後で呼ばれます．
double GetPx() const
引数: (なし)
戻り値: 横サンプリング間隔
補足: 基本的に後述のAddObjectField()関数によって引数のWaveFrontオブジェクトのそれを引き継ぐ デフォルトは1um
double GetPy() const
引数: (なし)
戻り値: 縦サンプリング間隔
補足: 基本的に後述のAddObjectField()関数によって引数のWaveFrontオブジェクトのそれを引き継ぐ デフォルトは1um
double GetLambda() const
引数: (なし)
戻り値: 波長
補足: 基本的に後述のAddObjectField()関数によって引数のWaveFrontオブジェクトのそれを引き継ぐ デフォルトは633nm
void SetShieldMethod(const Shield mtd)
引数: 隠面消去法
戻り値: (なし)
補足:デフォルトはEXACT
FILE* GetFilePointer() const

引数: (なし)
戻り値: モデルファイルへのファイルポインタ
char GetBuffer() const
引数: (なし)
戻り値: ファイル解析に用いるバッファ1
std::string GetString() const
引数: (なし)
戻り値: ファイル解析に用いるバッファ2
depthListArray GetDepthlist() const
引数: (なし)
戻り値: ポリゴンを深さ順にリストにしたもの(深度リスト)
std::vector<Material> GetMaterial()　const
引数: (なし)
戻り値: マテリアルをリストにしたもの
std::vector<Object> GetObject()　const
引数: (なし)
戻り値: オブジェクトをリストにしたもの
vec3 GetEmvironment() const
引数: (なし)
戻り値: 環境光ベクトル
bool MQO_Load(const char* FileName)
引数: MQO ファイル名
戻り値: 読み込めたかどうかのブーリアン
void CalcSurfaceNV()
引数: (なし)
戻り値: 面法線ベクトルを計算してセットする
void CalcVertexNV()
引数: (なし)
戻り値: 頂点法線ベクトルを計算してセットする
補足:呼ぶ場合は必ず面法線ベクトルを計算後に読んでください．強制終了します.
void CalcPolygonCenter()
引数: (なし)
戻り値: ポリゴンの中心を計算してセットする
void CalcModelCenter()
引数: (なし)
戻り値: モデルの中心を計算してセットする

vec3 center(const vec3& p0, const vec3& p1, const vec3& p2)
引数: 3 頂点
戻り値: 重心
void GenDepthList()
引数: (なし)
戻り値: (なし)
補足:メンバのポリゴン列から深度リストを作る
void SortByDepth(depthListArray& list)
引数: 深度リスト
戻り値: (なし)
補足:奥から手前に向かってソートする
void DivideByDepth(depthListArray& front, depthListArray& back, double z)
引数: 手前深度リスト/奥深度リスト/分割深度基準
戻り値: (なし)
補足:基準より手前か奥かで深度リストを分割する
void AccommodatePolygonInBB()
引数: (なし)
戻り値: (なし)
補足:ポリゴンをメンバ変数のBoundingBoxに設定した基準方向に基づいてフィットさせる

[物体光波生成周りの関数]
void RotInFourierSpaceForward(const WaveFront& source, WaveFront& reference, vec3* c, Interp interp)
引数: 伝搬元 WaveFront/伝搬先 WaveFront/キャリア周波数を保存するポインタ/補間法
戻り値: (なし)
補足:平行平面上の光波をポリゴン面上に光波の回転変換で伝搬する
void RotInFourierSpaceBackward(const WaveFront& source, WaveFront& reference, const vec3& c, Interp interp)
引数: 伝搬元 WaveFront/伝搬先 WaveFront/キャリア周波数を保存するポインタ/補間法
戻り値: (なし)
補足: ポリゴン面上の光波を平行平面上に光波の回転変換で伝搬する
mat3 RotMatFromG2L(const vec3 &global, const vec3 &local)
引数: グローバル座標系での z 軸方向単位ベクトル
戻り値: (なし)
補足: ポリゴン面上の光波を平行平面上に光波の回転変換で伝搬する

mat3 G2L()
引数: (なし)
戻り値: グローバル座標系からローカル座標への変換行列
BoundingBox GetDiffractionRect(const double targetZ)
引数: 目標位置の z 座標
戻り値: 注目しているポリゴンから測った回折範囲
補足:このポリゴンはSetCurrentPolygon()により物体光波計算中に設定されている
void SetCurrentPolygon(const depthList dpl)
引数: 深度リストの要素
戻り値: (なし)
補足:深度リストの要素からポリゴンを設定する
void ClipSubfield(WaveFront& sub, WaveFront& frame, bool dir = true)
引数: 部分光波/全光波/全光波から部分光波を切り出すか, 部分光波を前光波に加算するかを示すフラグ(true: 切り出す false: 加算する)
戻り値: (なし)
void AddFieldToMFB(WaveFront& mfb)
引数: 全光波
戻り値: (なし)
補足: 全光波に注目するポリゴンからの光波を加算する
void AddObjectFieldPersubModel(WaveFront& mfb, depthListArray& list)
引数: 全光波/深度リスト
戻り値: (なし)
補足: 分割した深度リストのポリゴンを用いて全光波に注目するポリゴンからの光波を加算する
void CalcCenterOfModel(depthListArray& Model)
引数: Model オブジェクト
戻り値: (なし)
補足: 設定した3Dモデルの中心を計算する
void AddObjectField(WaveFront& mfb, const unsigned int div, const mat3 &rot, bool exact = true, bool back = false)
引数: 全光波/モデル分割数/モデル回転行列/4 倍拡張 ASM によるモデル間伝搬を行うか(true: 行う false: 行わない)/背景を持つか(true: 持つ false: 持たない)
戻り値: (なし)
補足: 設定した3Dモデルの物体光波を全光波に加算する
void ExShieldingAddingField(WaveFront &pfb)
引数: 部分光波

戻り値: (なし)
補足: 精密に陰面消去を行った光波を部分光波に加算する
void SilhouetteShieldingAddingField(WaveFront &pfb)
引数: 部分光波
戻り値: (なし)
補足: 近似的に陰面消去を行った光波を部分光波に加算する
void GeneratePolygonField(WaveFront& field, const CurrentPolygon& polyL)
引数: ポリゴン光波/ポリゴン
戻り値: (なし)
補足: ポリゴンから光波を生成する
bool IsInTriangle(vec3 p, const CurrentPolygon& polyL)
引数: 点/ポリゴン
戻り値: 点がポリゴンの上にあるかのブーリアン(true: ある false: ない)
void PaintTriangle(WaveFront& tfb, const CurrentPolygon &polyL, double amp)
引数: ポリゴン面上の部分光波/ポリゴン/輝度値
戻り値: (なし)
補足:部分光波のポリゴンが占める部分を指定した輝度値で塗りつぶす
void MultiplyAperture(WaveFront& tfb, const CurrentPolygon& polyL)
引数: ポリゴン面上の部分光波/ポリゴン
戻り値: (なし)
補足:部分光波にポリゴン形状の開口でシールドする
int LineFunc(int x, int y, int x1, int y1, int x2, int y2)
引数: 位置を判断したい点の x 座標/y 座標/基準点 1 の x 座標/基準点 1 の y 座標/基準点 2 の x 座標/基準点 2 の y 座標
戻り値: 上か下か(1: 上 -1:下)
bool PolygonIsVisible()
引数: (なし)
戻り値: ポリゴンが見えるか見えないか(true: みえる false:見えない)
補足:空間周波数に基づいてそのポリゴンからの光がホログラム面に回折して到達するかを判断する
void MarkingRectangularPointsInFourierSpace(std::vector<vec3> &vec)
引数: ベクトルのリスト
戻り値: (なし)
補足:Ewald球表面上に打ち付けたドットを格納したベクトルが返される.
PolygonIsVisible()で使用. デフォルトで50×50点のドットを打つ.
BoundingBox GetBoundingBox(const std::vector<vec3> &vec)

引数: ベクトルのリスト
戻り値: リスト内の各座標を内包するBoundingBox

void Shading(WaveFront& field, const CurrentPolygon& polyL)
引数: 指定した手法でシェーディングを行う WaveFront オブジェクト/使用するポリゴン
戻り値: (なし)
補足:内部でFlatShading()/SmoothShading()を場合分けして使用する

void FlatShading(WaveFront& field, const CurrentPolygon& polyL)
引数: フラットシェーディングを行う WaveFront オブジェクト/使用するポリゴン
戻り値: (なし)

double GetCorrectedAmplitude(WaveFront& tfb, double brt)
引数: 輝度補正を行う WaveFront オブジェクト/使用する輝度
戻り値: (なし)

void SmoothShading(WaveFront& field, const CurrentPolygon& polyL)
引数: スムースシェーディングを行う WaveFront オブジェクト/使用するポリゴン
戻り値: (なし)

void Mapping(WaveFront& field, const CurrentPolygon& polyL)
引数: テクスチャマッピングを行う WaveFront オブジェクト/使用するポリゴン
戻り値: (なし)

void SetUp(const mat3 &rot)
引数: モデルを回転する行列
戻り値: (なし)
補足:物体光波計算時に呼ばれる

vec3 IntersectPoint(const Ray &ray)
引数: 直線オブジェクト
戻り値: 設定しているポリゴンとの交点座標

[オペレータなど]

Model & operator +=(const vec3& vec)
モデルの全ポリゴンに指定した座標を加算することで座標変換する

Model & operator *=(const mat3& mat)
モデルの全ポリゴンに指定した行列を乗算することで座標変換する

void mul(std::vector<vec3>& vec, const mat3& mat)
第一引数のベクトルのリストに指定した行列を乗算することで座標変換する

void sub(std::vector<vec3>& vec, const vec3& vv)
第一引数のベクトルのリストから指定した座標を減算することで座標変換する

void fouriermul(std::vector<vec3>& vec, const mat3& mat)

第一引数のベクトルのリストに指定した行列を乗算することで周波数空間上で座標変換する

```
void fouriersub(std::vector<vec3>& vec, const vec3& vv)
```

第一引数のベクトルのリストから指定した座標を減算することで周波数空間上で座標変換する