データ処理ライブラリ説明書

【ご注意】

１．本マニュアルの内容の一部または全部を無断転載することは禁止されています

２．本マニュアルの内容に関しては将来予告なしに変更することがあります

３．本マニュアルの内容について万全を期して作成しております、万一ご不審な点や誤り、記載漏れなどお気づきのことがございましたら、ご連絡ください

４．運用した結果の影響に関しては、３．項にかかわらず責任を負いかねますのでご了承ください

目次

1．概要

本ドキュメントは、提供されるデータ処理ライブラリについて述べたものです。

プロジェクトには、２つのデータ処理ライブラリとサポートクラスが含まれています。

①Soft Matching
　ステレオマッチングを行います。

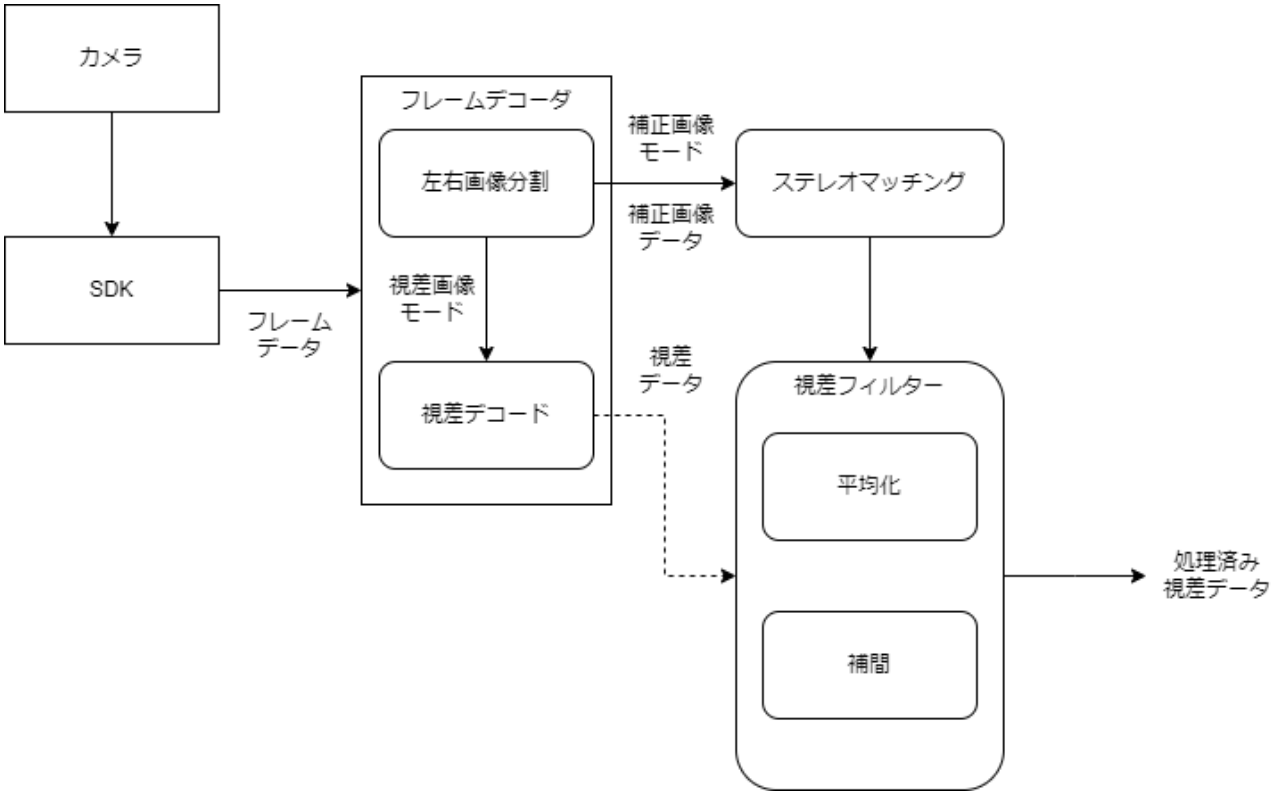
②Disparity Filter
　視差の平均化、補間処理を行います。

③Frame Decoder
　視差データの展開を行うサポートクラスです

本プロジェクトにおいては、それぞれ以下のモジュール名で提供されています。

処理ライブラリ名	モジュール名	DLL 名
Soft Matching	IscBlockMatching	IscBlockMatching.dll
Disparity Filter	IscDisparityFilter	IscDisparityFilter.dll
Frame Decoder	IscFrameDecoder	IscFrameDecoder.dll

１．１　処理の流れ



（1）SDK からフレームデータを取得します

ISC SDK の関数 を呼び出して、フレームデータを取得します。

シャッターの制御モードによって使用する関数が異なります。

カメラ	シャッターの制御モード	
	マニュアル・シングルシャッター	ダブルシャッター
VM	GetFullFrameInfo2()	GetFullFrameInfo3()
XC	GetFullFrameInfo2()	GetFullFrameInfo4()

（2）フレームデータを左右画像に分割します

（視差画像モードの場合、基準画像と視差エンコードデータに分割します）

フレームデコーダの関数 decodeFrameData()を呼び出します。

視差画像モードかつダブルシャッターモードでは、取得した現フレームデータと前フレームデータを、関数 decodeFrameData()を呼び出してそれぞれ分割します。

①補正画像モードの場合：

a. 左右の補正画像をステレオマッチングします

左右の補正画像を引数にして、ステレオマッチングの関数 matching()を呼び出します。

b. 視差データを取得します

ステレオマッチングの関数 :getBlockDisparity()を呼び出して、視差データを取得します。

c. 視差データをフィルター処理（平均化・補完）します

視差データを入力として、視差フィルターの関数 averageDisparityData()を呼び出します。

関数 averageDisparityData()の出力する処理済の視差データを取得します。

②視差画像モードの場合：

a. 視差エンコードデータをデコードします

フレームデータの左右画像分割で取得した視差エンコードデータを、関数 getDisparityData()を呼び出してデコードします。

関数 getDisparityData()は、デコード後内部で視差フィルターの関数 averageDisparityData()を呼び出して、視差データを処理（平均化・補完）します。

この関数 decodeDisparityData()の出力する処理済の視差データを取得します。

ダブルシャッターモードでは、現フレームと前フレームの視差エンコードデータを、フレームデコーダの関数 getDoubleDisparityData()を使用してデコードします。

関数 getDoubleDisparityData()は、現フレームと前フレームの視差データを合成し、内部で視差フィルターの関数 averageDisparityData()を呼び出して処理（平均化・補完）します。

＊視差画像モード（カメラ側でステレオマッチングを行う）の場合、カメラのオクリュージョン除去と特異点除去をオフとして使用します。

＊ダブルシャッターモードは、視差画像モードの場合のみ使用可能です。

１．２　関数の使用手順

フレームデコーダ、ステレオマッチング、視差フィルターを使用する場合は、開始時と終了時に次の関数を呼び出します。

（１）開始時に呼び出す関数

・フレームデコーダ

　　initialize()

・ステレオマッチング

　　initialize()

　　createMatchingThread()

　　setUseOpenCLForMatching()　　＊リアルタイムに設定変更可能

　　setMatchingParameter()　　＊リアルタイムに設定変更可能

　　setBackMatchingParameter()　　＊リアルタイムに設定変更可能

・視差フィルター

　　initialize()

　　createAveragingThread()

　　setUseOpenCLForAveragingDisparity()＊リアルタイムに設定変更可能

　　setAveragingParameter()　　＊リアルタイムに設定変更可能

　　setAveragingBlockWeight()　　＊リアルタイムに設定変更可能

　　setComplementParameter()　　＊リアルタイムに設定変更可能

　　setEdgeComplementParameter()　　＊リアルタイムに設定変更可能

　　setHoughTransformParameter()　　＊リアルタイムに設定変更可能

（２）終了時に呼び出す関数

・フレームデコーダ

　　finalize()

・ステレオマッチングライブラリ

　　deleteMatchingThread()

　　finalize()

・視差フィルター

　　deleteAveragingThread()

　　finalize()

2．Frame Decoder

2．1　フレームデータ取得

画像フレームデータは、SDK の以下の関数を呼び出して取得します。

関数	カメラ	内容
GetFullFrameInfo2()	VM/XC	現在のフレームを取得する場合
GetFullFrameInfo3()	VM	現在と一つ前のフレームデータを取得する場合
GetFullFrameInfo4()	XC	在と一つ前のフレームデータを取得する場合

ダブルシャッターモードの場合は、GetFullFrameInfo3()または GetFullFrameInfo4()を呼び出し、現在と一つ前のフレームデータを取得します。
この 2 つのフレームデータは、一方が高感度シャッターで、他方が低感度シャッターのフレームデータです。ダイナミックレンジの広い画像を取得するために、高感度シャッターと、低感度シャッターのデータを合成します。
高感度、低感度の区別は、同時に取得するシャッターの露光値とゲイン値で判別します。

＊高感度は暗い被写体用、低感度は明るい被写体用です。
＊現バージョンでは、補正画像の合成を行ってません。
　合成補正画像として、高感度シャッターの飽和領域を低感度の画像で補完した画像を出力します。

２．２　フレームデータの形式

２．２．１　フレームサイズ

VM：(752 x 480 x 2 バイト)

XC：(1280 x 720 x 2 バイト)

＊左右画像分：1 画素当たり 1 バイト

２．２．３　データ並び

（１）補正画像モードの場合：

左カメラ画像と、右カメラ画像の輝度データが交互に並ぶ。

＊ 輝度値は 8 ビット幅　256 階調

データの先頭は画像原点のデータ。原点は画像右下、データは画像の右下から左上へ並ぶ。

ライン＼カラム	0	1	2	3	4	5	6	7
0	[0]	[0]	[1]	[1]	[2]	[2]	[3]	[3]
1	[0]	[0]	[1]	[1]	[2]	[2]	[3]	[3]
:	:	:	:	:	:	:	:	:

偶数カラム：左カメラ画像の画素単位の輝度データ

奇数カラム：右カメラ画像の画素単位の輝度データ

フレームデータを分割して、左右カメラの画像データを取得する。

左カメラデータ

ライン＼カラム	0	1	2	3
0	[0]	[1]	[2]	[3]
1	[0]	[1]	[2]	[3]
:	:	:	:	:

右カメラデータ

ライン＼カラム	0	1	2	3
0	[0]	[1]	[2]	[3]
1	[0]	[1]	[2]	[3]
:	:	:	:	:

（2）視差画像モードの場合：

視差エンコードデータと、右カメラ画像の輝度データが交互に並ぶ。

データは、画像の右下から左上へ並ぶ。

ライン＼カラム	0	1	2	3	4	5	6	7
0	[0]	[0]	[1]	[1]	[2]	[2]	[3]	[3]
1	[0]	[0]	[1]	[1]	[2]	[2]	[3]	[3]
:	:	:	:	:	:	:	:	:

偶数カラム：視差エンコードデータ

奇数カラム：右カメラ画像の画素単位の輝度データ

フレームデータを分割して、視差エンコードデータと右カメラの画像データを取得する。

視差エンコードデータ

ライン＼カラム	0	1	2	3
0	[0]	[1]	[2]	[3]
1	[0]	[1]	[2]	[3]
:	:	:	:	:

右カメラ画像データ

ライン＼カラム	0	1	2	3
0	[0]	[1]	[2]	[3]
1	[0]	[1]	[2]	[3]
:	:	:	:	:

２．２．４　視差エンコード形式について

4 カラム（4 バイト）で 4 x 4 画素分の 1 視差ブロックの視差データを構成します。

（1 視差ブロック当たり 4 バイト）

視差ブロックのサイズは 4 x 4 画素であり、視差値は画素単位ではありません。4x4 画素分はすべて同じ視差値となります。

4 x 4 画素（16 バイト）に対し、視差データのデータ量は 4 バイトです。

視差データは、4 ライン（4 の倍数ライン）ごとに格納され、その他のラインは未使用です。

＊ライン 1、2、3 は、未使用

カラム ライン	0	1	2	3	4	5	6	7
0	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
1	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
2	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
3	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
4	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
5	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
:								

ライン：0 4 8 ・・

カラム 0 [0]　　視差値の整数部

カラム 1 [1]　　[7:4]　上位 4 ビット 視差値の小数部 (1/16 単位)

カラム 2 [2]　　[7:4]　マスクビット　　ブロック 4 ライン目 (4 画素分)

[3:0]　ブロック 3 ライン目 (4 画素分)

カラム 3 [3]　　[7:4]　ブロック 2 ライン目 (4 画素分)

[3:0]　ブロック 1 ライン目 (4 画素分)

4x4 画素マスクビット画素位置

	0	1	2	3
1Line 目	0	1	2	3
2Line 目	4	5	6	7
3Line 目	0	1	2	3
4Line 目	4	5	6	7

4 バイトの視差データのうち、カラム 0[0]とカラム 1[1]の 2 バイトは、視差値、残りの 2 バイトはマスクビットです。マスクビットは、輝度エッジ（DCDX）を示しています。

エッジの弱い画素のマスクビットは 0 になります。

輝度エッジ（DCDX）の強度は、隣り合う画素の輝度値の差です。

マスクビット 0 の画素の視差を、”視差なし”（視差値ゼロ）とすることで、エッジを強調することができます。

4 x 4 画素の視差ブッロクを画素へ展開すると以下のようなイメージになります。

カラム ライン	0	1	2	3	4	5	6	7
0	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
1	[4]	[5]	[6]	[7]	[4]	[5]	[6]	[7]
2	[8]	[9]	[10]	[11]	[8]	[9]	[10]	[11]
3	[12]	[13]	[14]	15]	[12]	[13]	[14]	15]
4	[0]	[1]	[2]	[3]	[0]	[1]	[2]	[3]
5	[4]	[5]	[6]	[7]	[4]	[5]	[6]	[7]
:								

２．２．５　フレームデコーダの処理について

フレームデコーダは、画像フレームデータの分割と、視差データをデコードする処理を提供する。

提供する関数は次の２種類である。

画像フレームデータを画像データまたは視差エンコードデータに分割する

関数：decodeFrameData()

視差エンコードデータをデコードして視差データを取得する

関数：getDisparityData()　　＊シングルシャッター用

関数：getDoubleDisparityData()　　＊ダブルシャッター用

シングルシャッターとダブルシャッターの関数操作の違いを、以下のコードに示す。

```
////// バッファの獲得 //////
// 画像フルフレーム
unsigned char *pFullFrame = (unsigned char *)malloc(2 * 720 * 1280);
// 左画像データ（視差画像モードの場合は視差画像データ）
unsigned char *pLeftImage = (unsigned char *)malloc(720 * 1280);
// 右画像データ
unsigned char *pRightImage = (unsigned char *)malloc(720 * 1280);
// 視差値データ（画素単位）
float *pDepthData = (float *)malloc(720 * 1280 * sizeof(float));
// 視差値データ（視差ブロック単位）
float *pBlockDepthData = (float *)malloc(720 * 1280 * sizeof(float));
// 視差値 (1000 倍サブピクセル精度整数 ) データ（視差ブロック単位）
int *pBlockValue = (int *)malloc(720 * 1280 * sizeof(int));
// コントラストデータ（視差ブロック単位）
int *pBlockContrast = (int *)malloc(720 * 1280 * sizeof(int));

////// シングルシャッターの画像フレーム処理 //////
// SDK から画像フレームデータ取得する
int ret = GetFullFrameInfo2(pFullFrame);
if (ret == ISC_OK) {
  // 画像フレームデータを画像データまたは視差エンコードデータに分割する
  ISCFrameDecoder::decodeFrameData(720, 1280, pFullFrame, pRightImage,
pLeftImage);
  //// 視差画像モードの場合 ////
  // 視差エンコードデータをデコードして視差データを取得する
  ISCFrameDecoder::getDisparityData(720, 1280, pRightImage, pLeftImage,
       pLeftImage, pDepthData, pBlockDepthData, pBlockValue, pBlockContrast);
     :
     : < 視差データ処理 >
}
```

```
////// バッファの獲得 //////
// 現フルフレームの
unsigned char *pFullFrameCur = (unsigned char *)malloc(2 * 720 * 1280);
// 前フルフレーム
unsigned char *pFullFramePrev = (unsigned char *)malloc(2 * 720 * 1280);
// 現右画像データ
unsigned char *pRightImageCur = (unsigned char *)malloc(720 * 1280);
// 前右画像データ
unsigned char *pRightImagePrev = (unsigned char *)malloc(720 * 1280);
// 現視差エンコードデータ
unsigned char *pDepthEncodeCur = (unsigned char *)malloc(720 * 1280);
// 前視差エンコードデータ
unsigned char *pDepthEncodePrev = (unsigned char *)malloc(720 * 1280);
// 右画像データ
unsigned char *pRightImage = (unsigned char *)malloc(720 * 1280);
// 左画像（視差画像）データ
unsigned char *pLeftDispImage = (unsigned char *)malloc(720 * 1280);
// 視差値データ（画素単位）
float *pDepthData = (float *)malloc(720 * 1280 * sizeof(float));
// 視差値データ（視差ブロック単位）
float *pBlockDepthData = (float *)malloc(720 * 1280 * sizeof(float));
// 視差値 (1000 倍サブピクセル精度整数 ) データ（視差ブロック単位）
int *pBlockValue = (int *)malloc(720 * 1280 * sizeof(int));
// コントラスト（視差ブロック単位）
int *pBlockContrast = (int *)malloc(720 * 1280 * sizeof(int));

////// ダブルシャッターの画像フレーム処理 //////
// SDK からダブルシャッターの画像フレームデータ取得する
RawSrcData rawSrcDataCur;
RawSrcData rawSrcDataPrev;
rawSrcDataCur.image = pFullFrameCur;
rawSrcDataPrev.image = pFullFramePrev;
int ret = GetFullFrameInfo4(&rawSrcDataCur, &rawSrcDataPrev, 0);
if (ret == ISC_OK) {
  // 画像フレームデータを画像データまたは視差エンコードデータに分割する
  ISCFrameDecoder::decodeFrameData(720, 1280, rawSrcDataCur.image, pRightImageCur,
pDepthEncodeCur);
  // 画像フレームデータを画像データまたは視差エンコードデータに分割する
  ISCFrameDecoder::decodeFrameData(720, 1280, rawSrcDataPrev.image, pRightImagePrev,
pDepthEncodePrev);
  //// 視差画像モードの場合 ////
  // 視差エンコードデータをデコードして視差データを取得する
  ISCFrameDecoder::getDoubleDisparityData(720, 1280,
      pRightImageCur, pDepthEncodeCur, rawSrcDataCur.exposure, rawSrcDataCur.gain,
      pRightImagePrev, pDepthEncodePrev, rawSrcDataPrev.exposure, rawSrcDataPrev.gain,
      pRightImage, pLeftDispImage, pDepthData, pBlockDepthData, pBlockValue,
pBlockContrast);
      :
      :< 視差データ処理 >
}
```

ダブルシャッターの場合、getDoubleDisparityData()は、合成した視差データと補正画像を出力しますが、関数：setDoubleShutterOutput()使って、高感度側、低感度側へ切り替えることができます。

また、センサーが輝度高解像度（High Resolution/XC）モードで使用されている場合に補正画像を見やすい側へ自動で切り替わる設定も提供されます。

２．３　パラメータ

パターンが弱い視差ブロックでは、誤った視差が検出される場合ある。

コントラストの閾値を指定して、デコードの段階で除去する。

コントラストが、この閾値未満の場合は、そのブロックの視差を”視差なし”（視差値ゼロ）になる。

コントラスト閾値　　0 以上の整数（デフォルト：40）

補足）

パターンの強度評価に次のコントラスト C を使用する。

C = (Lmax - Lmin - Lofs) / Lave

Lmax：マッチングブロック内の輝度の最大値

Lmin：マッチングブロック内の輝度の最小値

Lave：マッチングブロック内の輝度の平均値

Lofs：C が明るさに依存しないようにするオフセット

＊Lofs は、解像度によって異なる 。480x752 では 1.8、720x1280 では 1.2 固定

閾値の指定には、C を 1000 倍した値を指定する。

センサーが輝度高解像度（High Resolution）モードで使用されている場合、

輝度の最大値：Lmax と最小値：Lmin の差が、明るさによって変わらなくなる。

そのため、High Resolution モードのコントラスト C を以下の式で計算する。

C = (Lmax - Lmin - Lofs) / 255

計算を切り替えるパラメータを同時に指定する。

パラメータ推奨値

パラメータ	VM(480x752)	XC(720x1280)
コントラスト閾値	45	40
センサー輝度高解像度モード ＊ High Resolution モード使用時は 1 にする	0 1	0 1

3．Soft Matching

3．1　ステレオマッチング処理

3．1．1　ステレオマッチングパラメータについて

入力される右（基準）カメラと左（比較）カメラの補正画像に対して、以下のパラメータを使ってステレオマッチングを行います

No	内容	範囲
1	視差ブロック高さ（画素）	1 以上、512 未満の整数（デフォルト：256）
2	視差ブロック高さ（画素）	2 以上、64 以下の整数（デフォルト：4）
3	視差ブロック幅（画素）	2 以上、64 以下の整数（デフォルト：4）
4	マッチングブロック高さ（画素）	2 以上、64 以下の整数（デフォルト：4）
5	マッチングブロック幅（画素）	2 以上、64 以下の整数（デフォルト：4）
6	視差ブロック横オフセット（画素	0 以上、(マッチングブロック幅 – 視差ブロック幅)以上の整数 （デフォルト：0）
7	視差ブロック縦オフセット（画素）	0 以上、(マッチングブロック高さ – 視差ブロック高さ)以上の整数 （デフォルト：0）
8	コントラスト閾値	0 以上の整数（デフォルト：40）
9	センサー輝度高解像度モード	0：未使用 または 1：使用中（デフォルト：0）

入力される左右の補正画像は 256 階調のグレースケールです。
また、左右の補正画像はエピポーラ線が水平であることが前提です。
（対象の物体は、左右画像で同じ高さに見えている）
パラメータの詳細は以降で説明します。

３．１．２　ステレオマッチングの流れについて

ステレオマッチングの流れは以下のようになります。

①右（基準）カメラ画像から視差を求めたい小領域（ブロック）を決める
②左（比較）カメラ画像の中からそのブロックと最も類似している位置を求める
③前後の類似度を使ってサブピクセルを推定して視差値とする
④画像全域に亘ってブロックの視差値を求める
⑤ブロックごとに求めた視差を画像サイズの画素へ展開する

３．１．３　パラメータの扱いについて

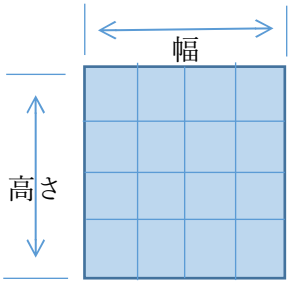
ステレオマッチング処理でやっている処理及び、それぞれのパラメータの扱いを説明します。

（１）マッチングブロックのサイズを決める

この例は、高さ、幅のサイズが 4x4 画素のブロックを示します。

サイズを小さくするとマッチングに失敗します。逆に、大きくすれば、視差の分解能が落ちます。

マッチング精度を見て、適切なサイズにする必要があります。



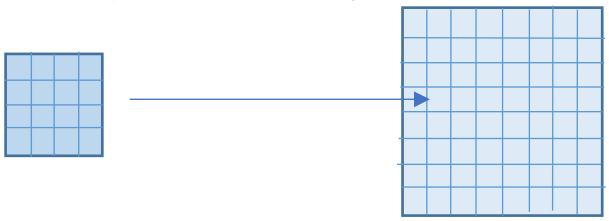
マッチングブロック

（２）ブロックの類似度を評価する

マッチング評価関数には、

- SAD（Sum of Absolute Deiffernce）、
- ZNCC（Zero-mean Normalized Cross-Correlation）
- SSD (Sum of Squared Difference)

などありますが、SSD を用います。



右画像　　左画像

SSD は、以下の式で類似度を算出します。

右画像の輝度を T(i,j)、左画像の輝度を I(i,j)とし、右画像ロック輝度平均を TA、 左画像ブロック輝度平均を IA とします。

$$R_{SSD}=\sum_{j=0}^{N-1}\sum_{i=0}^{M-1}((T(i,j)-TA)-(I(i,j)-IA))^2$$

ブロック内の同じ位置にある輝度の差分の二乗和を求め、それを領域全体で足し合わせます。

画像の明るさの影響を抑えるために輝度の平均を引いておきます。

SSD の値が小さいほど、類似している位置となります。

その位置から視差を求めるますが、そのままではピクセル精度（整数）です。

（3）サブピクセルを推定する

SSD の最小値とその前後の値を使ってサブピクセルを推定します。

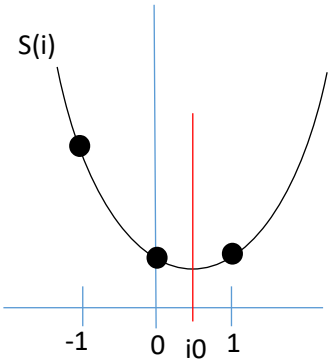
これら 3 点から放物線を近似し、その最小値の位置をサブピクセルとします。

放物線近似から次の式でサブピクセル i0 を求めます。

$$S(i) = a(i - i0)^2 + S0$$

サブピクセル i0 は、

$$i0 = (S(1) - S(-1)) / (2 \times S(-1) - 4 \times S(0) + 2 \times S(1))$$



（4）視差値を求める

視差とは、右画像のブロックが左画像でどれだけずれているか、その差です。

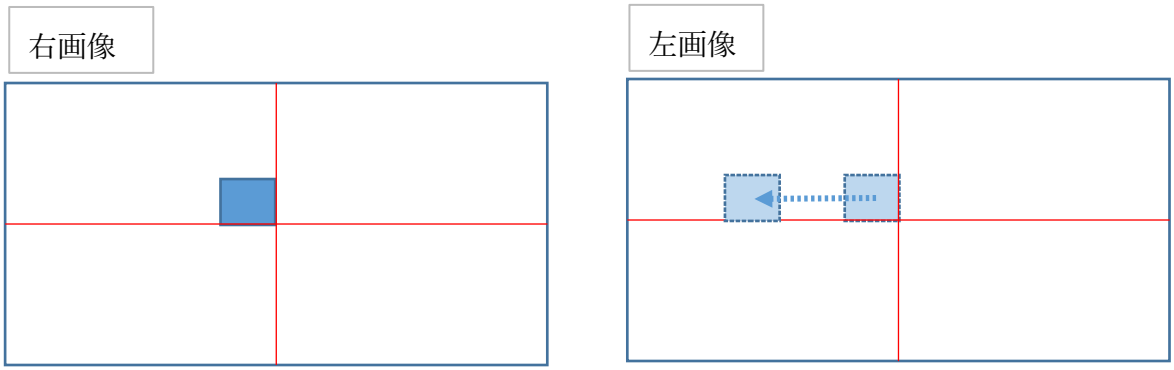
その差を求めるために、左画像の上を真っすぐ横に探索します。

左画像から類似するブロックを、予め決められた範囲で見つけ出し、サブピクセルまで推定し、視差値となります。

この予め決められた範囲が探索幅です。

近い距離の物体の視差を求めようとした場合には、この範囲を広げる必要があります。

ただし、広げれば計算時間が長くなります。



（５）視差の粒度を上げる

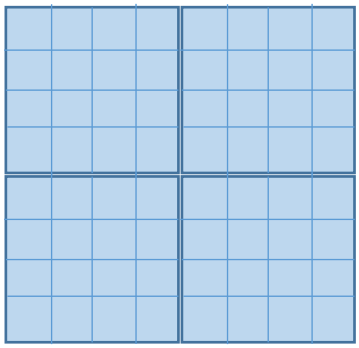
視差を求めたいブロックをタイル状に並べた場合は、視差の粒度はブロックのサイズで決まってしまいます。

ブロックを重ね合わせる（オーバーラップ）ことで、粒度を上げることができます。

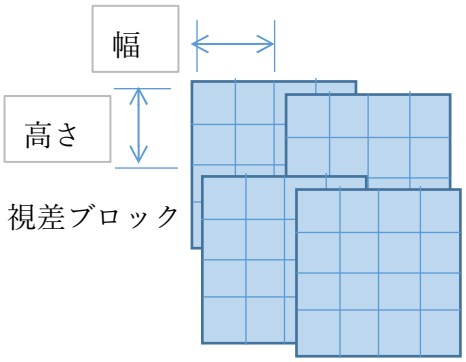
この例では、マッチング評価を行うブロックのサイズは 4x4 画素であり、重ね合わせることで得られるブロックのサイズは 2x2 画素になります。

このブロックを視差ブロックと呼びます。

マッチングブロックと視差ブロックのサイズを同じとすれば、タイル状に並べることになります。



タイル状に並べたブロック



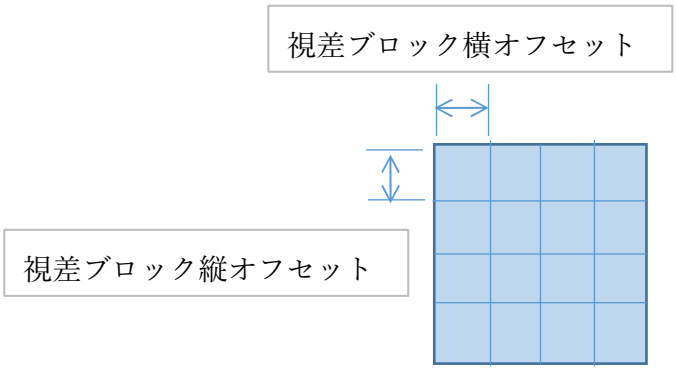
オーバーラップブロック

オーバーラップブロックでは、求めた視差を画像へ展開する場合、その位置を指定します。

それはマッチングブロックと視差ブロックのサイズで決まります。

この例は、横オフセット 1 画素、縦オフセット 1 画素です。

視差ブロックはマッチングブロックの中央になり、その位置で展開されます。



（6）弱パターンのノイズを除去する

パターンが弱い領域は、ノイズに埋もれ、マッチングがうまくいかなくなります。

誤った視差を出力しないようにマッチングの段階で ”視差なし”（視差値ゼロ）にしてします。

パターンの強度評価に次のコントラスト C を使用します。

C = (Lmax - Lmin - Lofs) / Lave

Lmax：マッチングブロック内の輝度の最大値

Lmin：マッチングブロック内の輝度の最小値

Lave：マッチングブロック内の輝度の平均値

Lofs：C が明るさに依存しないようにするオフセット

＊Lofs は、解像度によって異なる 。480x752 では 1.8、720x1280 では 1.2 固定

右画像のブロックの C が指定された閾値未満の場合に ”視差なし”（視差値ゼロ）にします。

閾値の指定には、C を 1000 倍した値を指定します。

また、輝度の最大値が 20 未満の場合、コントラスト C をゼロ、 ”視差なし”（視差値ゼロ）にします。

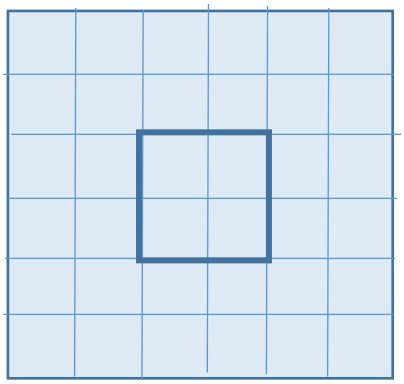
センサーが輝度高解像度（High Resolution）モードで使用されている場合では、輝度の最大値：Lmax と最小値：Lmin の差が、明るさによって変わらなくなります。

そのため、High Resolution モードのコントラスト C を以下の式で計算します。

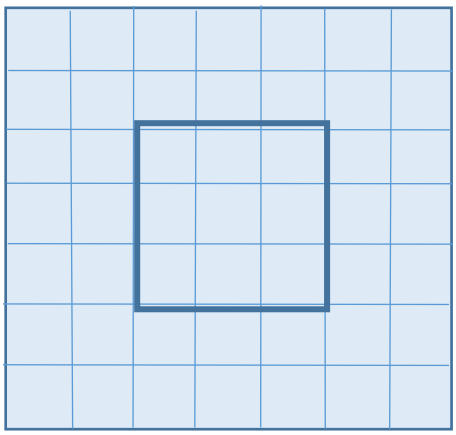
C = (Lmax - Lmin - Lofs) / 255

計算を切り替えるパラメータを同時に指定します。

（7）パラメータ推奨値



VM (480 x 752) のブロック



XC (720 x 1280) のブロック

パラメータ	VM(480x752)	XC(720x1280)
マッチング探索幅（画素）	112	256
視差ブロック高さ（画素）	2	3
視差ブロック幅（画素）	2	3
マッチングブロック高さ（画素）	6	7
マッチングブロック幅（画素）	6	7
視差ブロック横オフセット（画素）	2	2
視差ブロック縦オフセット（画素）	2	2
コントラスト閾値	45	40
センサー輝度高解像度モード ＊ High Resolution モード使用時は 1 にする	0 1	0 1

３．２　バックマッチング処理

基本的には、片側の画像を基準にして視差を推定します。
ステレオマッチング処理で説明したように、右カメラの画像を基準としています。（ここでは、これをフォアマッチングと呼びます）
これに対し、バックマッチングでは、左カメラの画像を基準にして視差を同時に推定します。
バックマッチングも、フォアマッチングも視差の推定方法は同じです。
バックマッチングの目的は、フォアマッチングの視差の正当性の評価です。
正当性の評価によって、オクルージョンによる誤った視差と、視野外の探索による誤った視差を除去することができます。
＊フォアマッチングのみの場合、誤った視差を検出しないように、視野の外へ出るような探索をしません。
評価後のフォアマッチングの視差データが、ステレオカメラの視差データとなります。

３．２．１　バックマッチングパラメータについて

バックマッチング処理のパラメータは、正当性の評価に使用するパラメータです。

No	内容	範囲
1	バックマッチング視差評価領域幅（片側）（画素）	1 以上、3 以下の整数（デフォルト：1）
2	バックマッチング評価正当視差値幅（画素）	2 以上、64 以下の整数（デフォルト：3）
3	バックマッチング評価視差正当率（%）	0 以上、100 以下の整数（デフォルト：30）
4	バックマッチング評価視差ゼロ率（%）	0 以上、100 以下の整数（デフォルト：60）

３．２．２　バックマッチングの流れについて

バックマッチングの流れは以下になります。
①左カメラ画像を基準にして画像全域に亘ってブロックの視差値を求めます
②ブロックの視差値を用いて右カメラ画像上での対応位置を求め、その位置のブロックの視差値とします
＊対応位置へ変換された視差データは、バックマッチングの視差データと呼びます
③　② を全ブロックに対して行います
＊バックマッチングの視差データは、フォアマッチングの視差データと重なります
④フォアマッチング、バックマッチングの視差を比較し、指定されたパラメータで正当性を評価します
⑤正当でない場合は、フォアマッチングのブロックの視差を ”視差なし”（視差値ゼロ）とします

３．２．３　パラメータの扱いについて

それぞれのパラメータの扱いを説明します。

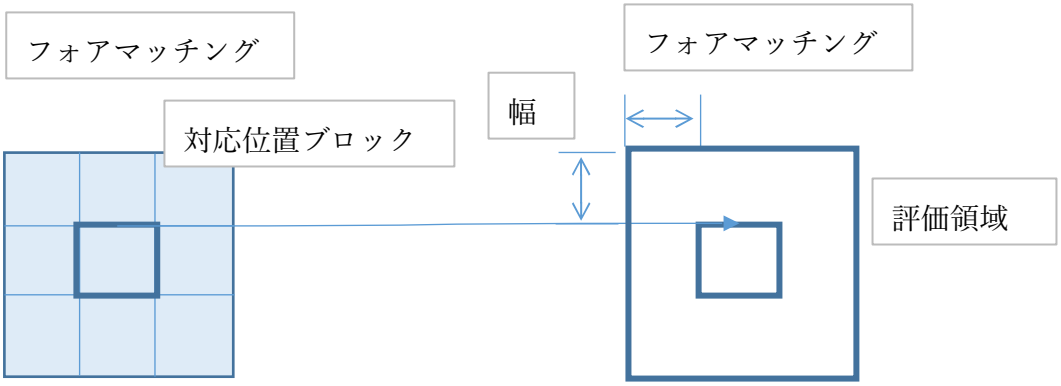
（１）バックマッチングの視差データ上の評価領域サイズ

バック、フォアのそれぞれのマッチングで、すべて正しく視差が求めらていれば、対応する位置のブロックの視差は一致します。

実際には、視差値は一致しない場合があり、視差値が誤っている場合もあります。

フォアマッチング、バックマッチングの対応ブロックを一対一で比較するのではなく、対応位置とその周辺のブロックを使用します。

次の図のように、その領域のサイズをバックマッチング視差評価領域幅で指定します。



（２）同一物体と判断できる視差の範囲

ブロックのサイズよりも大きく見える物体であれば、連続するブロックの視差がほぼ同じになります。

フォアマッチングの視差と、（１）で指定した領域内の複数ブロックの視差がほぼ同等になると仮定できます。

同等と判断できる視差値の幅をバックマッチング評価正当視差値幅で指定します。

フォアマッチングの視差との差が、この幅以下の場合に同等と判断します。

（３）フォアマッチングの正当性評価

フォアマッチングの視差が ”視差なし”（視差値ゼロ）の場合、そのままとします。バックマッチングで補完しません。

フォアマッチングの視差があるブロックに対しては、以下の評価を行います。

バックマッチング評価視差ゼロ率の評価

（１）で指定した領域内の”視差なし”（視差値ゼロ）ブロックの比率を求めます。

この比率が、指定されたバックマッチング評価視差ゼロ率以上の場合は、

フォアマッチングの視差を ”視差なし”（視差値ゼロ）とあします。

バックマッチング評価視差正当率の評価

（1）で指定した領域内の正当視差（2. で指定した範囲内）のブロックの比率を求めます。
この比率が、指定されたバックマッチング評価視差正当率未満の場合は、
フォアマッチングの視差を ”視差なし”（視差値ゼロ）にとします。

（4）パラメータ推奨値

パラメータ	VM(480x752)	XC(720x1280)
バックマッチング視差評価領域幅（片側）（画素）	1	1
バックマッチング評価正当視差値幅（画素）	3	3
バックマッチング評価視差正当率（%）	30	30
バックマッチング評価視差ゼロ率（%）	60	60

4．Disparity Filter

4．1　平均化処理

平均化は、単純な平均（平滑化）ではありません。

ノイズを除去した上で、平均を求め、その信頼度をチェックします。

平均化処理について、パラメータの扱いを見ながら解説します。

4．1．1　平均化パラメータについて

入力される視差データに対して、指定された以下のパラメータを使って平均化を行います。

No	内容	範囲
1	平均化ブロック高さ（片側）（ブロック数）	0 以上、8 以下の整数（デフォルト：3）
2	平均化ブロック幅（片側）（ブロック数）	0 以上、8 以下の整数（デフォルト：3）
3	平均化移動積分幅（片側）（視差値）	0.0 以上の実数（デフォルト：1）
4	平均化分布範囲最大幅（片側）（視差値）	0.0 以上の実数（デフォルト：2）
5	平均化視差含有率（%）	0 以上、100 以下の整数（デフォルト：20）
6	平均化有効比率（%）	0 以上、100 以下の整数（デフォルト：20）
7	平均化置換有効比率（%）	0 以上、100 以下の整数（デフォルト：50）
8	平均化ブロックの重み（中心）	1 以上の整数（デフォルト：1）
9	平均化ブロックの重み（近傍）	1 以上の整数（デフォルト：1）
10	平均化ブロックの重み（周辺）	1 以上の整数（デフォルト：1）

平均化は視差ブロックの単位で行います。画素単位ではありません。

画像の端の視差ブロックは平均化できないため、除外します。

４．１．２　平均化の流れについて

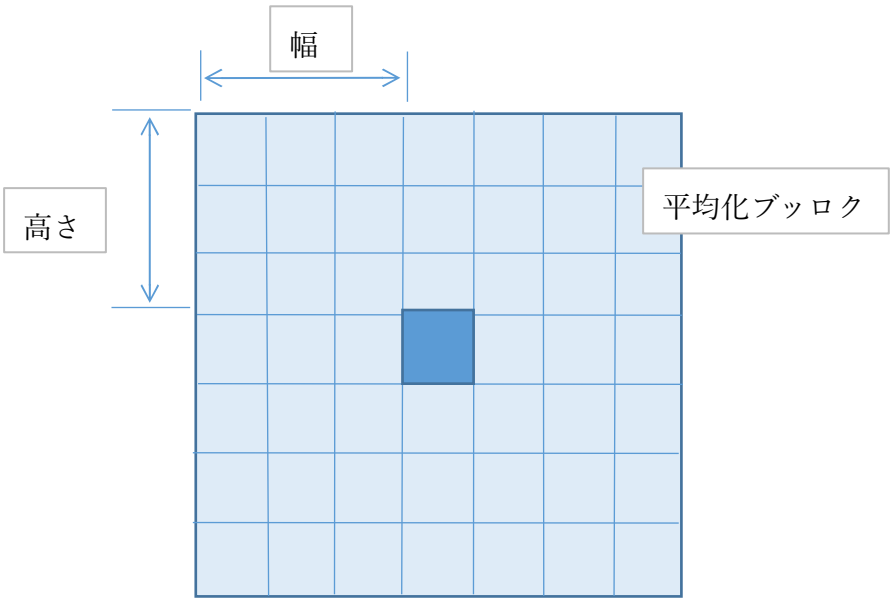
平均化処理へは視差データを入力します。
具体的には、ブロック単位の視差値、ブロックサイズ、マッチング探索幅です。
視差データは、カメラのフレームデータをデコードして取り出すか、ステレオマッチングライブラリから受け取ります。
平均化の流れは以下となります。

①平均化する視差ブロックを決めます
②そのブロックを中心として、平均化のブロック領域を決めます
③領域内のブロックに重みを付けます。
④ブロックの視差値のヒストグラムを作成します
⑤ヒストグラムの中央値を見つけます
⑥中央値を中心に、有効な範囲に入っているブロックを使って平均を求めます
⑦平均の対象となったブロックの割合から信頼度をチェックします
⑧信頼できれば、着目ブロックの視差を平均に置き換え、そうでなければ、”視差なし”（視差値ゼロ）とします

４．１．３　パラメータの扱いについて

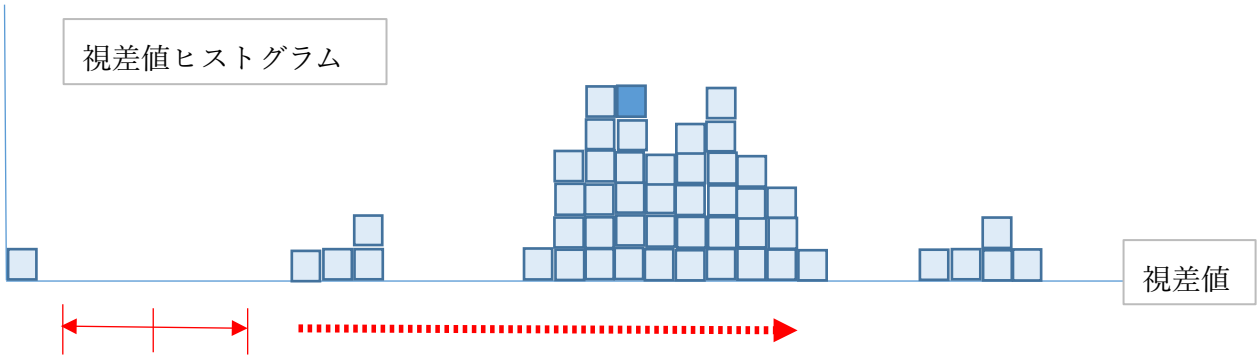
平均化の流れの中で、それぞれのパラメータをどのように扱うか解説するします。

（１）平均化視差ブロックの範囲を決めます
平均化は視差ブロック単位で行います。画素単位ではありません。
平均の視差で置き換えたいブロックを中心に、高さ、幅のブロック数で範囲を指定します。
指定する数×２＋１が、実際のブロック幅になります。
この例では、７×７ブロックの領域が指定されます。
ブロック数が少ないとノイズを除去しきれません。多くすると視差がぼけてしまいます。
ノイズの程度を見て、適切なサイズに決めます。



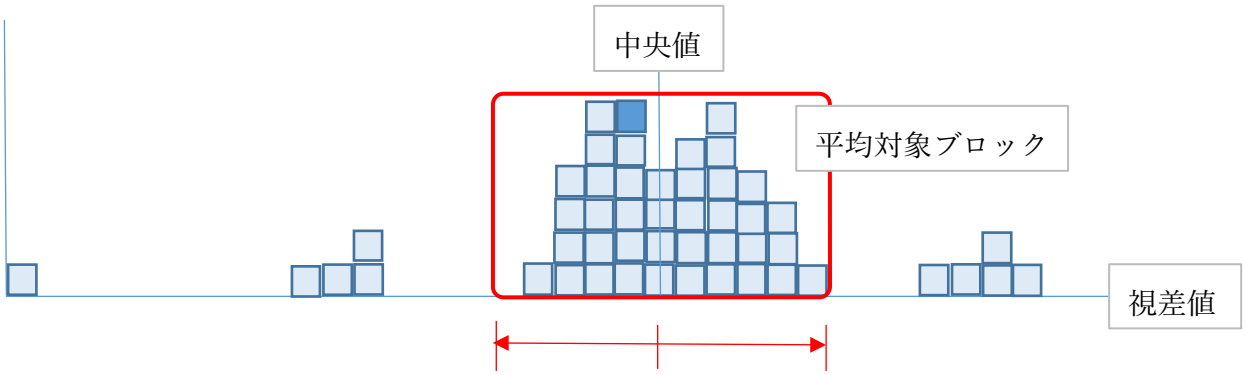
平均化ブロック数

（2）視差のヒストグラムを作成します
領域内のブロック数を度数とする視差のヒストグラムを作成します。
注）ブロックに重みを与えた場合は、重みの分だけ度数を増します。重みについては、あとで説明します。
視差を並べると以下のようなヒストグラムが得られます。
この例のように、単純に最頻値を代表値にできません。
ヒストグラムの下から ”視差なし”（視差値ゼロ）を除き、移動積分（平滑化）を行い、最頻値を検出し、そこを代表値とします。
この代表値を有効な視差の中央値として扱います。



平均化移動積分幅

（3）平均の対象範囲を決めます
中央値からある程度の範囲に入っているブロックを平均の対象とします。
この範囲は、平均化ブロック内では同じような視差になるという前提と、視差値そのもののバラツキから範囲を決めます。
そこから外れたブロックはノイズとし、平均に含めません。



平均化分布範囲最大幅

（4）平均を求めます

範囲内のブロックを使って平均を求める。

次に、この平均の信頼度をチェックする。

（5）信頼度をチェックすします

平均対象ブロックの数が十分になければ、その平均は有効ではなく、信頼できません。

指定された 3 つのパラメータを使って有効かどうか判定します。

A)視差平均化置換有効比率（%）

着目しているブロックが平均対象にはいっていない場合は無効としますが、、平均対象ブロック数が、指定された割合以上であれば有効とします。

着目しているブロックが特異であれば、周辺の視差で置き換えます。

B)視差平均化視差含有率（%）

全体で視差ありブロックの数がこの割合以上であることです。

この割合未満の場合は、無効とします。

視差が乏しい領域では平均を取る意味がありません。

C)視差平均化置換有効比率（%）

視差ありブロックの中で、平均化対象になったブロックがこの割合以上であることです。

この割合未満の場合は、無効とします。

視差がバラバラの領域では、平均が代表しているとはいえません。”視差なし”（視差値ゼロ）にして除去します。

有効な場合は、着目しているブロックの視差値を平均に置き換えます。

無効の場合は、 ”視差なし”（視差値ゼロ）とします。

（6）平均化ブロック領域内のブロックに重みをつけます

平均化ブロックの領域サイズに対し、小さい物体の視差はノイズとして除去されてしまいます。

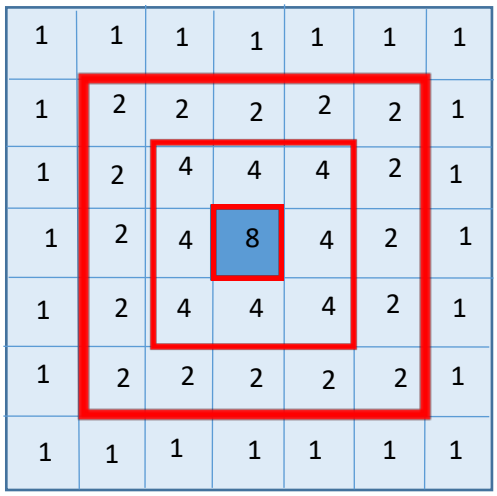
着目しているブロック（領域の中心）とその近傍のブロックの視差は、ほぼ同じになると仮定し、中央の十分な領域を占めていれば、有効な視差として扱えるようにします。

そのために中心からの距離に応じてブロックに重みを付けします。

重みは、視差値のヒストグラムから中央値を求めるとき、平均対象ブロックの平均を求めるときに使用します。

次の例は、中心の重みを 8、その周りの重みを 4、更にその周りの重みを 2 とした場合です。

重みの三つのパラメータ（中心、近傍、周辺）は、それぞれに該当します。



重みを付けた場合、ノイズが残る場合があります。
物体の大きさとの兼ね合いで決めます。

（5）パラメータ推奨値

パラメータ	VM(480x752)	XC(720x1280)
平均化ブロック高さ（片側）（ブロック数）	3	3
平均化ブロック幅（片側）（ブロック数）	3	3
平均化移動積分幅（片側）（視差値）	1	1
平均化分布範囲最大幅（片側）（視差値）	2	2
平均化視差含有率（%）	40	20
平均化有効比率（%）	40	20
平均化置換有効比率（%）	40	40
平均化ブロックの重み（中心）	1	1
平均化ブロックの重み（近傍）	1	1
平均化ブロックの重み（周辺）	1	1

４．２　補完処理

補完は、”視差なし”（視差値ゼロ）となった領域を周辺の視差を使って埋める処理です。
補完は視差ブロック単位で行います。
次のように、”視差なし”（視差値ゼロ）ブロックを、その前後の視差を使って補完します。
補完する視差値を線形近似によって求め、補完する領域を平面にします。

	0	1	2	3	4	5	6	7	8	9
元の視差値	32	31	30	0	0	0	0	25	24	23
補完された視差値				29	28	27	26			

これで、すべて埋めることができます。
埋めたくない領域をパラメータを使って埋めないように制限します。

４．２．１　補完パラメータについて

補完には以下のパラメータを使用します。
それぞれのパラメータを使って、補完を制限します。

No	内容	範囲
1	補完最小視差値（視差値）	0.0 以上の実数（デフォルト：5）
2	補完幅の最大視差勾配	0.0 以上の実数（デフォルト：0.1）
3	補完画素幅の視差値倍率（内側）	0.0 以上の実数（デフォルト：1）
4	補完画素幅の視差値倍率（周辺）	0.0 以上の実数（デフォルト：0.2）
5	補完画素幅の視差値倍率（下端）	0.0 以上の実数（デフォルト：0.1）
6	補完ブロックのコントラスト上限値	0 以上の整数（デフォルト：40）
7	穴埋め幅（画素数）	0.0 以上の実数（デフォルト：8）

４．２．２　補完の流れについて

補完は、平均化された視差の入力を前提としています。
補完は、縦方向、横方向、斜め方向に走査して行います。
補完は、最終処理の穴埋めで、埋め潰しを行うが、これを省くことができます。
また、穴埋め処理だけを選択することもできます。

４．２．３　パラメータの扱いについて

そのぞれのパラメータの詳細を説明します。

（１）補完最小視差値

空などの視差のない背景のブロックは、そのままにしておくようにしたいため、前後の視差値が指定された値未満の場合は、補完しないようにします。

視差値が小さい遠方のブロックを使った補完を制限します。

例えば、5 以上を補完、5 未満を補完しないとした場合、以下のケースは補完されずに”視差なし”のままになります。

	0	1	2	3	4	5	6	7	8	9
元の視差値	0	0	4	0	0	0	0	25	24	23

（２）補完幅の最大視差勾配

狭い画素幅の中で視差値が大きく変化している場合、補完がふさわしくない場合があります。

連続する同一平面でない可能性が高い状態です。オクリュージョンの領域などです。

前後の視差の差が大きい場合、補完を制限します。

この視差の差を、次の例のように視差勾配で評価します。

次の例の視差勾配は、

視差値	80	0	0	10

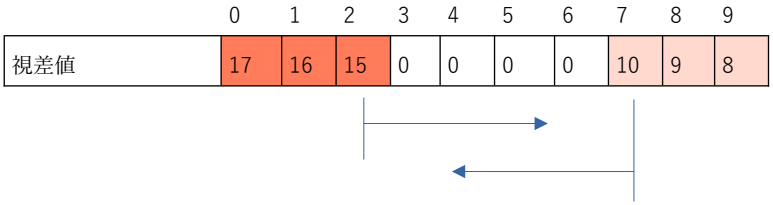
視差勾配 : 5.83 = (80 - 10) / (3 x 4)

＊ ブロックの画素幅を 4 とする

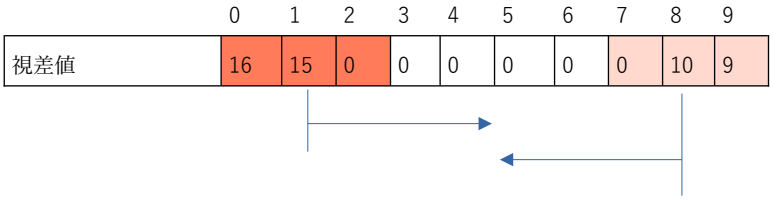
視差勾配を計算して、指定された値以上の場合は補完しないようにします。

（3）補完画素幅の視差値倍率（内側）

広い範囲に亘って”視差なし”ブロックが続く場合、補完がふさわしくない場合があります。
同一平面として見なせない場合があります。
補完幅の制限に、次の例のように前後の視差値を使います。
次の例は、前後の視差値が 15 と 10 で、補完する場合です。



視差値は、その距離の基線長の幅の画素幅を表しています。
基線長が 10cm の場合、視差値が 15 であれば、その距離に置いた 10cm 幅の物体は、画素幅 15 で見えています。
視差値が 10 であれば、10cm 幅は画素幅 10 になります。
ここで、15 + 10 = 25 画素は、およそ基線長の 2 倍の 20cm に相当します。
一方、補完対象は、5 ブロック幅で、ブロックの画素幅を 4 とすると、20 画素分です。
画総数で 25 ≧ 20 なので、幅が 20cm より狭いと判断して、補完を行います。
次のように、補完対象が 7 ブロック幅に広がると、28 画素分になって、25 を超えてしまいます。
幅が 20cm を超えたと判断して、補完しないようにします。



このことから、補完幅を、次の式のように倍率を与えて調整することができます。

(前視差 + 後視差値) × 倍率 ≧ ”視差なし”ブロックの画素幅

倍率は、補完したい物体の幅と基線長によって決めます。

（4）補完画素幅の視差値倍率（周辺）

（5）補完画素幅の視差値倍率（下端）

画面の端においては、前後に視差がない場合があります。

視差なしブロックから始まる、または、視差なブロックで終わるケースです。

この場合は、一方だけの視差値を使って、同様に倍率で補完幅を制限します。

視差値×倍率 ≧ ”視差なし"ブロックの画素幅

倍率は、下端と周辺（上端、左端、右端）に分けて指定します。

（6）補完ブロックのコントラスト上限値

補完はパターンが弱いために"視差なし"になった視差ブロックに対してのみ行います。

本来、パターンが強ければ”視差なし”とはならないため、補完すべきではありません。

パターンの強度がこの上限値を超えるブロックは補完しません。

パターンの強度評価に次のコントラスト C を使用します。

C = (Lmax - Lmin - Lofs) / Lave

Lmax：マッチングブロック内の輝度の最大値

Lmin：マッチングブロック内の輝度の最小値

Lave：マッチングブロック内の輝度の平均値

Lofs：C が明るさに依存しないようにするオフセット

＊Lofs は、解像度によって異なる 。480x752 では 1.8、720x1280 では 1.2 固定

閾値の指定には、C を 1000 倍した値を指定します。

（7）穴埋め幅

単純な穴埋めを行うこともできます。

連続する”視差なし”の画素数が指定された幅より狭い場合に補完します。

ただし、視差勾配の制限は行います。

（8）パラメータ推奨値

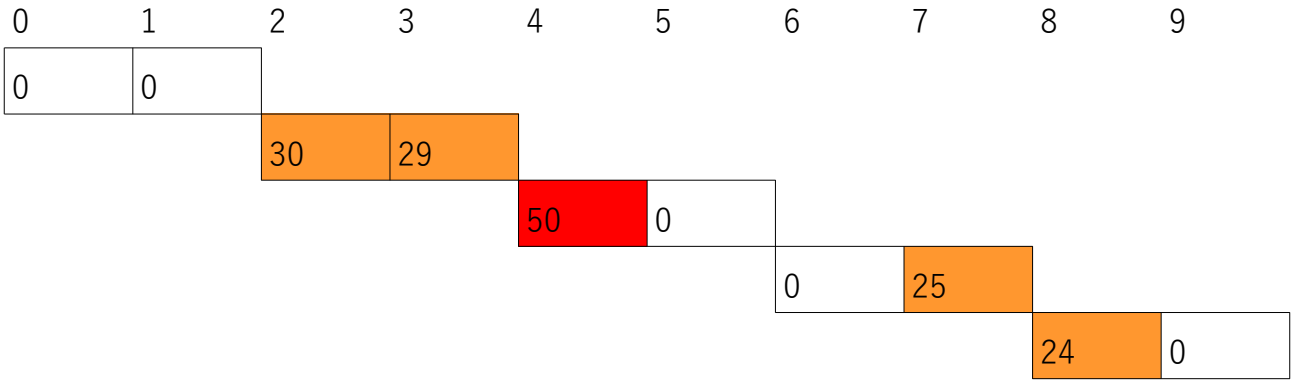
パラメータ	VM(480x752)	XC(720x1280)
補完最小視差値（視差値）	5	5
補完幅の最小視差勾配	0.1	0.1
補完画素幅の視差値倍率（内側）	1	1
補完画素幅の視差値倍率（周辺）	0.2	0.2
補完画素幅の視差値倍率（下端）	0.1	0.1
補完ブロックのコントラスト上限値	40	40
穴埋め幅（画素数）	8	8

４．３　エッジ補間処理

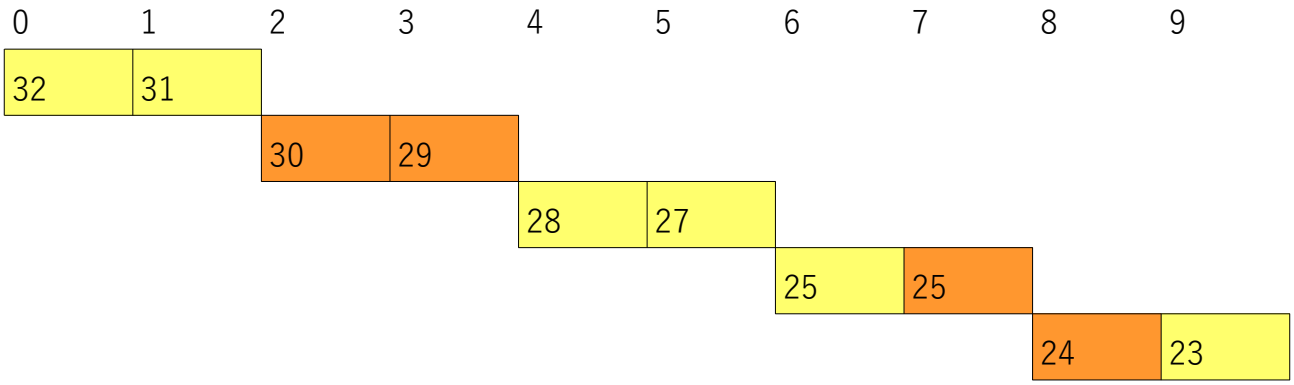
４．３．１　補間処理

画像上のエッジ部分はパターン強度は高く、そのためマッチングに成功し視差を検出しやすくなることが想定されます。
しかしながら、物体の輪郭であっても、背景のパターンなどにより正しい視差を検出できない場合があります。
そこで、直線に見えるエッジを同一物体の輪郭とし、その直線の上の有効な視差を使い、誤った視差を除去、補完します。
この補完処理は、下図のようなイメージとなります。
元の視差値では、画像の上の斜め直線エッジの位置に検出された視差の様子を示しています。
”視差なし”（視差値ゼロ）や、誤った視差が検出されています。
その部分を、有効な視差を使って補完すると、補完された視差値のようになります。

元の視差値



補間された視差値



４．３．２　エッジ補間パラメータ

エッジ補完には以下のパラメータを使用します。

（１）エッジ線分上の最小視差ブロック数　0 以上の整数（デフォルト：20）

（２）エッジ視差の最小線形性指数（回帰線の決定係数）　0 以上、100 以下の実数（デフォルト：20.0）

（３）エッジ線の補完視差ブロック幅（ブロック数）　1 以上の整数（デフォルト：1）

直線エッジを検出するパラメータは以下です。

直線エッジの検出には OpenCV を使用するため、そのために必要なパラメータです。

エッジ検出：Canny

（１）閾値１　整数（デフォルト：50）

（２）閾値２　整数（デフォルト：100）

＊閾値２より大きいものをエッジと判断し、かつ閾値１より大きければ繋がっていると判断します。

直線検出（ハフ変換）：HoughLinesP

（１）投票閾値　整数（デフォルト：100）

（２）最小線分長　整数（デフォルト：80）

（３）最大ギャップ長　整数（デフォルト：5）

＊最小線分長より短い線分は棄却し、最大ギャップ長より離れると同一線分としません。

４．３．３　エッジ補完の流れ

（１）補正画像から直線を検出します。

エッジの検出には、OpenCV の Canny を使用します。

直線線分の検出に、OpenCV の HoughLinesP を使用します。

（２）水平な直線を除きます。

＊水平な線は、ステレオマッチングに失敗します。

（３）直線上の視差を取得します。

（４）外れている視差を除き、”視差なし”（視差値ゼロ）とします。

最頻値を求め、最頻値から±４分の１を超えた視差を外れとします。

（５）残った視差（有効な視差）から回帰直線と、その線形性指数（決定係数）を求めます。

（６）有効な視差が少なく、線形性指数が低い場合は、補完しません。

（７）”視差なし”（視差値ゼロ）を補完します。

”視差なし”区間の両端に有効な視差があれば、その値を使い線形補完します。

”視差なし”区間が線上の端にある場合は、回帰直線の傾きを使い線形補完します。

（8）補完の対象は、基本的にエッジ直上の視差ブロックのみとします。
（9）補完視差ブロック幅の指定により、直線上の視差ブロックと前後のブッロクを補完します。
　　＊エッジ線を太くして、強調することができます。

パラメータ推奨値

パラメータ	VM(480x752)	XC(720x1280)
エッジ線分上の最小視差ブロック数	20	20
エッジ視差の最小線形性指数（回帰線の決定係数）	20	20
エッジ線の補完視差ブロック幅	1	1
Canny エッジ検出閾値 1	50	50
Canny エッジ検出閾値 2	100	100
HoughLinesP 投票閾値	100	100
HoughLinesP 最小線分長	80	80
HoughLinesP 最大ギャップ長	5	5

5．Frame DecoderFunctions

一覧

関数	概要
setFrameDecoderParameter()	フレームデコーダのパラメータを設定する
setDoubleShutterOutput()	ダブルシャッター出力を設定する
decodeFrameData()	フレームデータを画像データまたは視差エンコードデータに分割する
getDisparityData()	視差データをデコードして視差画像と視差情報に戻し、平均化、補完処理を行う
getDoubleDisparityData()	ダブルシャッターの視差エンコードデータをデコードして、視差の平均化、補完処理を行う
initialize()	フレームデコーダを初期化する
finalize()	フレームデコーダを終了する
setDisparityLimitation()	視差の下限値、上限値を設定する

setFrameDecoderParameter()

フレームデコーダのパラメータを設定する

static void ISCFrameDecoder::setFrameDecoderParameter(
　　int crstthr,
　　int crsthrm);

引数

crstthr	コントラスト閾値(IN)
crsthrm	センサー輝度高解像度モードステータス 0:オフ 1:オン(IN)

補足説明

コントラスト閾値 crstthr は、コントラスト C を 1000 倍した値で指定する。

センサーの輝度高解像度モードが使用されている場合は、モード crsthrm を 1 にする。

setDoubleShutterOutput()

ダブルシャッター出力を設定する

static void ISCFrameDecoder::setDoubleShutterOutput(
　　int dbdout,
　　int dbcout);

引数

dbdout	ダブルシャッター出力 0:ブレンド 1:高感度 2:低感度(IN)
dbcout	ダブルシャッター補正画像出力 0:ブレンド 1:高感度 2:低感度 3:適当(IN)

補足説明

ダブルシャッター出力 dbdout に 0:ブレンドを指定したとき、ダブルシャッター補正画像出力 dbcout を指定できる。

decodeFrameData()

フレームデータを画像データまたは視差エンコードデータに分割する

```
static void ISCFrameDecoder::decodeFrameData(
        int imghgt,
        int imgwdt,
         unsigned char* pfrmdat,
        unsigned char* prgtimg,
        unsigned char* plftimg);
```

引数

imghgt	画像の高さ(IN)
imgwdt	画像の幅(IN)
pfrmdat	フレームデータ(IN)
prgtimg	右（基準）画像データ 右下原点(OUT)
plftimg	左（比較）画像データまたは視差エンコードデータ 右下原点(OUT)

補足説明

補正画像モードのフレームデータでは、左右の画像データをステレオマッチングの入力と使用することができる。

視差画像モードのフレームデータでは、視差データは出力されない。視差エンコードデータ plftimg が出力される。

関数 getDisparityData()または getDoubleDisparityData()を呼び出して、視差エンコードデータをデコードすること

getDisparityData()

視差データをデコードして視差画像と視差情報に戻し、平均化、補完処理を行う

```
static void getDisparityData(
        int imghgt,
        int imgwdt,
        unsigned char* prgtimg,
        unsigned char* pdspenc,
        int* pblkhgt,
        int* pblkwdt,
        int* pmtchgt,
        int* pmtcwdt,
        int* pblkofsx,
        int* pblkofsy,
        int* pdepth,
        int* pshdwdt,
        unsigned char* pdspimg,
        float* ppxldsp,
        float* pblkdsp,
        int *pblkval,
        int *pblkcrst);
```

引数

imghgt	画像の高さ
imgwdt	画像の幅
prgtimg	右（基準）画像データ 右下原点
pdspenc	視差エンコードデータ
pblkhgt	視差ブロック高さ
pblkwdt	視差ブロック幅
pmtchgt	マッチングブロック高さ
pmtcwdt	マッチングブロック幅
pblkofsx	視差ブロック横オフセット
pblkofsy	視差ブロック縦オフセット
pdepth	マッチング探索幅
pshdwdt	画像遮蔽幅
pdspimg	視差画像 右下原点

ppxldsp　　視差データ 右下原点
pblkdsp　　ブロック視差データ 右下基点
pblkval　　視差ブロック視差値(1000 倍サブピクセル精度整数)
pblkcrst　　ブロックコントラスト

補足説明

平均化処理、補完処理がオンの場合は、処理された視差データが出力される

視差画像データ ppxldsp は表示用のデータである。視差値を 256 階調（0 から 255 の整数）に正規化している。

出力先の視差画像データ pdspimg は、入力の視差エンコードデータ pdspenc と同じ領域でもよい。視差画像は、エンコードデータ領域に展開される。

getDoubleDisparityData()

ダブルシャッターの視差エンコードデータをデコードして、視差の平均化、補完処理を行う

```
static void getDoubleDisparityData(
        int imghgt,
        int imgwdt,
        unsigned char* pimgcur,
        unsigned char* penccur,
        int expcur,
        int gaincur,
        unsigned char* pimgprev,
        unsigned char* pencprev,
        int expprev,
        int gainprev,
        int* pblkhgt,
        int* pblkwdt,
        int* pmtchgt,
        int* pmtcwdt,
        int* pblkofsx,
        int* pblkofsy,
        int* pdepth,
        int* pshdwdt,
        unsigned char* pbldimg,
        unsigned char* pdspimg,
        float* ppxldsp,
        float* pblkdsp,
        int* pblkval,
        int* pblkcrst);
```

引数

imghgt	画像の高さ
imgwdt	画像の幅
pimgcur	現フレーム画像データ
penccur	現フレーム視差エンコードデータ
pexpcur	現フレームシャッター露光値
pgaincur	現フレームシャッターゲイン値

pimgprev	前フレーム画像データ
pencprev	前フレーム視差エンコードデータ
pexpprev	前フレームシャッター露光値
pgainprev	前フレームシャッターゲイン値
pblkhgt	視差ブロック高さ
pblkwdt	視差ブロック幅
pmtchgt	マッチングブロック高さ
pmtcwdt	マッチングブロック幅
pblkofsx	視差ブロック横オフセット
pblkofsy	視差ブロック縦オフセット
pdepth	マッチング探索幅
pshdwdt	画像遮蔽幅
pbldimg	合成画像 右下基点
pdspimg	視差画像 右下基点
ppxldsp	視差情報 右下基点
pblkdsp	ブロック視差情報 右下基点
pblkval	ブロック視差値(1000 倍サブピクセル精度の整数)
pblkcrst	ブロックコントラスト

補足説明

平均化処理、補完処理がオンの場合は、処理された視差データが出力される。

視差画像データ ppxldsp は表示用のデータである。視差値を 256 階調（0 から 255 の整数）に正規化している。

initialize()

フレームデコーダを初期化する

static void ISCFrameDecoder::initialize(
　　int imghgt,
　　int imgwdt);

引数

　　imghgt　　画像の高さ(IN)
　　imgwdt　　画像の幅(IN)

補足説明

　　指定されたサイズで処理バッファが確保される。
　　終了時に、必ず 関数 finalize()を呼び出す

finalize()

フレームデコーダを終了する

static void ISCFrameDecoder::finalize();

setDisparityLimitation()

視差の下限値、上限値を設定する

void ISCFrameDecoder::setDisparityLimitation(
　　int limit,
　　double lower,
　　double upper);

引数

　　limit　視差値の制限 0:しない 1:する(IN)
　　lower 視差値の下限(IN)
　　upper 視差値の上限(IN)

6．Soft Matching Functions

一覧

関数	概要
setMatchingParameter()	ブロックマッチングパラメータを設定する
setBackMatchingParameter()	バックマッチングパラメータを設定する
matching()	ステレオマッチングを実行する
getDisparity()	視差画素情報を取得する
getBlockDisparity()	視差ブロック情報を取得する
setUseOpenCLForMatching()	ステレオマッチングに OpenCL の使用を設定する
createMatchingThread()	ステレオマッチングスレッドを生成する
deleteMatchingThread()	マッチングスレッドを破棄する
initialize()	ステレオマッチングを初期化する
finalize()	ステレオマッチングを終了する

setMatchingParameter()

ブロックマッチングパラメータを設定する

```
static void StereoMatching::setMatchingParameter(
        int imghgt,
        int imgwdt,
        int depth,
        int blkhgt,
        int blkwdt,
        int mtchgt,
        int mtcwdt,
        int blkofsx,
        int blkofsy,
        int crstthr,
        int crsthrm);
```

引数

imghgt	補正画像の高さ(IN)
imgwdt	補正画像の幅(IN)
depth	マッチング探索幅(IN)
stphgt	視差ブロック高さ(IN)
stpwdt	視差ブロック幅(IN)
blkhgt	マッチングブロック高さ(IN)
blkwdt	マッチングブロック幅(IN)
blkofsx	視差ブロック横オフセット(IN)
blkofsy	視差ブロック縦オフセット(IN)
crstthr	コントラスト閾値（コントラスト C x 1000）
crsthrm	センサー輝度高解像度モードステータス 0:オフ 1:オン(IN)

補足説明

視差ブロックのサイズは、マッチングブロック以下にすること

コントラスト閾値 crstthr は、コントラスト C を 1000 倍した値で指定する。

センサーの輝度高解像度モードが使用されている場合は、モード crsthrm を 1 にする。

setBackMatchingParameter()

バックマッチングパラメータを設定する

```
static void StereoMatching::setBackMatchingParameter(
        int enb,
        int bkevlwdt,
        int bkevlrng,
        int bkvldrt,
        int bkzrrt
);
```

引数

- enb　　バックマッチング 0:しない 1:する(IN)
- bkevlwdt　バックマッチング視差評価領域幅（片側）(IN)
- bkevlrng　バックマッチング視差評価視差値幅(IN)
- bkvldrt　　バックマッチング評価視差正当率（%）
- bkzrrt　　バックマッチング評価視差ゼロ率（%）(IN)

matching()

ステレオマッチングを実行する

```
static void StereoMatching::matching(
        unsigned char* prgtimg,
        unsigned char* plftimg
);
```

引数

・prgtimg　右（基準）補正画像データ(IN)

・plftimg　左（比較）補正画像データ(IN)

補足説明

画像データは、右下を原点として、右下から左上へ向かって格納されていること

getDisparity()

視差画素情報を取得する

```
void StereoMatching::getDisparity(
        int hight,
        int width,
        unsigned char *pdspimg,
        float *ppxldsp
)
```

引数

- imghgt　視差画像を格納するバッファの高さ(IN)
- imgwdt　視差画像を格納するバッファの幅(IN)
- pdspimg　視差画像データを格納するバッファのポインタ(OUT)
- ppxldsp　視差値データを格納するバッファのポインタ(OUT)

補足説明

視差画像データ pdspimg、視差値データ ppxldsp は、右下を原点として、右下から左上へ向かって格納される。

視差画像データ pdspimg は、表示用のデータである。視差値を 256 階調（0 から 255 の整数）に正規化している。

視差の最大値は、関数 setMatchingParameter()で指定されたマッチング探索幅 - 1 である。

getBlockDisparity()

視差ブロック情報を取得する

```
static void StereoMatching::getBlockDisparity(
        int *pblkhgt,
        int *pblkwdt,
        int *pmtchgt,
        int *pmtcwdt,
        int *pblkofsx,
        int *pblkofsy,
        int *pdepth,
        int *pshdwdt,
        float *pblkdsp,
        int *pblkval,
        int *pblkcrst
);
```

引数

- pblkhgt　視差ブロック高さ(OUT)
- pblkwdt　視差ブロック幅(OUT)
- pmtchgt　マッチングブロック高さ(OUT)
- pmtcwdt　マッチングブロック幅(OUT)
- pblkofsx　視差ブロック横オフセット(OUT)
- pblkofsy　視差ブロック縦オフセット(OUT)
- pdepth　マッチング探索幅(OUT)
- pshdwdt　画像遮蔽幅(OUT)
- pblkdsp　視差ブロック視差値(OUT)
- pblkval　視差ブロック視差値(1000 倍サブピクセル精度の整数)(OUT)
- pblkcrst　マッチングブロックコントラスト(OUT)

補足説明

視差ブロック視差値 pblkdsp、マッチングブロックコントラスト pblkcrst を格納するサイズは、次の高さ×幅分が必要である。

高さ：補正画像の高さ÷視差ブロック高さ　（小数点以下切り捨て）

幅：補正画像の幅÷視差ブロック幅　（小数点以下切り捨て）

視差ブロック視差値、マッチングブロックコントラストは、右下から左上へ向かって格納される。

画像遮蔽幅には、補正画像の幅に対して、視差が出力されない左端領域の幅が返る。

画像遮蔽幅は、バックマッチングを使用しない場合はマッチング探索幅になり、使用する場合は 0（ゼロ）になる。

コントラストは、コントラスト C を 1000 倍した値である。

setUseOpenCLForMatching()

ステレオマッチングマッチングに OpenCL の使用を設定する

static void StereoMatching::StereoMatching::setUseOpenCLForMatching(
　　int usecl
);

引数

　　・usecl　　OpenCL を使用 0:しない 1:する(IN)

補足説明

　　OpenCL を使用しない場合は、関数 createMatchingThread()を呼び出して、マッチング処理を行うスレッドを生成すること

createMatchingThread()

ステレオマッチングマッチングスレッドを生成する

static void StereoMatching::createMatchingThread();

deleteMatchingThread()

マッチングスレッドを破棄する

static void StereoMatching::deleteMatchingThread();

initialize()

ステレオマッチングマッチングを初期化する

static void StereoMatching::initialize(
　　int imghgt,
　　int imgwdt
);

引数

imghgt	補正画像の高さ(IN)
imgwdt	補正画像の幅(IN)

補足説

　　指定されたサイズで処理バッファが確保される。
　　終了時に、必ず 関数 finalize()を呼び出すこと。

finalize()

ステレオマッチングマッチングを終了する

static void StereoMatching::finalize();

7．Disparity Filter Functions

カメラから出力されるフレームデータを SDK 経由で取得し、画像データへ変換する。
視差の平均化、補完処理を行う。

一覧

関数	概要
setAveragingParameter()	視差平均化パラメータを設定する
setAveragingBlockWeight()	視差平均化ブロックの重みを設定する
setComplementParameter()	視差補完パラメータを設定する
setEdgeComplementParameter()	エッジ補完パラメータを設定する
setHoughTransformParameter()	ハフ変換パラメータを設定する
averageDisparityData()	視差を平均化する
createAveragingThread()	視差平均化スレッドを生成する
deleteAveragingThread()	視差平均化スレッドを破棄する
setUseOpenCLForAveragingDisparity()	視差平均化処理に OpenCL の使用を設定する
initialize()	フレームデコーダを初期化する
finalize()	フレームデコーダを終了する
setDisparityLimitation()	視差の下限値、上限値を設定する

setAveragingParameter()

視差平均化パラメータを設定する

```
static void DisparityFilter::setAveragingParameter(
        int enb,
        int blkshgt,
        int blkswdt,
        double intg,
        double range,
        int dsprt,
        int vldrt,
        int reprt);
```

引数

enb　平均化処理 しない：0 する：1(IN)

blkshgt　　平均化ブロック高さ（片側）(IN)

blkswdt　　平均化ブロック幅（片側）(IN)

intg　平均化移動積分幅（片側）(IN)

range 平均化分布範囲最大幅（片側）(IN)

dsprt 平均化視差含有率(IN)

vldrt　平均化有効比率(IN)

reprt　平均化置換有効比率(IN)

setAveragingBlockWeight()

視差平均化ブロックの重みを設定する

```
static void DisparityFilter::setAveragingBlockWeight(
        int cntwgt,
        int nrwgt,
        int rndwgt);
```

引数

cntwgt　　ブロックの重み（中央）(IN)
nrwgt ロックの重み（近傍）(IN)
rndwgt　　ブロックの重み（周辺）(IN)

setComplementParameter()

視差補完パラメータを設定する

```
static void DisparityFilter::setComplementParameter(
        int enb,
        double lowlmt,
        double slplmt,
        double insrt,
        double rndrt,
        double btmrt,
        int crstlmt,
        int hlfil,
        double hlsz);
```

引数

enb　補完処理しない：0 する：1(IN)
lowlmt　　補完最小視差値(IN)
slplmt 補完幅の最大視差勾配(IN)
insrt　補完画素幅の視差値倍率（内側）(IN)
rndrt　補完画素幅の視差値倍率（周辺）(IN)
btmrt 補完画素幅の視差値倍率（下端）(IN)
crstlmt　　補完ブロックのコントラスト上限値(IN)
hlfil　穴埋め処理しない：0 する：1 (IN)
hlsz　穴埋め幅 (IN)

補足説明

補完ブロックのコントラスト上限値 crstlmt は、コントラスト C を 1000 倍した値で指定する。

setEdgeComplementParameter()

エッジ補完パラメータを設定する

static void DisparityFilter::setEdgeComplementParameter(
int edgcmp,
int minblks,
double mincoef,
int cmpwdt);

引数

edgcmp	エッジ補完 0:しない 1:する(IN)
minblks	エッジ線分上の最小視差ブロック数(IN)
mincoef	エッジ視差の最小線形性指数（回帰線の決定係数）(IN)
cmpwdt	エッジ線の補完視差ブロック幅(IN)

setHoughTransformParameter()

ハフ変換パラメータを設定する

static void DisparityFilter::setHoughTransformParameter(
int edgthr1,
int edgthr2,
int linthr,
int minlen,
int maxgap)

引数

edgthr1	Canny エッジ検出閾値 1(IN)
edgthr2	Canny エッジ検出閾値 2(IN)
linthr	HoughLinesP 投票閾値(IN)
minlen	HoughLinesP 最小線分長(IN)
maxgap	HoughLinesP 最大ギャップ長(IN)

補足説明

エッジ視差の補完では、エッジの検出に OpenCV が提供する Canny エッジ検出とハフ変換：HoughLinesP を使用している。

この関数で、それぞれに指定するパラメータを設定する。

averageDisparityData()

視差を平均化する

static bool DisparityFilter::averageDisparityData(
int imghgt,
int imgwdt,
unsigned char* prgtimg,
int blkhgt,
int blkwdt,
int mtchgt,
int mtcwdt,
int dspofsx,
int dspofsy,
int depth,
int shdwdt,
int *pblkval,
int *pblkcrst,
unsigned char* pdspimg,
float* ppxldsp,
float* pblkdsp

引数

imghgt　画像の高さ(IN)
imgwdt　画像の幅(IN)
prgtimg　右（基準）画像データ 右下原点(IN)
blkhgt視差ブロックの高さ(IN)
blkwdt　視差ブロックの幅(IN)
mtchgt　マッチングブロックの高さ(IN)
mtchgt　マッチングブロックの幅(IN)
blkofsx　視差ブロック横オフセット(IN)
blkofsy　視差ブロック縦オフセット(IN)
depth マッチング探索幅(IN)
shdwdt　遮蔽領域幅(IN)
pblkval　視差ブロック視差値(1000 倍サブピクセル精度整数)(IN)
pblkcrst　ブロックコントラスト(IN)

pdspimg　　視差画像データ 右下原点(OUT)
ppxldsp　　視差値データ 右下原点(OUT)
pblkdsp　　ブロック視差値データ 右下基点(OUT)

戻り値

処理結果を返す

補足説明

ステレオマッチングが出力する視差データを、平均化処理、補完処理する場合に、この関数を呼び出す。

平均化処理がオフかつエッジ補完がオフの場合は、false を返す。

平均化処理、補完処理がオンの場合は、処理された視差データが出力される。

視差値データ ppxldsp は画素単位、ブロック視差値データ ppxldsp はブロック単位である。

視差画像データ pdspimg は表示用のデータである。視差値を 256 階調（0 から 255 の整数）に正規化している。

createAveragingThread()

視差平均化スレッドを生成する

static void DisparityFilter::createAveragingThread();

deleteAveragingThread()

視差平均化スレッドを破棄する

static void DisparityFilter::deleteAveragingThread();

setUseOpenCLForAveragingDisparity()

視差平均化処理に OpenCL の使用を設定する

static void DisparityFilter::setUseOpenCLForAveragingDisparity(int usecl);

引数

usecl OpenCL を使用 0:しない 1:する(IN)

補足説明

OpenCL を使用しない場合は、関数 createAveragingThread()を呼び出して、マッチング処理を行うスレッドを生成すること

initialize()

視差フィルターを初期化する

static void DisparityFilter::initialize(int imghgt, int imgwdt);

引数

imghgt　画像の高さ(IN)

imgwdt　画像の幅(IN)

補足説明

指定されたサイズで処理バッファが確保される。

終了時に、必ず 関数 finalize()を呼び出すこと

finalize()

視差フィルターを終了する

static void DisparityFilter::finalize();

setDisparityLimitation()

視差の下限値、上限値を設定する

static void DisparityFilter::setDisparityLimitation(int limit, double lower, double upper);

引数

limit　視差値の制限 0:しない 1:する(IN)

lower 視差値の下限(IN)

upper 視差値の上限(IN)

改版履歴

Rev	Date	Content
0.0.1	2023/4/30	初版発行
0.0.2	2023/7/31	ライブラリ 2.0 対応 BlockMatching ->StereoMatching に移行 Frame Decoder より Filter 機能を Disparity Filter として分離

End of Document