1.기본 구조체(Basic Structures)

CvPoint

정수 좌표계에 의한다. 2 차원의 점

```
typedef struct CvPoint
{
    int x; /* x 좌표.통상은 0 하지만 원점 */
    int y; /* y 좌표.통상은 0 하지만 원점 */
}
CvPoint;

/* constructor      */
inline CvPoint cvPoint( int x, int y );

/* CvPoint2D32f (으)로부터의 변환 */
inline CvPoint cvPointFrom32f( CvPoint2D32f point );
```

CvPoint2D32f

부동 소수점형(단정도) 좌표계에 의한다 2 차원의 점

```
typedef struct CvPoint2D32f
{
    float x; /* x 좌표.통상은 0 하지만 원점 */
    float y; /* y 좌표.통상은 0 하지만 원점 */
}
CvPoint2D32f;

/* constructor      */
inline CvPoint2D32f cvPoint2D32f( double x, double y );

/* CvPoint (으)로부터의 변환 */
inline CvPoint2D32f cvPointTo32f( CvPoint point );
```

CvPoint3D32f

부동 소수점형(단정도) 좌표계에 의한다 3 차원의 점

```
typedef struct CvPoint3D32f
{
    float x; /* x 좌표.통상은 0 하지만 원점 */
    float y; /* y 좌표.통상은 0 하지만 원점 */
```

```
    float z; /* z 좌표.통상은 0 하지만 원점 */
}
CvPoint3D32f;

/* constructor      */
inline CvPoint3D32f cvPoint3D32f( double x, double y, double z );
```

CvPoint2D64f

부동 소수점형(배정도) 좌표계에 의한다 2 차원의 점

```
typedef struct CvPoint2D64f
{
    double x; /* x 좌표.통상은 0 하지만 원점 */
    double y; /* y 좌표.통상은 0 하지만 원점 */
}
CvPoint2D64f;

/* constructor      */
inline CvPoint2D64f cvPoint2D64f( double x, double y );

/* CvPoint (으)로부터의 변환 */
inline CvPoint2D64f cvPointTo64f( CvPoint point );
```

CvPoint3D64f

부동 소수점형(배정도) 좌표계에 의한다 3 차원의 점

```
typedef struct CvPoint3D64f
{
    double x; /* x 좌표.통상은 0 하지만 원점 */
    double y; /* y 좌표.통상은 0 하지만 원점 */
    double z; /* z 좌표.통상은 0 하지만 원점 */
}
CvPoint3D64f;

/* constructor      */
inline CvPoint3D64f cvPoint3D64f( double x, double y, double z );
```

CvSize

구형의 픽셀 정도로의 사이즈

```
typedef struct CvSize
{
    int width; /* 구형의 폭 */
```

```
    int height; /* 구형의 높이 */
}
CvSize;

/* constructor      */
inline CvSize cvSize( int width, int height );
```

CvSize2D32f

구형의 사브픽셀 정도로의 사이즈

```
typedef struct CvSize2D32f
{
    float width; /* 구형의 폭 */
    float height; /* 구형의 높이 */
}
CvSize2D32f;

/* constructor      */
inline CvSize2D32f cvSize2D32f( double width, double height );
```

CvRect

구형의 오프셋과 사이즈

```
typedef struct CvRect
{
    int x; /* 구형의 좌각의 x 좌표 */
    int y; /* 구형 상각(혹은 하각)의 y 좌표 */
    int width; /* 구형의 폭 */
    int height; /* 구형의 높이 */
}
CvRect;

/* constructor      */
inline CvRect cvRect( int x, int y, int width, int height );
```

CvScalar

4 개까지의 수를 납입하는 컨테이너

```
typedef struct CvScalar
{
    double val[4];
}
CvScalar;
```

```
    /* constructor       : val[0] (을)를 val0 그리고 초기화.val[1] (을)를 val1 그리고
초기화.… */
    inline CvScalar cvScalar( double val0, double val1=0,
                              double val2=0, double val3=0 );
    /* constructor       : val[0]...val[3] (을)를 val0123 그리고 초기화 */
    inline CvScalar cvScalarAll( double val0123 );

    /* constructor       : val[0] (을)를 val0 그리고 초기화.val[1]...val[3] (을)를 0
그리고 초기화 */
    inline CvScalar cvRealScalar( double val0 );
```

CvTermCriteria

반복 알고리즘을 위한 종료 조건

```
#define CV_TERMCRIT_ITER    1
#define CV_TERMCRIT_NUMBER  CV_TERMCRIT_ITER
#define CV_TERMCRIT_EPS     2

typedef struct CvTermCriteria
{
    int    type;  /* CV_TERMCRIT_ITER (와)과 CV_TERMCRIT_EPS 의 편성 */
    int    max_iter; /* 반복수의 최대치 */
    double epsilon; /* 목표 정도 */
}
CvTermCriteria;

/* constructor      */
inline  CvTermCriteria  cvTermCriteria( int type, int max_iter, double epsilon );

/* 종료 조건을 체크해,type=CV_TERMCRIT_ITER+CV_TERMCRIT_EPS (으)로 설정해,
   반복수의 max_iter(와)과 eprilon 의 양쪽 모두가 유효하게 되도록(듯이) 변환한다 */
CvTermCriteria cvCheckTermCriteria( CvTermCriteria criteria,
                                    double default_eps,
                                    int default_max_iters );
```

CvMat

다중 행렬

```
    typedef struct CvMat
    {
        int type; /* CvMat 서명 (CV_MAT_MAGIC_VAL).요소의 형태와 플래그 */
        int step; /* 전행의 아르바이트장 */
```

```
    int* refcount; /* 내부적으로 이용되는 데이터 reference counter */

    union
    {
        uchar* ptr;
        short* s;
        int* i;
        float* fl;
        double* db;
    } data; /* 데이터 포인터 */

#ifdef __cplusplus
    union
    {
        int rows;
        int height;
    };

    union
    {
        int cols;
        int width;
    };
#else
    int rows; /* 행의 수 */
    int cols; /* 열의 수 */
#endif

} CvMat;
```

CvMatND

다차원, 다채널의 조밀한 행렬

```
typedef struct CvMatND
{
    int type; /* CvMatND 서명 (CV_MATND_MAGIC_VAL).요소의 형태와 플래그 */
    int dims; /* 배열의 차원수 */

    int* refcount; /* 내부적으로 이용되는 데이터 reference counter */

    union
    {
        uchar* ptr;
        short* s;
        int* i;
```

```
        float* fl;
        double* db;
    } data; /* 데이터 포인터 */

    /* 각 차원에서의(요소수, 요소간의 아르바이트 거리)의 조 */
    struct
    {
        int size;
        int step;
    }
    dim[CV_MAX_DIM];

} CvMatND;
```

CvSparseMat

다차원, 다채널의 드문드문한 행렬

```
typedef struct CvSparseMat
{
    int type;            /* CvSparseMat 서명(CV_SPARSE_MAT_MAGIC_VAL).요소의 형태와
플래그 */
    int dims;            /* 차원수 */
    int* refcount;       /* reference counter - 사용되지 않는다 */
    struct CvSet* heap;  /* 해시 테이블 노드의 보존 영역(풀) */
    void** hashtable;    /* 해시 테이블 : 각 엔트리는, 같을 「hashvalue 의 hashsize
(을)를 법으로 하는 잉여」
                           (을)를 노드의 리스트로서 가진다 */
    int hashsize;        /* 해시 테이블의 사이즈 */
    int total;           /* 엉성한 배열 노드의 총수 */
    int valoffset;       /* 배열 노드의 값의 오프셋(아르바이트 단위) */
    int idxoffset;       /* 배열 노드의 인덱스의 오프셋(아르바이트 단위) */
    int size[CV_MAX_DIM]; /* 차원 사이즈의 배열 */

} CvSparseMat;
```

IplImage

IPL 이미지 헤더

```
typedef struct _IplImage
{
    int  nSize;         /* sizeof(IplImage) */
    int  ID;            /* 버젼 (=0)*/
    int  nChannels;     /* OpenCV 의 대부분의 함수가,1,2,3 및 4 채널을 서포트한다 */
    int  alphaChannel;  /* OpenCV 그럼 무시된다 */
```

```
        int  depth;         /* 픽셀의 색심도의 비트수 :
                               IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U,
                               IPL_DEPTH_16S, IPL_DEPTH_32S,
                               IPL_DEPTH_32F, IPL_DEPTH_64F 하지만 서포트된다 */
        char colorModel[4]; /* OpenCV 그럼 무시된다 */
        char channelSeq[4]; /* 전술 */
        int  dataOrder;     /* 0 - 인타리브카라체넬.1 - 분리 칼라 채널,
                               cvCreateImage 하지만 작성할 수 있는 것은, 인타리브
이미지만.*/
        int  origin;        /* 0 - 좌상 원점,
                               1 - 좌하 원점(Windows 의 비트 맵 형식) */
        int  align;         /* 이미지의 행의 얼라이먼트(4 혹은 8).
                               OpenCV (은)는 이것을 무시하고, 대신에 widthStep (을)를
사용한다. */
        int  width;         /* 이미지의 픽셀폭 */
        int  height;        /* 이미지의 픽셀 높이 */
        struct _IplROI *roi;/* 이미지 ROI.이것이 NULL (이)가 아닌 경우, 이 특정의 영역이
처리의 대상이 된다 */
        struct _IplImage *maskROI; /* OpenCV 그럼 반드시 NULL */
        void  *imageId;     /* 전술 */
        struct _IplTileInfo *tileInfo; /* 전술 */
        int  imageSize;     /* 이미지 데이터의 바이트사이즈
                               (인타리브데이타의 경우는,=image->height*image->widthStep)
*/
        char *imageData;    /* 얼라이먼트가 조정된 이미지 데이터에의 포인터 */
        int  widthStep;     /* 얼라이먼트가 조정된 이미지의 행의 바이트사이즈 */
        int  BorderMode[4]; /* 이미지 경계의 설정.OpenCV 그럼 무시된다 */
        int  BorderConst[4]; /* 전술 */
        char *imageDataOrigin; /* 오리지날 이미지 데이터에의 포인터
                                  (얼라이먼트가 조정되고 있다고는 할 수 없다)
                                  - 이것은 이미지를 올바르게 해방하기 위해서 필요. */
    }
    IplImage;
```

IplImage 구조체는 원래,*Intel Image Processing Library* 고유의 것이었다. OpenCV 그럼 IplImage 포맷의 일부만을 서포트하고 있다.

- alphaChannel 하 OpenCV 그럼 무시된다.
- colorModel (와)과 channelSeq 하 OpenCV 그럼 무시된다. OpenCV 의 색공간(color space)를 변환하는 함수 cvCvtColor (은)는, 변환원과 변환처의 색공간(color space)를 파라미터로 한다.
- dataOrder 하 IPL_DATA_ORDER_PIXEL (이)가 아니면 안된다(색채널은 인타리브 된다). 그러나,COI(channel of interest)(이)가 설정되어 있는 경우, 그 채널이 처리의 대상이 된다.
- align 하 OpenCV 그럼 무시되어 이미지의 행방향에의 연속적인 액세스에 widthStep 하지만 이용된다.
- maskROI (은)는 서포트되지 않는다.마스크를 이용하는 함수는, 그것을 다른 파라미터로서 취급한다. OpenCV 그럼 마스크도 8 비트이지만,IPL 그럼 1 비트가 되고 있다.

- tileInfo (은)는 서포트되지 않는다.
- BorderMode (와)과 BorderConst (은)는 서포트되지 않는다. OpenCV 의 각 함수는, 미리 결정할 수 있던 하나의 경계 설정(많은 경우는, 복제 경계 모드)으로 근방의 픽셀을 처리한다.

상술의 제한에 가세해 OpenCV 그럼 ROI 의 취급이 다르다. OpenCV그럼 모든 입력 및 출력 이미지의 사이즈, 또는 그 ROI 사이즈가 정확하게 일치할 필요가 있다 (처리에 의존한다.예를 들면 cvPyrDown 의 출력폭(높이)은, 입력폭(높이)의 1/2±1(이)가 아니면 안된다).그러나IPL 그럼 중복 하는 영역이 처리의 대상이 된다.즉 이미지의 사이즈, 또는 ROI 의 사이즈가 모든 이미지에 대해 다를 가능성도 있다.

2.배열 조작

2-1초기화(Initialization)

CreateImage

헤더의 작성과 데이터 영역의 확보

```
IplImage* cvCreateImage( CvSize size, int depth, int channels );
```

size

이미지의 폭과 높이.

depth

이미지 요소의 빗트데프스.이하 중의 언젠가.

IPL_DEPTH_8U - 부호 없음 8 비트 정수

IPL_DEPTH_8S - 부호 있어 8 비트 정수

IPL_DEPTH_16U - 부호 없음 16 비트 정수

IPL_DEPTH_16S - 부호 있어 16 비트 정수

IPL_DEPTH_32S - 부호 있어 32 비트 정수

IPL_DEPTH_32F - 단정도 부동 소수점수(실수)

IPL_DEPTH_64F - 배정도 부동 소수점수(실수)

channels

요소(픽셀) 마다의 채널수.1, 2, 3, 4 의 언젠가.이 채널은 인타리브 된다.예를 들면, 통상의 칼라 이미지의 데이터 레이아웃은,

b0 g0 r0 b1 g1 r1 ...

되고 있다.일반적인 IPL 이미지 포맷은 논인타리브 이미지를 격납할 수 있어 이것을 처리할 수 있다 OpenCV 의 함수도 존재하지만, 이 함수는 인타리브 이미지 밖에 작성할 수 없다.

함수 cvCreateImage (은)는 헤더를 작성해, 데이터 영역을 확보한다.이것은, 이하의 형식을 단축한 것이다.

```
header = cvCreateImageHeader(size,depth,channels);
cvCreateData(header);
```

CreateImageHeader

메모리 확보와 초기화를 실시해,IplImage 구조체를 돌려준다

```
IplImage* cvCreateImageHeader( CvSize size, int depth, int channels );
```

size

이미지의 폭과 높이.

depth

이미지의 카라데프스(CreateImage (을)를 참조).

channels

채널수(CreateImage (을)를 참조).

함수 cvCreateImageHeader (은)는, 메모리 확보와 초기화를 실시해, IplImage 구조체를 돌려준다.이것은,

```
iplCreateImageHeader( channels, 0, depth,
                      channels == 1 ? "GRAY" : "RGB",
                      channels == 1 ? "GRAY" : channels == 3 ? "BGR" :
                      channels == 4 ? "BGRA" : "",
                      IPL_DATA_ORDER_PIXEL, IPL_ORIGIN_TL, 4,
                      size.width, size.height,
                      0,0,0,0);
```

(와)과 닮아 있지만, 이 함수는 디폴트에서는 IPL 의 함수를 이용하지 않는다(매크로 CV_TURN_ON_IPL_COMPATIBILITY 도 참조하는 것).

ReleaseImageHeader

헤더를 해방한다

```
void cvReleaseImageHeader( IplImage** image );
```

image

확보한 헤더에의 포인터의 포인터.

함수 cvReleaseImageHeader (은)는, 헤더를 해방한다.이것은,

```
if( image )
{
    iplDeallocate( *image, IPL_IMAGE_HEADER | IPL_IMAGE_ROI );
    *image = 0;
}
```

(와)과 닮아 있지만, 이 함수는 디폴트에서는 IPL 의 함수를 이용하지 않는다(매크로 CV_TURN_ON_IPL_COMPATIBILITY 도 참조하는 것).

ReleaseImage

헤더와 이미지 데이터를 해방한다

```
void cvReleaseImage( IplImage** image );
```

image

확보한 이미지 헤더에의 포인터의 포인터.

함수 cvReleaseImage (은)는, 헤더와 이미지 데이터를 해방한다. 이것은, 이하의 형식을 단축한 것이다.

```
    if( *image )
    {
        cvReleaseData( *image );
        cvReleaseImageHeader( image );
    }
```

InitImageHeader

유저에 의해서 확보된 이미지 헤더를 초기화한다

```
IplImage* cvInitImageHeader( IplImage* image, CvSize size, int depth,
                             int channels, int origin=0, int align=4 );
```

image

초기화되는 이미지 헤더.

size

이미지의 폭과 높이.

depth

이미지의 카라데프스(CreateImage (을)를 참조).

channels

채널수(CreateImage (을)를 참조).

origin

IPL_ORIGIN_TL 또는 IPL_ORIGIN_BL.

align

이미지의 행의 얼라이먼트, 통상은4, 혹은 8 아르바이트.

함수 cvInitImageHeader (은)는, 유저로부터 건네받은 포인터가 가리키는 이미지의 헤더 구조체를 초기화해, 그 포인터를 돌려준다.

CloneImage

이미지의 완전한 카피를 작성한다

```
IplImage* cvCloneImage( const IplImage* image );
```

image

오리지날 이미지.

함수 cvCloneImage (은)는, 헤더,ROI, 데이터를 포함한 이미지의 완전한 카피 (을)를 작성한다.

SetImageCOI

주어진 값을 channel of interest(COI)(으)로서 세트 한다

```
void cvSetImageCOI( IplImage* image, int coi );
```

image

이미지 헤더.

coi

COI(Channel of interest).

함수 cvSetImageCOI (은)는, 주어진 값을 COI(channel of interest)(으)로서 세트 한다. 치 0 (은)는 모든 채널이 선택되고 있는 일을 의미해, 값 1 (은)는 최초의 채널이 선택되고 있는 일을 의미한다. 만약 COI 하지만 NULL 한편, 파라미터가 coi != 0 의 경우에는,COI 하지만 확보된다. OpenCV 의 함수의 상당수는 COI (을)를 서포트하고 있지 않는 것에 주의하는 것. 다른 이미지/행렬 채널을 처리하려면 , (cvCopy (이)나 cvSplit 에 의해서) 다른 이미지/행렬에 채널을 카피해, 그것을 처리하고 나서, 필요하면 그 결과를(cvCopy (이)나 cvCvtPlaneToPix 에 의해서) 다시 한번 더 카피해 돌려준다.

GetImageCOI

COI 의 인덱스를 되돌린다

```
int cvGetImageCOI( const IplImage* image );
```

image

이미지 헤더.

함수 cvGetImageCOI (은)는, 이미지의 COI(channel of interest)(을)를 돌려준다(전채널이 선택되는 경우에는,0 하지만 돌려주어진다).

SetImageROI

주어진 구형을 이미지의 ROI (으)로서 세트 한다

```
void cvSetImageROI( IplImage* image, CvRect rect );
```

image

이미지 헤더.

rect

ROI (을)를 나타내는 구형.

함수 cvSetImageROI (은)는, 주어진 구형을 이미지의 ROI (으)로서 세트 한다. 만약,ROI 하지만 NULL 그리고, 파라미터 rect 의 값이 이미지 전체가 아닌 경우에는,ROI 하지만 확보된다. COI 과는 달리,OpenCV 의 함수의 대부분이 ROI (을)를 서포트하고 있다. 그리고,ROI (은)는 어느 의미, 다른 이미지로서 다루어진다(예를 들면, 모든 픽셀 좌표는 ROI 의 좌상 혹은 좌하(이미지의 원점에 의존)로부터 카운트 된다).

ResetImageROI

이미지의 ROI(을)를 해방한다

```
void cvResetImageROI( IplImage* image );
```

image

이미지 헤더.

함수 cvResetImageROI(은)는, 이미지의 ROI(을)를 해방한다. 해방 후는 전이미지가 선택되고 있는 상태와 같게 된다. 이하와 같이 해도, 똑같이 전선택을 할 수 있지만, 이 방법에서는 image->roi(은)는 해방되지 않는다.

```
cvSetImageROI( image, cvRect( 0, 0, image->width, image->height ));
cvSetImageCOI( image, 0 );
```

GetImageROI

이미지의 ROI 좌표를 돌려준다

```
CvRect cvGetImageROI( const IplImage* image );
```

image

이미지 헤더.

함수 cvGetImageROI(은)는, 이미지의 ROI 좌표를 돌려준다. ROI 하지만 존재하지 않는 경우에는, 구형 cvRect(0,0,image->width,image->height) 하지만 돌려주어진다.

CreateMat

새로운 행렬을 작성한다

```
CvMat* cvCreateMat( int rows, int cols, int type );
```

rows

행렬의 행수.

cols

행렬의 렬수.

type

행렬 요소의 종류.통상, 다음과 같은 지정 형식이 된다. CV_<bit_depth>(S|U|F)C<number_of_channels>.예를 들면,

CV_8UC1 (은)는, 부호 없음 8 비트 1 채널 행렬, CV_32SC2 (은)는, 부호 있어 32 비트 2 채널 행렬을 의미한다.

함수 cvCreateMat(은)는, 새로운 행렬과 그 내부 데이터를 위한 헤더를 확보해, 작성된 행렬에의 포인터를 돌려준다.이것은 이하의 생략형이다.

```
CvMat* mat = cvCreateMatHeader( rows, cols, type );
cvCreateData( mat );
```

행렬은 행 단위로 보존되어 모든 행은 4 아르바이트로 얼라이먼트 된다.

CreateMatHeader

새로운 행렬의 헤더를 작성한다

```
CvMat* cvCreateMatHeader( int rows, int cols, int type );
```

rows

행렬의 행수.

cols

행렬의 렬수.

type

행렬 요소의 종류(cvCreateMat(을)를 참조).

함수 cvCreateMatHeader(은)는, 새로운 행렬의 헤더를 작성해, 그 포인터를 돌려준다. 게다가 cvCreateData (을)를 이용하는지,cvSetData 에 의해, 유저가 확보한 데이터 영역을 명시적으로 세트 하는 것으로, 행렬 데이터가 확보된다.

ReleaseMat

행렬을 해방한다

```
void cvReleaseMat( CvMat** mat );
```

mat

행렬에의 포인터의 포인터.

함수 cvReleaseMat (은)는, 행렬 데이터의 reference counter 를 감소 하고, 행렬 헤더를 해방한다.

```
    if( *mat )
        cvDecRefData( *mat );
    cvFree( (void**)mat );
```

InitMatHeader

행렬 헤더를 초기화한다

```
CvMat* cvInitMatHeader( CvMat* mat, int rows, int cols, int type,
                        void* data=NULL, int step=CV_AUTOSTEP );
```

mat

초기화하는 행렬의 헤더에의 포인터.

rows

행렬의 행수.

cols

행렬의 렬수.

type

행렬 요소의 종류.

data

행렬의 헤더로 지정되는 데이터에의 포인터(옵션).

step

할당할 수 있었던 데이터의 유키나가를 아르바이트 단위로 나타낸다. 디폴트에서는,step에는 가능한 한 작은 값이 이용된다.즉, 행렬이 연속하는 행간에 갭이 존재하지 않는다.

함수 cvInitMatHeader (은)는, 이미 확보된 구조체 CvMat (을)를 초기화한다. 이것은,OpenCV 의 행렬 함수로 생 데이터를 처리하기 위해서 이용할 수도 있다.

예를 들면, 이하의 코드는 두 개의 행렬의 적을 계산해, 통상의 행렬로서 격납한다.

두 개의 행렬의 적의 계산

```
   double a[] = { 1, 2, 3, 4,
                  5, 6, 7, 8,
                  9, 10, 11, 12 };

   double b[] = { 1, 5, 9,
                  2, 6, 10,
                  3, 7, 11,
                  4, 8, 12 };

   double c[9];
   CvMat Ma, Mb, Mc ;

   cvInitMatHeader( &Ma, 3, 4, CV_64FC1, a );
   cvInitMatHeader( &Mb, 4, 3, CV_64FC1, b );
   cvInitMatHeader( &Mc, 3, 3, CV_64FC1, c );

   cvMatMulAdd( &Ma, &Mb, 0, &Mc );
   // 이 행렬 c 에는, 행렬 a(3×4) (와)과 행렬 b(4×3) 의 적이 들어가 있다
```

Mat

행렬 헤더를 초기화한다(경량판)

```
CvMat cvMat( int rows, int cols, int type, void* data=NULL );
```

rows

행렬의 행수.

cols

행렬의 렬수.

type

행렬 요소의 종류(CreateMat (을)를 참조).

data

행렬의 헤더로 지정되는 데이터에의 포인터(옵션).
함수 cvMat (은)는, cvInitMatHeader 의 고속의 인 라인 치환이다.즉, 이것은 이하와 등가이다.

```
CvMat mat;
cvInitMatHeader( &mat, rows, cols, type, data, CV_AUTOSTEP );
```

CloneMat

행렬의 카피를 작성한다

```
CvMat* cvCloneMat( const CvMat* mat );
```

mat

입력 행렬.

함수 cvCloneMat (은)는, 입력 행렬의 카피를 작성해, 그 포인터를 돌려준다.

CreateMatND

다차원의 조밀한 배열을 작성한다

```
CvMatND* cvCreateMatND( int dims, const int* sizes, int type );
```

dims

배열의 차원수.CV_MAX_DIM(디폴트에서는32.빌드시로 변경될 가능성도 있다)를 넘어선 안 된다.

sizes

차원 사이즈의 배열.

type

배열 요소의 종류.CvMat 의 것과 같다.

함수 cvCreateMatND (은)는, 다차원의 조밀한 배열의 헤더와 그 내부 데이터를 확보해, 작성된 배열의 포인터를 돌려준다. 이것은 이하의 생략형이다.

```
CvMatND* mat = cvCreateMatNDHeader( dims, sizes, type );
cvCreateData( mat );
```

배열 데이터는 행 단위로 보존된다.모든 행은 4 아르바이트로 얼라이먼트 된다.

CreateMatNDHeader

새로운 행렬의 헤더를 작성한다

```
CvMatND* cvCreateMatNDHeader( int dims, const int* sizes, int type );
```

dims

배열의 차원수.

sizes

차원 사이즈의 배열.

type

배열 요소의 종류.CvMat 의 것과 같다.

함수 cvCreateMatND (은)는, 다차원의 조밀한 배열의 헤더를 확보한다. 게다가 cvCreateData (을)를 이용하는지, cvSetData 에 의해, 유저가 확보한 데이터 영역을 명시적으로 세트 하는 것으로, 배열 데이터가 확보된다.

ReleaseMatND

다차원 배열을 해방한다

```
void cvReleaseMatND( CvMatND** mat );
```

mat

배열에의 포인터의 포인터.

함수 cvReleaseMatND (은)는, 배열 데이터의 reference counter 를 감소 하고, 배열 헤더를 해방한다.

```
    if( *mat )
        cvDecRefData( *mat );
    cvFree( (void**)mat );
```

InitMatNDHeader

다차원 배열의 헤더를 초기화한다

```
CvMatND* cvInitMatNDHeader( CvMatND* mat, int dims, const int* sizes, int type, void*
data=NULL );
```

mat

초기화하는 배열의 헤더에의 포인터.

dims

배열의 차원수.

sizes

차원 사이즈의 배열.

type

배열 요소의 종류.CvMat 의 것과 같다.

data

행렬의 헤더로 지정되는 데이터에의 포인터(옵션).

함수 cvInitMatNDHeader (은)는, 유저에 의해서 확보된 구조체 CvMatND (을)를 초기화한다.

CloneMatND

다차원 배열의 완전한 카피를 작성한다

```
CvMatND* cvCloneMatND( const CvMatND* mat );
```

mat

입력 배열.
함수 cvCloneMatND (은)는, 입력 배열의 카피를 작성해, 그 포인터를 돌려준다.

DecRefData

배열 데이터의 reference counter 를 감소 한다

```
void cvDecRefData( CvArr* arr );
```

arr

배열 헤더.
함수 cvDecRefData (은)는, reference counter 의 포인터가 NULL (이)가 아닌 경우에 CvMat 혹은 CvMatND 의 데이터의 reference counter 를 감소 해, 한층 더 카운터가 0 (이)가 되었을 경우에는 데이터를 해방한다. 현재의 실장에서는, 데이터가 함수 cvCreateData 에 의해서 확보되었을 경우에게만, reference counter 는 NULL (은)는 아니게 된다. 그 외의 경우로서 이하의 것이 있다.
cvSetData 에 의해서, 외부 데이터를 헤더에 할당할 수 있었다.
행렬 헤더가, 보다 큰 행렬 혹은 이미지의 일부가 되어 있다.
행렬 헤더가, 이미지 혹은 n 차원 행렬 헤더로부터 변환되었다.

이러한 경우, reference counter 는 NULL 에 세트 되어 감소 되지 않는다. 데이터가 해방되는지 아닌지에 관련되지 않고, 데이터 포인터와 reference counter 포인터는 이 함수에 의해서 클리어 된다.

IncRefData

배열 데이터의 reference counter 을 인크리먼트(increment) 한다

```
int cvIncRefData( CvArr* arr );
```

arr

배열 헤더.
함수 cvIncRefData (은)는, reference counter 포인터가 NULL (이)가 아닌 경우에, CvMat 혹은 CvMatND 의 데이터의 reference counter 을 인크리먼트(increment) 해, 새로운 카운터치를 돌려준다. 그렇지 않은 경우는 0 (을)를 돌려준다.

CreateData

배열 데이터를 확보한다

```
void cvCreateData( CvArr* arr );
```

arr

배열 헤더.
함수 cvCreateData (은)는, 이미지, 행렬 혹은 다차원 배열의 데이터를 확보한다. 행렬의 경우는 OpenCV 의 확보 함수가 이용된다.IplImage 의 경우도 OpenCV 의 함수가 이용된다.

다만,CV_TURN_ON_IPL_COMPATIBILITY 하지만 불렸을 경우는 예외적으로, 데이터를 확보하기 위해서 IPL 함수가 이용된다.

ReleaseData

배열 데이터를 해방한다

```
void cvReleaseData( CvArr* arr );
```

arr

배열 헤더.

함수 cvReleaseData (은)는, 배열 데이터를 해방한다. CvMat 혹은 CvMatND 의 경우, 이것은 단지 cvDecRefData() (을)를 부르는 것만으로 있다.즉, 이 함수는 외부 데이터를 해방할 수 없다. cvCreateData 의 주의 사항도 참조하는 것.

SetData

유저 데이터를 배열 헤더에 할당한다

```
void cvSetData( CvArr* arr, void* data, int step );
```

arr

배열 헤더.

data

유저 데이터.

step

아르바이트 단위로 나타내진 행의 길이.

함수 cvSetData (은)는, 유저 데이터를 배열의 헤더에 할당한다. 헤더는, 함수 cvCreate*Header, 함수 cvInit*Header 혹은 함수 cvMat(행렬의 경우)(을)를 이용하고, 미리 초기화 되는 것이 당연하다.

GetRawData

배열의 저레벨 정보를 꺼낸다

```
void cvGetRawData( const CvArr* arr, uchar** data,
                   int* step=NULL, CvSize* roi_size=NULL );
```

arr

배열 헤더.

data

출력인 전이미지의 원점에의 포인터, 혹은 ROI 하지만 설정되어 있는 경우는 ROI 의 원점에의 포인터.

step

출력인 아르바이트 단위로 나타내진 행의 길이.

roi_size

출력이다ROI 사이즈.

함수 cvGetRawData (은)는, 배열 데이터에 관한 저레벨 정보를 변수에 출력한다. 모든 출력 파라미터는 임의이며, 몇개의 포인터는 NULL 에 세트 되는 경우가 있다. 배열이 ROI (을)를 가진다 IplImage 인 경우,ROI 의 파라미터가 돌려주어진다.

이하에, 이 함수를 이용해 배열의 요소에 액세스 하는 예를 나타낸다.

GetRawData (을)를 이용하고, 싱글 채널 부동 소수점형수배열의 요소의 절대치를 계산한다.

```
float* data;
int step;

CvSize size;
int x, y;

cvGetRawData( array, (uchar**)&data, &step, &size );
step /= sizeof(data[0]);

for( y = 0; y < size.height; y++, data += step )
    for( x = 0; x < size.width; x++ )
        data[x] = (float)fabs(data[x]);
```

GetMat

임의의 배열에 대한 행렬 헤더를 돌려준다

```
CvMat* cvGetMat( const CvArr* arr, CvMat* header, int* coi=NULL, int allowND=0 );
```

arr

입력 배열.

header

텐포라리밧파로서 이용되는 구조체 CvMat 에의 포인터.

coi

COI(을)를 기억하기 위한 , 옵션의 출력 파라미터.

allowND

이것이 0 (이)가 아닌 경우, 이 함수는 다차원의 조밀한 배열(CvMatND*)(을)를 취급하는 것이 가능하고, 2차원 행렬(CvMatND 하지만2차원의 경우) 혹은 1차원 행렬(CvMatND하지만 1차원, 혹은 2차원보다 큰 경우)를 돌려준다. 배열은 연속이 아니면 안된다.

함수 cvGetMat (은)는, 입력 배열에 대한 행렬 헤더를 돌려준다. 입력 배열이 될 수 있는 것은, 행렬 - CvMat, 이미지 - IplImage 혹은, 다차원의 조밀한 배열 - CvMatND*(마지막 예는,allowND != 0 의 경우뿐)이다. 행렬의 경우, 이 함수는 단지 입력 포인터를 돌려준다. IplImage* 혹은 CvMatND* 의 경우는, 현재의 이미지의 ROI 의 파라미터로 구조체 header (을)를 초기화해, 이 텐포라리 구조체에의 포인터를 돌려준다. COI 하 CvMat 그럼 서포트되지 않기 때문에, 이것은 따로 돌려주어진다.

이 함수는, 같은 코드로 2 종류의 배열 - IplImage 및 CvMat - (을)를 취급하는 간단한 방법을 제공한다. 함수 cvGetImage 에 의해서,CvMat (으)로부터 IplImage 에의 역변환이 가능하다.

입력 배열은, 확보 혹은 첨부된 내부 데이터를 가지지 않으면 안되어, 그렇지 않은 경우는, 이 함수는 실패한다.

입력 배열이, 평면(이차원) 데이터 레이아웃 및 COI (을)를 가진다 IplImage 의 경우, 이 함수는 선택된 평면에의 포인터 및 COI = 0 (을)를 돌려준다. OpenCV 의 함수를 이용하고, 평면 데이터 레이아웃을 가지는 멀티 채널 이미지를 평면마다 처리하는 것이 가능하다.

GetImage

임의의 배열에 대한 이미지 헤더를 돌려준다

```
IplImage* cvGetImage( const CvArr* arr, IplImage* image_header );
```

arr

입력 배열.

image_header

텐포라리밧파로서 이용되는 구조체 IplImage 에의 포인터.

함수 cvGetImage (은)는, 입력 배열, 행렬 - CvMat*, 이미지 - IplImage*, 에 대한 이미지 헤더를 돌려준다. 이미지의 경우는, 이 함수는 단지 입력 포인터를 돌려준다.CvMat* 의 경우는, 입력 행렬의 파라미터로 구조체 image_header (을)를 초기화한다. ROI 하지만 세트 되고 있는 경우,IplImage (으)로부터 CvMat 에 변환한 후에, 그 CvMat (으)로부터 IplImage 에 되돌리면, 다른 헤더가 될 가능성이 있다. 따라서, 이미지의 길이를 그 폭과 얼라이먼트로부터 계산하는 IPL 함수는, 이 함수의 결과로서 얻을 수 있는 이미지에 대해서는 실패할 가능성이 있다.

CreateSparseMat

드문드문한 배열을 작성한다

```
CvSparseMat* cvCreateSparseMat( int dims, const int* sizes, int type );
```

dims

배열의 차원수.조밀한 행렬과는 반대로, 차원수는 실질적으로는 무제한하다(2^{16} 까지).

sizes

차원 사이즈의 배열.

type

배열 요소의 종류.CvMat 의 것과 같다.

함수 cvCreateSparseMat (은)는, 다차원의 드문드문한 배열을 확보한다. 초기 상태에서는 배열은 요소를 가지지 않는다. 즉,cvGet*D 혹은 cvGetReal*D (은)는, 전인덱스에 대해서 0 (을)를 돌려준다.

ReleaseSparseMat

드문드문한 배열을 해방한다

void cvReleaseSparseMat(CvSparseMat** mat);

mat

배열에의 포인터의 포인터.

함수 cvReleaseSparseMat (은)는, 드문드문한 배열을 해방해, 종료시에 배열 포인터를 클리어 한다.

CloneSparseMat

드문드문한 배열의 완전한 카피를 작성한다

CvSparseMat* cvCloneSparseMat(const CvSparseMat* mat);

mat

입력 배열.

2-2 구성요소에의 접근과와 부분 배열(Accessing Elements and sub-Arrays)

GetSubRect

입력 이미지 또는 행렬의 구형 부분 배열에 상당하는 헤더를 돌려준다

CvMat* cvGetSubRect(const CvArr* arr, CvMat* submat, CvRect rect);

arr

입력 배열.

submat

결과적으로 얻을 수 있는 부분 배열의 헤더에의 포인터.

rect

주목하는 구형 영역의,0 (을)를 기준으로 한 좌표.

함수 cvGetSubRect(은)는, 입력 배열중의 지정한 구형 영역에 상당하는 헤더를 돌려준다. 즉, 입력 배열의 일부의 구형 영역을, 독립한 배열로서 취급할 수 있도록(듯이) 한다. 이 함수에서는 ROI (을)를 고려해, 실제로는 ROI 의 부분 배열이 꺼내진다.

GetRow, GetRows

배열중의 1 행 또는, 지정된 범위의 행(복수행)을 돌려준다

```
CvMat* cvGetRow( const CvArr* arr, CvMat* submat, int row );
CvMat* cvGetRows( const CvArr* arr, CvMat* submat, int start_row, int end_row, int
delta_row=1 );
```

arr

입력 배열.

submat

결과적으로 얻을 수 있는 부분 배열의 헤더에의 포인터.

row

선택한 행의,0(을)를 기준으로 한 인덱스.

start_row

범위의 최초의(이 값을 포함한다) 행의,0(을)를 기준으로 한 인덱스.

end_row

범위의 마지막(이 값을 포함하지 않는다) 행의,0(을)를 기준으로 한 인덱스.

delta_row

행의 범위의 인덱스 간격. 이 함수는,start_row(으)로부터end_row(는 포함하지 않는다)까지,delta_row마다행을 추출한다.

함수 GetRow(와)과 GetRows(은)는, 입력 배열중의 지정했다 1 행, 혹은 범위의 복수행에 상당하는 헤더를 돌려준다.주석 : GetRow 하 cvGetRows 의 쇼트 컷이다.

```
cvGetRow( arr, submat, row );  // ~ cvGetRows( arr, submat, row, row + 1, 1 );
```

GetCol, GetCols

배열중의 1 열 또는, 지정된 범위의 열(복수열)을 돌려준다

```
CvMat* cvGetCol( const CvArr* arr, CvMat* submat, int col );
CvMat* cvGetCols( const CvArr* arr, CvMat* submat, int start_col, int end_col );
```

arr

입력 배열.

submat

결과적으로 얻을 수 있는 부분 배열의 헤더에의 포인터.

col

선택한 열의,0(을)를 기준으로 한 인덱스.

start_col

범위의 최초의(이 값을 포함한다) 열의,0(을)를 기준으로 한 인덱스

end_col

범위의 마지막(이 값을 포함하지 않는다) 열의,0(을)를 기준으로 한 인덱스.

함수 GetCol(와)과 GetCols(은)는 입력 배열중의 지정했다 1 열, 혹은 범위의 복수열에 상당하는 헤더를 돌려준다.주석 : GetCol 하 cvGetCols 의 쇼트 컷이다.

```
cvGetCol( arr, submat, col ); // ~ cvGetCols( arr, submat, col, col + 1 );
```

GetDiag

배열의 대각열의 하나를 돌려준다

```
CvMat* cvGetDiag( const CvArr* arr, CvMat* submat, int diag=0 );
```

arr

입력 배열.

submat

결과적으로 얻을 수 있는 부분 배열의 헤더에의 포인터.

diag

대각 배열.0(은)는 메인의 대각열에 대응해,-1(은)는 메인대 각열의 하나상의 기울기열,1(은)는 메인대 각열의 하나하의 기울기열, 이라고 하는 것처럼 대응한다.

함수 cvGetDiag(은)는, 입력 배열중의 지정된 대각열에 상당하는 헤더를 돌려준다.

GetSize

행렬 또는 이미지의 ROI 의 사이즈를 돌려준다

```
CvSize cvGetSize( const CvArr* arr );
```

arr

배열의 헤더.

함수 cvGetSize(은)는, 입력 행렬 또는 이미지의 행수(CvSize::height)(와)과 렬수(CvSize::width)(을)를 돌려준다.이미지의 경우는 ROI 의 사이즈가 돌려주어진다.

InitSparseMatIterator

드문드문한 배열 요소의 이테레이타를 초기화한다

```
CvSparseNode* cvInitSparseMatIterator( const CvSparseMat* mat,
                                       CvSparseMatIterator* mat_iterator );
```

mat

입력 배열.

mat_iterator

초기화되는 이테레이타.

함수 cvInitSparseMatIterator(은)는, 드문드문한 배열 요소의 이테레이타를 초기화해, 선두 요소에의 포인터를 돌려준다.배열이 하늘때는 NULL(을)를 돌려준다.

GetNextSparseNode

드문드문한 배열에 대해 다음의 요소의 포인터를 돌려준다

```
CvSparseNode* cvGetNextSparseNode( CvSparseMatIterator* mat_iterator );
```

mat_iterator

드문드문한 배열의 이테레이타.

함수 cvGetNextSparseNode(은)는, 이테레이타를 다음의 드문드문한 행렬 요소에 움직여, 거기에의 포인터를 돌려준다. 현재의 버젼에서는, 개개의 요소는 해시 테이블에 보존되고 있으므로, 그것들에 특별한 차례는 없다. 드문드문한 행렬내를 어떻게 반복 처리 해 나갈까를, 이하의 샘플에 나타낸다.

cvInitSparseMatIterator (와)과 cvGetNextSparseNode (을)를 이용한 드문드문한 부동 소수점형 배열의 총화 계산.

```
    double sum;
    int i, dims = cvGetDims( array );
    CvSparseMatIterator mat_iterator;
    CvSparseNode* node = cvInitSparseMatIterator( array, &mat_iterator );

    for( ; node != 0; node = cvGetNextSparseNode( &mat_iterator ))
    {
        /* 요소 인덱스에의 포인터를 취득 */
        const int* idx = CV_NODE_IDX( array, node );
        /* 요소치의 취득(타입이 CV_32FC1 이라고 가정) */
        float val = *(float*)CV_NODE_VAL( array, node );
        printf( "(" );
        for( i = 0; i < dims; i++ )
            printf( "%4d%s", idx[i], i < dims - 1 "," : "): " );
        printf( "%gWn", val );

        sum += val;
    }

    printf( "WnTotal sum = %gWn", sum );
```

GetElemType

배열 요소의 타입을 돌려준다

```
int cvGetElemType( const CvArr* arr );
```

arr

입력 배열.

함수 GetElemType(은)는,cvCreateMat 의 설명에 기술되어 있는 이하와 같은 배열 요소의 타입을 돌려준다.

CV_8UC1 ... CV_64FC4

GetDims, GetDimSize

배열의 차원수와 그러한 사이즈, 또는 특정의 차원의 사이즈를 돌려준다

```
int cvGetDims( const CvArr* arr, int* sizes=NULL );
```

```
int cvGetDimSize( const CvArr* arr, int index );
```

arr

입력 배열.

sizes

배열의 차원의 크기를 나타내는 옵션의 출력 벡터.2차원 배열의 경우는1번째에 행수(높이), 다음은 렬수(폭)를 나타낸다.

index

0(을)를 기준으로 한 차원의 인덱스(행렬에서는0하행수,1(은)는 렬수를 나타낸다.이미지에서는0(은)는 높이, 1(은)는 폭을 나타낸다).

함수 cvGetDims(은)는 배열의 차원과 그러한 사이즈를 돌려준다. IplImage 또는 CvMat 의 경우에는, 이미지나 행렬의 행수에 관계없이 항상 2 (을)를 돌려준다. 함수 cvGetDimSize(은)는 특정의 차원의 사이즈(지정된 차원의 요소수)를 돌려준다.예를 들면, 다음의 코드는 배열 요소의 수를 계산하는 2 개의 방법이다.

```
// cvGetDims()(을)를 이용하는 방법
int sizes[CV_MAX_DIM];
int i, total = 1;
int dims = cvGetDims( arr, size );
for( i = 0; i < dims; i++ )
    total *= sizes[i];

// cvGetDims() (와)과 cvGetDimSize() (을)를 이용하는 방법
int i, total = 1;
int dims = cvGetDims( arr );
for( i = 0; i < dims; i++ )
    total *= cvGetDimsSize( arr, i );
```

Ptr*D

특정의 배열 요소에의 포인터를 돌려준다

```
uchar* cvPtr1D( const CvArr* arr, int idx0, int* type=NULL );
uchar* cvPtr2D( const CvArr* arr, int idx0, int idx1, int* type=NULL );
uchar* cvPtr3D( const CvArr* arr, int idx0, int idx1, int idx2, int* type=NULL );
uchar* cvPtrND( const CvArr* arr, const int* idx, int* type=NULL,
                                        int create_node=1, unsigned*
precalc_hashval=NULL );
```

arr

입력 배열.

idx0

요소 인덱스의,0(을)를 기준으로 한 제1성분.

idx1

요소 인덱스의,0(을)를 기준으로 한 제2성분.

idx2

요소 인덱스의,0(을)를 기준으로 한 제3성분.

idx

요소 인덱스의 배열.

type

옵션의 출력 파라미터.행렬 요소의 타입.

create_node

드문드문한 행렬에 대한 옵션의 입력 파라미터.비0의 경우, 지정된 요소가 존재하지 않을 때는 요소를 생성한다.

precalc_hashval

드문드문한 행렬에 대한 옵션의 입력 파라미터.포인터가NULL(이)가 아닐 때, 함수는 노드의 해시치를 재계산하지 않고, 지정된 장소로부터 차지해 온다. 이것에 의해, 페어 와이즈 조작의 속도가 향상한다.

함수 cvPtr*D(은)는, 특정의 배열 요소에의 포인터를 돌려준다.1 차원으로부터 N 차원까지의 조밀한 배열에의 연속적인 액세스에 이용되는 함수 cvPtr1D 의 경우를 제외하고, 배열의 차원수는 함수의 당겨 수라고 해 건네받는 인덱스의 수로 일치해야 한다.

같이 이 함수는, 드문드문한 배열에 대해서도 이용된다.지정한 노드가 존재하지 않는 경우, 함수는 그것을 생성해,0(을)를 세트 한다.

배열 요소에 액세스 하는 그 외의 함수(cvGet*D, cvGetReal*D, cvSet*D, cvSetReal*D)도 마찬가지이지만, 요소의 인덱스가 범위외이면, 에러가 일어난다.

Get*D

특정의 배열 요소를 돌려준다

```
CvScalar cvGet1D( const CvArr* arr, int idx0 );
CvScalar cvGet2D( const CvArr* arr, int idx0, int idx1 );
CvScalar cvGet3D( const CvArr* arr, int idx0, int idx1, int idx2 );
CvScalar cvGetND( const CvArr* arr, const int* idx );
```

arr

입력 배열.

idx0

요소 인덱스의0(을)를 기준으로 한 제1성분.

idx1

요소 인덱스의0(을)를 기준으로 한 제2성분.

idx2

요소 인덱스의0(을)를 기준으로 한 제3성분.

idx

요소 인덱스의 배열.

함수 cvGet*D(은)는, 특정의 배열 요소를 돌려준다.드문드문한 배열로, 지정한 노드가 존재하지 않는 경우, 이 함수는 0(을)를 돌려준다(이 함수에 의해서 새로운 노드는 생성되지 않는다).

GetReal*D

싱글 채널의 배열의 특정의 요소를 돌려준다

```
double cvGetReal1D( const CvArr* arr, int idx0 );
double cvGetReal2D( const CvArr* arr, int idx0, int idx1 );
double cvGetReal3D( const CvArr* arr, int idx0, int idx1, int idx2 );
double cvGetRealND( const CvArr* arr, const int* idx );
```

arr

입력 배열. 싱글 채널이 아니면 안 된다.

idx0

요소 인덱스의,0(을)를 기준으로 한 제1성분.

idx1

요소 인덱스의,0(을)를 기준으로 한 제2성분.

idx2

요소 인덱스의,0(을)를 기준으로 한 제3성분.

idx

요소 인덱스의 배열.

함수 cvGetReal*D(은)는, 싱글 채널의 배열의 특정의 요소를 돌려준다.배열이 멀티 채널의 경우는, 런타임 에러가 발생한다. 주석 : 함수 cvGet*D(은)는 싱글 채널과 멀티 채널의 배열에 대해서 안전하게 사용할 수 있지만, 약간 처리 속도가 늦다.

지정한 노드가 존재하지 않으면, 이 함수는 0(을)를 돌려준다(이 함수에 의해서 새로운 노드는 생성되지 않는다).

---

mGet

싱글 채널 부동 소수점형 행렬의 특정의 요소를 돌려준다

```
double cvmGet( const CvMat* mat, int row, int col );
```

mat

입력 행렬.

row

행의0(을)를 기준으로 한 인덱스.

col

열의0(을)를 기준으로 한 인덱스.

함수 cvmGet(은)는, 싱글 채널 부동 소수점형 행렬의 경우에 있어서의,cvGetReal2D 의 고속화판 함수이다. 인 라인 처리되어 배열의 타입이나 요소의 타입의 체크를 실시하지 않고, 또 행 열의 범위의 체크도 debug 모드 때 밖에 실시하지 않기 때문에 고속으로 있다.

Set*D

특정의 배열 요소를 변경한다

```
void cvSet1D( CvArr* arr, int idx0, CvScalar value );
```

```
void cvSet2D( CvArr* arr, int idx0, int idx1, CvScalar value );
void cvSet3D( CvArr* arr, int idx0, int idx1, int idx2, CvScalar value );
void cvSetND( CvArr* arr, const int* idx, CvScalar value );
```

arr

입력 배열.

idx0

요소 인덱스의,0(을)를 기준으로 한 제1성분.

idx1

요소 인덱스의,0(을)를 기준으로 한 제2성분.

idx2

요소 인덱스의,0(을)를 기준으로 한 제3성분.

idx

요소 인덱스의 배열.

value

할당하는 값.

함수 cvSet*D(은)는, 새로운 값을 지정한 배열 요소에 할당한다. 드문드문한 배열의 경우, 노드가 존재하지 않으면, 이 함수는 노드를 생성한다.

SetReal*D

특정의 배열 요소를 변경한다

```
void cvSetReal1D( CvArr* arr, int idx0, double value );
void cvSetReal2D( CvArr* arr, int idx0, int idx1, double value );
void cvSetReal3D( CvArr* arr, int idx0, int idx1, int idx2, double value );
void cvSetRealND( CvArr* arr, const int* idx, double value );
```

arr

입력 배열.

idx0

요소 인덱스의,0(을)를 기준으로 한 제1성분.

idx1

요소 인덱스의,0(을)를 기준으로 한 제2성분.

idx2

요소 인덱스의,0(을)를 기준으로 한 제3성분.

idx

요소 인덱스의 배열.

value

할당하는 값.

함수 cvSetReal*D(은)는 싱글 채널의 배열의 지정한 요소에 새로운 값을 할당한다. 배열이 멀티 채널 때는, 런타임 에러가 일어난다. 주석 : 함수 cvSet*D(은)는 싱글 채널과 멀티 채널의 양쪽 모두에 안전하게 사용할 수 있지만, 약간 처리 속도가 늦다.

드문드문한 배열의 경우에, 노드가 존재해 되면, 이 함수는 노드를 생성한다.

mSet

싱글 채널의 부동 소수점형 행렬의 특정의 요소의 값을 변경한다

```
void cvmSet( CvMat* mat, int row, int col, double value );
```

mat

행렬.

row

행의,0(을)를 기준으로 한 인덱스.

col

열의,0(을)를 기준으로 한 인덱스.

value

행렬의 요소가 새로운 값.

함수 cvmSet(은)는, 싱글 채널 부동 소수점형 행렬의 경우에 있어서의,cvSetReal2D 의 고속화판 함수이다. 인 라인 처리되어 배열의 타입이나 요소의 타입의 체크를 실시하지 않고, 또 행 열의 범위의 체크도 debug 모드 때 밖에 실시하지 않기 때문에 고속으로 있다.

ClearND

특정의 요소의 값을 클리어 한다

```
void cvClearND( CvArr* arr, const int* idx );
```

arr

입력 배열.

idx

요소의 인덱스의 배열.

함수 cvClearND(은)는, 조밀한 배열과 드문드문한 배열의 지정한 요소를

2-3 복사와와 채우기(Copying and Filling)

Copy

하나의 배열을 다른 배열에 카피한다

```
void cvCopy( const CvArr* src, CvArr* dst, const CvArr* mask=NULL );
```

src

카피원의 배열.

dst

카피처의 배열.

mask

8 비트 싱글 채널 배열의 처리 마스크.카피처의 배열의 변경하는 요소를 지정한다.

함수 cvCopy (은)는, 입력 배열로부터 출력 배열에 선택된 요소를 카피한다.

mask(I)!=0 의 경우,dst(I)=src(I)

인수의 배열이 IplImage 형태의 경우, 그 ROI (와)과 COI 하지만 이용된다. 카피원배열과 카피처 배열은, 같은 형태, 같은 차원, 같은 사이즈가 아니면 안된다.이 함수는, 드문드문한 배열도 카피할 수 있다(이 경우, 마스크는 서포트되지 않는다).

Set

배열의 각 요소에게 줄 수 있었던 값을 세트 한다

```
void cvSet( CvArr* arr, CvScalar value, const CvArr* mask=NULL );
```

arr

값을 세트 하는 배열.

value

배열을 묻는 값.

mask

8 비트 싱글 채널 배열의 처리 마스크.배열의 변경하는 요소를 지정한다.

함수 cvSet (은)는, 스칼라치 value (을)를, 배열의 선택된 각 요소에 카피한다.

mask(I)!=0 의 경우,arr(I)=value
배열 arr 하지만 IplImage 형태의 경우, ROI (은)는 이용되지만,COI 하지만 세트 되고 있어서는 안 된다.

SetZero

배열을 클리어 한다

```
void cvSetZero( CvArr* arr );
#define cvZero cvSetZero
```

arr

클리어 되는 배열.

함수 cvSetZero (은)는, 배열을 클리어 한다. 조밀한 배열(CvMat, CvMatND, IplImage)에 대한다 cvZero(array) (은)는,cvSet(array,cvScalarAll(0),0) (와)과 등가이다. 드문드문한 배열의 경우는, 모든 요소가 삭제된다.

SetIdentity

스칼라배 된 단위행렬을 이용하고 초기화를 실시한다

```
void cvSetIdentity( CvArr* mat, CvScalar value=cvRealScalar );
```

arr

초기화되는 행렬(정방일 필요는 없다).

value

대각 성분의 값.

함수 cvSetIdentity (은)는, 스칼라배 된 단위행렬을 이용한 초기화를 실시한다.

i=j (이)라면, arr(i,j)=value
그렇지 않으면, 0

Range

주어진 범위의 수로 행렬을 묻는다

```
void cvRange( CvArr* mat, double start, double end );
```

mat

초기화되는 행렬.이것은, 정수형 혹은 부동 소수점형 32 비트 싱글 채널이 아니면 안 된다.

start

범위의 하한(범위에 포함된다).

end

범위의 상한(범위에 포함되지 않는다).

함수 cvRange (은)는, 다음과 같이 행렬을 초기화한다.

arr(i,j)=(end-start)*(i*cols(arr)+j)/(cols(arr)*rows(arr))
예를 들면, 이하의 코드는 연속한 정수의1차원 벡터를 초기화한다.

```
CvMat* A = cvCreateMat( 1, 10, CV_32S );
cvRange( A, 0, A->cols ); // A (은)는,[0,1,2,3,4,5,6,7,8,9] 그리고 초기화된다.
```

2-4 변환과 치환(Transforms and Permutations)

Reshape

데이터의 카피없이 행렬/이미지의 형상을 바꾼다

```
CvMat* cvReshape( const CvArr* arr, CvMat* header, int new_cn, int new_rows=0 );
```

arr

입력 배열.

header

써지는 출력 헤더.

new_cn

새로운 채널수 .new_cn = 0(은)는 채널수가 변경되어 있지 않은 것을 의미한다.

new_rows

새로운 행수. new_rows = 0(은)는, 행수가new_cn의 값에 따라 변경할 필요가 있는데도 관련되지 않고, 변경

되지 않은 채인 것을 의미한다.
함수 cvReshape(은)는, 오리지날의 배열과 같은 데이터이지만, 다른 형상(다른 채널수, 다른 행수, 또 그 양쪽 모두)을 가진다 CvMat 의 헤더를 초기화한다.

예를 들면, 이하는 하나의 이미지 버퍼와 두 개의 이미지의 헤더(1 번째는 320×240×3 의 이미지,2 번째는 960×240x1 의 이미지)을 작성하기 위한 코드이다.

```
IplImage* color_img = cvCreateImage( cvSize(320,240), IPL_DEPTH_8U, 3 );
CvMat gray_mat_hdr;
IplImage gray_img_hdr, *gray_img;
cvReshape( color_img, &gray_mat_hdr, 1 );
gray_img = cvGetImage( &gray_mat_hdr, &gray_img_hdr );
```

그리고, 다음의 예는 3×3 의 행렬로부터 1×9 의 벡터에의 변환이다.

```
CvMat* mat = cvCreateMat( 3, 3, CV_32F );
CvMat row_header, *row;
row = cvReshape( mat, &row_header, 0, 1 );
```

ReshapeMatND

데이터의 카피없이 다차원 배열의 형상을 바꾼다

```
CvArr* cvReshapeMatND( const CvArr* arr,
                       int sizeof_header, CvArr* header,
                       int new_cn, int new_dims, int* new_sizes );

#define cvReshapeND( arr, header, new_cn, new_dims, new_sizes )   ₩
      cvReshapeMatND( (arr), sizeof(*(header)), (header),         ₩
                      (new_cn), (new_dims), (new_sizes))
```

arr
: 입력 배열.

sizeof_header
: IplImage(와)과CvMat,CvMatND각각의 출력 헤더를 구별하기 위한 출력 헤더의 사이즈.

header
: 써지는 출력 헤더.

new_cn
: 새로운 채널수. new_cn = 0(은)는, 채널수가 변경되어 있지 않은 것을 의미한다.

new_dims
: 새로운 차원수. new_dims = 0(은)는, 차원수가 같은 까지 있는 것을 의미한다.

new_sizes
: 새로운 차원 사이즈의 배열.요소의 총수는 변화해선 안 되기 때문에,new_dims-1의 값만 사용된다.따라서,new_dims = 1이면new_sizes(은)는 사용되지 않는다.

함수 cvReshapeMatND(은)는,cvReshape 의 확장 버젼이다.이것은 다차원 배열을 취급하는 것이 가능(보통 이미지와 행렬에 대해서도 사용하는 것이 가능)으로, 한층 더 차원의 변경도

가능하다.이하의 2 개의 샘플은 cvReshape(으)로의 기술을,cvReshapeMatND(을)를 이용해 고쳐 쓴 것이다.

```
IplImage* color_img = cvCreateImage( cvSize(320,240), IPL_DEPTH_8U, 3 );
IplImage gray_img_hdr, *gray_img;
gray_img = (IplImage*)cvReshapeND( color_img, &gray_img_hdr, 1, 0, 0 );

...

/* 2 번째는 2x2x2 의 배열을 8x1 의 벡터로 변환하는 예이다 */
int size[] = { 2, 2, 2 };
CvMatND* mat = cvCreateMatND( 3, size, CV_32F );
CvMat row_header, *row;
row = cvReshapeND( mat, &row_header, 0, 1, 0 );
```

Repeat

출력 배열을 입력 배열로 타일장에 묻는다

```
void cvRepeat( const CvArr* src, CvArr* dst );
```

src

입력 배열, 이미지 또는 행렬.

dst

출력 배열, 이미지 또는 행렬.

함수 cvRepeat(은)는, 입력 배열을 타일장에 배치해 출력 배열을 묻는다.

```
dst(i,j)=src(i mod rows(src), j mod cols(src))
```

출력 배열은 입력 배열보다 큰 일도 있으면, 작은 일도 있다.

Flip

2 차원 배열을 수직, 수평, 또는 양축으로 반전한다

```
void cvFlip( const CvArr* src, CvArr* dst=NULL, int flip_mode=0);
#define cvMirror cvFlip
```

src

입력 배열.

dst

출력 배열.만약dst = NULL이면, 반전은 인프레이스모드로 행해진다.

flip_mode

배열의 반전 방법의 지정.

flip_mode = 0 (은)는,x축주위에서의 반전, flip_mode > 0(예를 들면,1)(은)는,y축주위에서의 반전, flip_mode < 0(예를 들면,-1)(은)는, 양축주위에서의 반전. 이하의 식도 참조.

함수 cvFlip(은)는, 3 종류가 다른 방법중 1 개를 지정해 배열을 반전시킨다(행과 열의 인덱스는 0 하지만 기준이다).

dst(i,j)=src(rows(src)-i-1,j) (flip_mode = 0 의 경우)
dst(i,j)=src(i,cols(src1)-j-1) (flip_mode > 0 의 경우)
dst(i,j)=src(rows(src)-i-1,cols(src)-j-1) (flip_mode < 0 의 경우)
함수의 사용예는 이하대로 :

- 이미지 원점을 좌상으로부터 좌하에, 혹은 그 역방향으로 바꾸어 넣기 위해(Win32 시스템의 동이미지 처리에서는 전형적인 처리)에, 수직 반전(flip_mode > 0)(을)를 이용한다.
- 수직축대칭성을 체크하기 위해서, 이미지의 수평 반전과 수평 방향 시프트를 실시해, 절대 오차를 계산한다(flip_mode > 0).
- 점대칭성을 체크하기 위해서, 이미지의 수평 수직의 동시 반전을 이라고 시프트를 실시해, 절대 오차를 계산한다(flip_mode < 0).
- 1 차원의 배열의 줄을 역전한다(flip_mode > 0).

Split

멀티 채널의 배열을, 복수의 싱글 채널의 배열에 분할한다.또는, 배열로부터 하나의 채널을 꺼낸다.

```
void cvSplit( const CvArr* src, CvArr* dst0, CvArr* dst1,
              CvArr* dst2, CvArr* dst3 );
#define cvCvtPixToPlane cvSplit
```

src

입력 배열.

dst0...dst3

출력 채널.

함수 cvSplit(은)는, 멀티 채널의 배열을 싱글 채널의 배열에 분할한다.이 조작에는 두 개의 모드가 있다. 입력 배열이 N 채널의 경우, 선두로부터 N 번째까지의 출력 채널이 NULL (이)가 아니면, 그것들은 모두 입력 배열로부터 꺼내진다. 그렇지 않고,N 개의 출력 채널중 하나만이 NULL (이)가 아닌 경우는, 이 특정의 채널만을 추출한다.이 머지않아도 아닌 경우는 에러가 된다. N 번째 이후의 출력 채널은 항상 NULL (이)가 아니면 안 된다. IplImage 그럼, 이미지로부터 하나의 싱글 채널을 추출하기 위해서,COI(을)를 따른다 cvCopy 도 이용된다.

Merge

복수의 싱글 채널의 배열로부터 멀티 채널 배열을 구성한다.또는, 배열에 하나의 싱글 채널을 삽입한다

```
void cvMerge( const CvArr* src0, const CvArr* src1,
              const CvArr* src2, const CvArr* src3, CvArr* dst );
#define cvCvtPlaneToPix cvMerge
```

src0... src3

입력 배열.

dst

출력 배열.

함수 cvMerge(은)는, 전술의 함수와 반대의 조작이다.만약 출력 배열이 N 채널로,N 개의 입력 채널이 NULL (이)가 아닐 때, 이러한 모든 것은 출력 배열에 카피된다.그렇지 않고, 하나의 입력 배열만이 NULL (이)가 아니면, 이 특정의 채널이 출력 배열에 카피되어 이 모두가 아닌 경우는 에러가 된다.N 번째 이후의 입력 채널은 항상 NULL (이)가 아니면 안 된다. IplImage 그럼, 이미지에 하나의 싱글 채널을 삽입하기 위해서,COI(을)를 따른다 cvCopy 도 이용된다.

MixChannels

입력 배열의 채널을 출력 배열의 지정된 채널에 카피한다

```
void cvMixChannels( const CvArr** src, int src_count,
                    CvArr** dst, int dst_count,
                    const int* from_to, int pair_count );
```

src

입력 배열의 배열.

src_count

입력 배열의 수.

dst

출력 배열의 배열.

dst_count

출력 배열의 수.

from_to

카피되는 평면(채널)의 인덱스의 페어 배열. from_to[k*2](은)는 입력 평면의0(을)를 기준으로 한 인덱스로,from_to[k*2+1](은)는 출력 평면의 인덱스.여기서, 입력 및 출력 배열 모두에게 대해서, 각 평면에의 연속적인 번호부를 한다.from_to[k*2]하지만 부 때, 대응하는 출력 평면은0그리고 묻힌다.

pair_count

from_to의 페어수, 또는 카피된 평면의 수.

함수 cvMixChannels(은)는,cvSplit,cvMerge, 및 cvCvtColor 의 몇개의 서식의 범용형이다.이것은, 평면의 차례의 변경, 알파 채널의 추가나 삭제,1 매의 평면이나 복수 평면의 추출이나 삽입 등에 이용된다.이하에, 어떻게 4 채널의 RGBA 이미지를,3 채널의 BGR 이미지(즉,R(와)과 B 의 교체)(으)로 하는지, 그리고 알파 채널을 분리하는지를 나타낸다.

```
CvMat* rgba = cvCreateMat( 100, 100, CV_8UC4 );
CvMat* bgr = cvCreateMat( rgba->rows, rgba->cols, CV_8UC3 );
CvMat* alpha = cvCreateMat( rgba->rows, rgba->cols, CV_8UC1 );
CvArr* out[] = { bgr, alpha };
int from_to[] = { 0, 2, 1, 1, 2, 0, 3, 3 };
cvSet( rgba, cvScalar(1,2,3,4) );
cvMixChannels( (const CvArr**)&rgba, 1, out, 2, from_to, 4 );
```

RandShuffle

배열의 요소를 랜덤에 상홀 한다

```
void cvRandShuffle( CvArr* mat, CvRNG* rng, double iter_factor=1. );
```

mat

입력/출력 행렬.인프레이스모드로 상홀 된다.

rng

요소의 상홀로 이용된다 Random Number Generator.포인터가NULL의 경우, 일시적인RNG하지만 생성되어 이용된다.

iter_factor

상홀의 힘을 지정하는 파라미터.이하를 참조.

함수 cvRandShuffle(은)는, 랜덤으로 선택된 배열 요소 페어의 교체를 반복하는 것으로써 행렬을 상홀 한다(멀티 채널 배열의 경우, 각각의 요소는 복수의 성분을 포함한다). 반복 회수(즉, 페어의 교체)는 round(iter_factor*rows(mat)*cols(mat))(이어)여, iter_factor=0(은)는 상홀을 실시하지 않는 것을, iter_factor=1(은)는 이 함수가 rows(mat)*cols(mat)회랜덤인 페어를 바꿔 넣는 것을 의미한다.

2-5 사칙 연산, 논리 연산, 비교 연산(Arithmetic, Logic and Comparison)

LUT

배열의 룩업테이블에 의한 변환

```
void cvLUT( const CvArr* src, CvArr* dst, const CvArr* lut );
```

src

입력 배열(각 요소는8비트 데이터).

dst

출력 배열(임의의 데프스, 입력 배열과 같은 채널수).

lut

요소수가256인 룩업테이블(출력 배열과 같은 데프스가 아니면 안된다). 멀티 채널의 입력/출력 배열의 경우, 테이블은 싱글 채널(이 경우 모든 채널 대하고, 같은 테이블을 사용한다)인가, 입력/출력 배열과 같은 채널수가 아니면 안된다.

함수 cvLUT (은)는, 출력 배열의 각 요소치를 룩업테이블을 이용해서 결정한다. 배열의 인덱스는 입력 배열보다 구할 수 있다. src 의 각 요소에 대해서 이하와 같은 처리를 실시한다.

```
dst(I)=lut[src(I)+DELTA]
```

여기서,src의 데프스가 CV_8U 의 경우는DELTA=0,CV_8S 의 경우는 DELTA=128.

ConvertScale

임의의 선형 변환에 의해서 배열의 값을 변환한다

```
void cvConvertScale( const CvArr* src, CvArr* dst, double scale=1, double shift=0 );

#define cvCvtScale cvConvertScale
#define cvScale  cvConvertScale
#define cvConvert( src, dst )  cvConvertScale( (src), (dst), 1, 0 )
```

src

입력 배열.

dst

출력 배열.

scale

슬캘링 계수.

shift

슬캘링 한 입력 배열의 요소에 가세하는 값.

함수 cvConvertScale (은)는, 여러가지 목적을 가지기 위해, 다른 이름으로의 함수(매크로)라고 해도 정의되고 있다. 이 함수는, 입력 배열을 슬캘링 해 카피해, 한층 더 이하와 같은 변환을 실시한다(혹은 변환만을 실시한다).

dst(I)=src(I)*scale + (shift,shift,...)
멀티 채널 배열의 모든 채널은 독립에 처리된다.

다른 형태에의 변환에서는, 둥근이나 포화를 수반한다.즉, 슬캘링＋변환의 결과가 출력처의 데이터 타입으로 표현할 수 없는 값이 되는 경우에는, 표현할 수 있는 가장 가까운 실축상의 값으로 표현한다.

scale=1, shift=0 의 경우는, 아무것도 변경하지 않는다.이것은 특별한 케이스이며,cvConvert (와)과 같은 의미이다. 입력 배열과 출력 배열이 같은 타입의 경우, 행렬이나 이미지의 슬캘링과 이동을 실시할 수 있어 cvScale 에 상당한다.

ConvertScaleAbs

임의의 선형 변환에 의해서, 입력 배열의 요소를 8 비트 부호 없음 정수형의 배열로 변환한다.

```
void cvConvertScaleAbs( const CvArr* src, CvArr* dst, double scale=1, double shift=0 );
#define cvCvtScaleAbs cvConvertScaleAbs
```

src

입력 배열.

dst

출력 배열(데프스는 8u).

scale

ScaleAbs 계수.

shift

슬캘링 한 입력 배열의 요소에 가세하는 값.

함수 cvConvertScaleAbs (은)는 전술의 함수 와 유사하지만, 이하에 나타내도록(듯이), 변환 결과의 절대치를 출력한다.

dst(I)=abs(src(I)*scale + (shift,shift,...))

이 함수는, 출력 배열의 타입으로서 8u(8 비트 부호 없음 정수)만을 서포트하고 있다.그 외의 형태의 경우는,cvConvertScale (와)과 cvAbs (을)를 조합하는 것으로 같은 효과를 얻을 수 있다.

Add

두 개의 배열을 요소 마다 가산한다

void cvAdd(const CvArr* src1, const CvArr* src2, CvArr* dst, const CvArr* mask=NULL);

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvAdd (은)는, 이하와 같이 src1 의 각 요소에 src2 의 각 요소를 더한다.

dst(I)=src1(I)+src2(I) (mask(I)!=0 의 경우)

모든 배열(마스크를 제외하다)은 같은 타입으로<, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

AddS

스칼라와 배열을 가산한다

void cvAddS(const CvArr* src, CvScalar value, CvArr* dst, const CvArr* mask=NULL);

src

입력 배열.

value

가산하는 스칼라.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvAddS (은)는, 입력 배열 src1 의 모든 요소에 스칼라 value (을)를 더해 결과를 dst 에 보존한다.

dst(I)=src(I)+value (mask(I)!=0 의 경우)
모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

AddWeighted

두 개의 배열의 중량감 화를 계산한다

```
void cvAddWeighted( const CvArr* src1, double alpha,
                    const CvArr* src2, double beta,
                    double gamma, CvArr* dst );
```

src1

1번째의 입력 배열.

alpha

1번째의 배열 요소에의 중량감.

src2

2번째의 입력 배열.

beta

2번째의 배열 요소에의 중량감.

dst

출력 배열.

gamma

가산 결과에, 한층 더 더해지는 스칼라치.

함수 cvAddWeighted (은)는 이하와 같이 두 개의 배열의 중량감 화를 계산한다.

dst(I)=src1(I)*alpha+src2(I)*beta+gamma
모든 배열은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

Sub

두 개의 배열의 요소마다의 감산을 실시한다

```
void cvSub( const CvArr* src1, const CvArr* src2, CvArr* dst, const CvArr* mask=NULL );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvSub (은)는, 이하와 같이 src1 의 각 요소로부터 src2 의 각 요소를 당긴다.

dst(I)=src1(I)-src2(I) (mask(I)!=0 의 경우)
모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

SubS

배열 요소로부터 스칼라를 감산한다

void cvSubS(const CvArr* src, CvScalar value, CvArr* dst, const CvArr* mask=NULL);

src

입력 배열.

value

감산하는 스칼라

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvSubS (은)는, 이하와 같이 입력 배열의 모든 요소로부터 지정된 스칼라를 당긴다.

dst(I)=src(I)-value if mask(I)!=0
모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

SubRS

스칼라로부터 배열 요소를 감산한다

void cvSubRS(const CvArr* src, CvScalar value, CvArr* dst, const CvArr* mask=NULL);

src

입력 배열.

value

감산되는 스칼라.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvSubRS (은)는, 이하와 같이 지정된 스칼라로부터 입력 배열의 모든 요소 각각을 당긴다.

dst(I)=value-src(I) (mask(I)!=0 의 경우)
모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

Mul

두 개의 배열의 요소끼리를 곱셈한다

```
void cvMul( const CvArr* src1, const CvArr* src2, CvArr* dst, double scale=1 );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

scale

임의의 슬캘링 계수.

함수 cvMul (은)는, 이하와 같이 두 개의 배열의 요소끼리의 곱셈을 실시한다.

dst(I)=scale•src1(I)•src2(I)
모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

Div

두 개의 배열의 요소끼리를 제산한다

```
void cvDiv( const CvArr* src1, const CvArr* src2, CvArr* dst, double scale=1 );
```

src1

1번째의 입력 배열.포인터가 NULL 의 경우는, 모든 요소가 1이라고 가정한다.

src2

2번째의 입력 배열.

dst

출력 배열.

scale

옵션의 슬캘링 계수.

함수 cvDiv (은)는, 이하와 같이 1 번째의 입력 배열의 각 요소를 2 번째의 입력 배열의 각 요소로 나눈다.

dst(I)=scale•src1(I)/src2(I), (src1!=NULL 의 경우)
dst(I)=scale/src2(I), (src1=NULL 의 경우)
모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

And

비트 단위의 논리적을 계산한다

```
void cvAnd( const CvArr* src1, const CvArr* src2, CvArr* dst, const CvArr* mask=NULL );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvAnd (은)는 이하와 같이, 두 개의 배열의 요소마다의 논리적(AND)(을)를 계산한다.

dst(I)=src1(I)&src2(I) (mask(I)!=0 의 경우)
부동 소수점형 배열의 경우, 그러한 비트 표현이 처리에 사용된다.모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

AndS

배열의 각 요소와 스칼라와의 비트 단위의 논리적을 계산한다

```
void cvAndS( const CvArr* src, CvScalar value, CvArr* dst, const CvArr* mask=NULL );
```

src

입력 배열.

value

처리에 이용하는 스칼라.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 AndS (은)는 이하와 같이, 배열의 각 요소와 지정된 스칼라의 비트마다의 논리적을 계산한다.

dst(I)=src(I)&value if mask(I)!=0
실제의 계산 전에, 스칼라는 배열과 같은 타입에 변환된다.부동 소수점형 배열의 경우, 그러한 비트 표현이 처리에 사용된다.모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

이하의 샘플 코드는, 최상위비트를 클리어 하는 것으로 부동 소수점형 배열의 각 요소의 절대치를 요구하는 방법을 나타내고 있다.

```
float a[] = { -1, 2, -3, 4, -5, 6, -7, 8, -9 };
CvMat A = cvMat( 3, 3, CV_32F, &a );
int i, abs_mask = 0x7fffffff;
cvAndS( &A, cvRealScalar(*(float*)&abs_mask), &A, 0 );
for( i = 0; i < 9; i++ )
    printf("%.1f ", a[i] );
```

출력은 이하대로이다.

1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0

Or

두 개의 배열 요소의 비트 단위의 논리합을 계산한다

```
void cvOr( const CvArr* src1, const CvArr* src2, CvArr* dst, const CvArr* mask=NULL );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvOr (은)는 이하와 같이, 두 개의 배열의 요소마다의 논리합(OR)(을)를 계산한다.

dst(I)=src1(I)|src2(I)

부동 소수점형 배열의 경우, 그러한 비트 표현이 처리에 사용된다.모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

OrS

배열의 각 요소와 스칼라와의 비트 단위의 논리합을 계산한다

```
void cvOrS( const CvArr* src, CvScalar value, CvArr* dst, const CvArr* mask=NULL );
```

src1

입력 배열.

value

처리에 이용하는 스칼라.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 OrS (은)는 이하와 같이, 배열의 각 요소와 스칼라의 비트마다의 논리합을 계산한다.

dst(I)=src(I)|value (mask(I)!=0 의 경우)
실제의 계산 전에, 스칼라는 배열과 같은 타입에 변환된다.부동 소수점형 배열의 경우, 그러한 비트 표현이 처리에 사용된다.모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

Xor

두 개의 배열 요소의 비트 단위의 배타적 논리합을 계산한다

```
void cvXor( const CvArr* src1, const CvArr* src2, CvArr* dst, const CvArr* mask=NULL );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 cvXor (은)는 이하와 같이, 두 개의 배열의 요소마다의 배타적 논리합(XOR)(을)를 계산한다.

dst(I)=src1(I)^src2(I) (mask(I)!=0 의 경우)
부동 소수점형 배열의 경우, 그러한 비트 표현이 처리에 사용된다.모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

XorS

배열의 각 요소와 스칼라와의 비트 단위의 배타적 논리합을 계산한다

```
void cvXorS( const CvArr* src, CvScalar value, CvArr* dst, const CvArr* mask=NULL );
```

src

입력 배열.

value

처리에 이용하는 스칼라.

dst

출력 배열.

mask

처리 마스크.8비트 싱글 채널 배열(출력 배열의 어느 요소가 변경되는지를 지정한다).

함수 XorS (은)는 이하와 같이, 배열의 각 요소와 스칼라의, 비트마다의 배타적 논리합을 계산한다.

dst(I)=src(I)^value if mask(I)!=0
부동 소수점형 배열의 경우, 그러한 비트 표현이 처리에 사용된다.모든 배열(마스크를 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

이하는, 허부의 최상위비트의 ON-OFF(을)를 바꿔 넣는 것으로 공역복소벡터를 요구하는 샘플 코드이다.

```
float a[] = { 1, 0, 0, 1, -1, 0, 0, -1 }; /* 1, j, -1, -j */
CvMat A = cvMat( 4, 1, CV_32FC2, &a );
int i, neg_mask = 0x80000000;
cvXorS( &A, cvScalar( 0, *(float*)&neg_mask, 0, 0 ), &A, 0 );
for( i = 0; i < 4; i++ )
    printf("(%.1f, %.1f) ", a[i*2], a[i*2+1] );
```

출력은 이하대로.

```
(1.0,0.0) (0.0,-1.0) (-1.0,0.0) (0.0,1.0)
```

Not

각 배열 요소의 비트 단위의 반전을 실시한다

```
void cvNot( const CvArr* src, CvArr* dst );
```

src1

입력 배열.

dst

출력 배열.

함수 Not(은)는 이하와 같이, 배열의 각 요소의 비트를 모두 반전(NOT) 한다.

```
dst(I)=~src(I)
```

Cmp

두 개의 배열의 각 요소마다의 비교를 실시한다

```
void cvCmp( const CvArr* src1, const CvArr* src2, CvArr* dst, int cmp_op );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.어느 쪽의 입력 배열도 싱글 채널이 아니면 안된다.

dst

출력 배열(타입은 8u 인가 8s 그렇지 않으면 안 된다).

cmp_op

비교 방법을 나타내는 플래그.

CV_CMP_EQ - src1(I) (와)과 src2(I) (은)는 동일하다

CV_CMP_GT - src1(I) 하 src2(I) 보다 크다

CV_CMP_GE - src1(I) 하 src2(I) 보다 큰가 동일하다

CV_CMP_LT - src1(I) 하 src2(I) 보다 작다

CV_CMP_LE - src1(I) 하 src2(I) 보다 작은가 동일하다

CV_CMP_NE - src1(I) (와)과 src2(I) (은)는 동일하지 않다

함수 cvCmp (은)는, 이하와 같이 두 개의 배열의 대응하는 요소를 비교해, 출력 배열의 값에 세트 한다.

dst(I)=src1(I) op src2(I),
여기서 op 하 '=', '>', '>=', '<', '<=', '!=' 의 언젠가.

비교 결과가 진(TRUE)이면 dst(I) 에는 0xff(요소 모든 비트가 1)(을)를 세트 해, 그 이외의 경우(FALSE)이면 0 (을)를 세트 한다.모든 배열(출력 배열을 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

CmpS

배열 요소와 스칼라를 비교한다

void cvCmpS(const CvArr* src, double value, CvArr* dst, int cmp_op);

src
: 입력 배열(싱글 채널).

value
: 각각의 배열 요소라고 비교되는 스칼라.

dst
: 출력 배열(타입은8u 또는8s).

cmp_op
: 비교 방법을 나타내는 플래그.
CV_CMP_EQ - src1(I) (와)과 value (은)는 동일하다
CV_CMP_GT - src1(I) 하 value 보다 크다
CV_CMP_GE - src1(I) 하 value 보다 큰가 동일하다
CV_CMP_LT - src1(I) 하 value 보다 작다
CV_CMP_LE - src1(I) 하 value 보다 작은가 동일하다
CV_CMP_NE - src1(I) (와)과 value (은)는 동일하지 않다

함수 cvCmpS (은)는 이하와 같이 배열 요소와 스칼라를 비교해, 출력 배열의 값을 세트 한다.

dst(I)=src(I) op scalar,
여기서 op 하 '=', '>', '>=', '<', '<=' or '!=' 의 언젠가.

비교 결과가 진(TRUE)이면 dst(I) 에는 0xff(요소 모든 비트가 1) (을)를 세트 해, 그 이외의 경우(FALSE)이면 0 (을)를 세트 한다.모든 배열의 사이즈(또는 ROI 의 사이즈)는 같지 않으면 안 된다.

InRange

배열의 요소치가 다른 두 개의 배열 요소로 나타내지는 범위내에 위치하는지를 체크한다

void cvInRange(const CvArr* src, const CvArr* lower, const CvArr* upper, CvArr* dst);

src

입력 배열.

lower

하한치(그 값을 포함한다)를 나타내는 배열.

upper

상한치(그 값은 포함하지 않는다)를 나타내는 배열.

dst

출력 배열(타입은8u 또는 8s).

함수 cvInRange (은)는 이하와 같이, 입력 배열의 모든 요소에 대해 범위 체크를 실시한다.

$dst(I)=lower(I)_0 <= src(I)_0 < upper(I)_0$
(싱글 채널 배열의 경우),

$dst(I)=lower(I)_0 <= src(I)_0 < upper(I)_0 \&\&$
$lower(I)_1 <= src(I)_1 < upper(I)_1$
(2 채널등의 경우).

src(I) 하지만 범위내이면 dst(I) 에는 0xff(요소 모든 비트가 '1')(을)를 세트 해, 그 이외의 경우는 0 (을)를 세트 한다.모든 배열(출력 배열을 제외하다)은 같은 타입으로, 배열의 사이즈(또는 ROI 의 사이즈)도 같지 않으면 안 된다.

InRangeS

배열의 요소치가 두 개의 스칼라의 사이에 위치하는지를 체크한다

```
void cvInRangeS( const CvArr* src, CvScalar lower, CvScalar upper, CvArr* dst );
```

src

입력 배열.

lower

하한치(그 값을 포함한다).

upper

상한치(그 값은 포함하지 않는다).

dst

출력 배열(타입은8u 또는 8s).

함수 cvInRangeS (은)는 이하와 같이, 입력 배열의 모든 요소의 것에 임해서 범위 체크를 실시한다.

$dst(I)=lower_0 <= src(I)_0 < upper_0$
(싱글 채널 배열의 경우),

$dst(I)=lower_0 <= src(I)_0 < upper_0 \&\&$
$lower_1 <= src(I)_1 < upper_1$
(2 채널등의 경우).

src(I) 하지만 범위내이면 dst(I) 에는 0xff(요소 모든 비트가 '1')(을)를 세트 해, 그 이외의 경우는 0 (을)를 세트 한다. 모든 배열의 사이즈(또는 ROI 의 사이즈)는 같지 않으면 안 된다.

Max

두 개의 배열의 각 요소에 대한 최대치를 요구한다

```
void cvMax( const CvArr* src1, const CvArr* src2, CvArr* dst );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

함수 cvMax (은)는 이하와 같이, 두 개의 배열의 요소마다의 최대치를 계산한다.

```
dst(I)=max(src1(I), src2(I))
```

모든 배열은 싱글 채널로, 타입, 배열의 사이즈(또는 ROI 의 사이즈)는 같지 않으면 안 된다.

MaxS

배열의 각 요소와 스칼라에 대한 최대치를 요구한다

```
void cvMaxS( const CvArr* src, double value, CvArr* dst );
```

src

입력 배열.

value

스칼라.

dst

출력 배열.

함수 cvMaxS (은)는 이하와 같이, 배열의 각 요소와 스칼라와의 최대치를 계산한다.

```
dst(I)=max(src(I), value)
```

모든 배열은 싱글 채널로, 타입, 배열의 사이즈(또는 ROI 의 사이즈)는 같지 않으면 안 된다.

Min

두 개의 배열의 각 요소에 대한 최소치를 요구한다

```
void cvMin( const CvArr* src1, const CvArr* src2, CvArr* dst );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

함수 cvMin (은)는 이하와 같이, 두 개의 배열의 요소마다의 최소치를 계산한다.

dst(I)=min(src1(I),src2(I))

모든 배열은 싱글 채널로, 타입, 배열의 사이즈(또는 ROI 의 사이즈)는 같지 않으면 안 된다.

MinS

배열의 각 요소와 스칼라에 대한 최소치를 요구한다

void cvMinS(const CvArr* src, double value, CvArr* dst);

src

입력 배열.

value

스칼라.

dst

출력 배열.

함수 cvMinS (은)는 이하와 같이, 배열의 각 요소와 스칼라와의 최소치를 계산한다.

dst(I)=min(src(I), value)

모든 배열은 싱글 채널로, 타입, 배열의 사이즈(또는 ROI 의 사이즈)는 같지 않으면 안 된다.

AbsDiff

두 개의 배열의 요소마다의 차이의 절대치를 계산한다

void cvAbsDiff(const CvArr* src1, const CvArr* src2, CvArr* dst);

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

출력 배열.

함수 cvAbsDiff (은)는 이하와 같이,src1 의 각 요소와 src2 의 차이의 절대치를 계산한다.

$dst(I)_c = abs(src1(I)_c - src2(I)_c)$.

모든 배열은 같은 타입, 같은 사이즈(또는 같다 ROI 사이즈)가 아니면 안된다.

AbsDiffS

배열의 요소와 정수와의 차이의 절대치를 계산한다

```
void cvAbsDiffS( const CvArr* src, CvArr* dst, CvScalar value );
#define cvAbs(src, dst) cvAbsDiffS(src, dst, cvScalarAll(0))
```

src

입력 배열.

dst

출력 배열.

value

스칼라.

함수 cvAbsDiffS (은)는 이하와 같이, 배열의 각 요소와 스칼라와의 차이의 절대치를 계산한다.

$dst(I)_c = abs(src(I)_c - value_c)$.

모든 배열은 같은 타입, 같은 사이즈(또는 같다 ROI 사이즈)가 아니면 안된다.

2-6 통계(Statistics)

CountNonZero

배열 요소에 대해 0 (이)가 아닌 요소를 카운트 한다

```
int cvCountNonZero( const CvArr* arr );
```

arr

배열(싱글 채널 또는COI하지만 세트 된 멀티 채널의 이미지).

함수 cvCountNonZero (은)는 이하와 같이, 입력 배열내의 0 (이)가 아닌 요소수를 돌려준다.

$result = sum_I arr(I)!=0$

배열이 IplImage 의 경우, ROI,COI 의 양쪽 모두에 대응하고 있다.

Sum

배열 요소의 총화를 계산한다

```
CvScalar cvSum( const CvArr* arr );
```

arr배열.

함수 cvSum (은)는 이하와 같이, 배열 요소의 총화 S (을)를 각 채널로 독립에 계산한다.

$S_c = sum_I arr(I)_c$

배열의 타입이 IplImage 그리고 COI 하지만 세트 되고 있는 경우, 지정된 채널만을 처리해, 총화를 1번째의 스칼라치(S_0)(으)로서 보존한다.

Avg

배열 요소의 평균치를 계산한다

```
CvScalar cvAvg( const CvArr* arr, const CvArr* mask=NULL );
```

arr

배열.

mask

옵션의 처리 마스크.

함수 cvAvg (은)는, 배열 요소의 평균치 M (을)를 각 채널로 독립에 계산한다.

$N = sum_I\ mask(I)!=0$

$M_c = 1/N \bullet sum_{I,mask(I)!=0}\ arr(I)_c$

배열의 타입이 IplImage 그리고 COI 하지만 세트 되고 있는 경우, 지정된 채널만을 처리해, 평균치를 1번째의 스칼라치(M_0)(으)로서 보존한다.

AvgSdv

배열 요소의 평균과 표준 편차를 계산한다

```
void cvAvgSdv( const CvArr* arr, CvScalar* mean, CvScalar* std_dev, const CvArr*
mask=NULL );
```

arr

배열.

mean

계산 결과의 평균치에의 포인터.필요하지 않은 경우는 NULL.

std_dev

계산 결과의 표준 편차에의 포인터.

mask

옵션의 처리 마스크.

함수 cvAvgSdv (은)는, 배열 요소의 평균과 표준 편차를 각 채널로 독립에 계산한다.

$N = sum_I\ mask(I)!=0$

$mean_c = 1/N \bullet sum_{I,mask(I)!=0}\ arr(I)_c$

$std\_dev_c = sqrt(1/N \bullet sum_{I,mask(I)!=0}\ (arr(I)_c - M_c)^2)$

배열의 타입이 IplImage 그리고 COI 하지만 세트 되고 있는 경우, 지정된 채널만을 처리해, 평균치와 표준 편차를 각각 1번째의 스칼라치(M_0(와)과 S_0)(으)로서 보존한다.

MinMaxLoc

배열 혹은 부분 배열내의 최소치와 최대치를 요구한다

```
void cvMinMaxLoc( const CvArr* arr, double* min_val, double* max_val,
                  CvPoint* min_loc=NULL, CvPoint* max_loc=NULL, const CvArr* mask=NULL );
```

arr

입력 배열(싱글 채널 또는COI하지만 세트 된 멀티 채널).

min_val

반환값의 최소치에의 포인터.

max_val

반환값의 최대치에의 포인터.

min_loc

반환값의 최소치를 가지는 위치에의 포인터.

max_loc

반환값의 최대치를 가지는 위치에의 포인터.

mask

부분 배열을 지정하기 위한 옵션의 마스크.

함수 MinMaxLoc (은)는, 배열 요소중에서 최소치·최대치와 그 위치를 요구한다. 각 극값은 배열 전체 또는 선택되었다 ROI(IplImage 의 경우)를 스캔 해 요구한다. mask 하지만 NULL (이)가 아닌 경우는, 지정된 영역만을 스캔 한다. 멀티 채널 배열의 경우, 입력은 COI 하지만 세트 되었다 IplImage 데이터 타입이 아니면 안된다. 다차원 배열의 경우는,min_loc->x(와)과 max_loc->y 에는, 극값의 좌표가 그대로 들어간다.

Norm

배열의 절대치 법칙(absolute array norm), 절대치 차분 법칙(absolute difference norm), 상대 가격차분 법칙(relative difference norm)(을)를 계산한다

```
double cvNorm( const CvArr* arr1, const CvArr* arr2=NULL, int norm_type=CV_L2, const
CvArr* mask=NULL );
```

arr1

1번째의 입력 이미지.

arr2

2번째의 입력 이미지.NULL 의 경우,arr1의 절대치 법칙이 계산되어 그렇지 않은 경우는,arr1-arr2 의 절대치 혹은 상대치 법칙이 계산된다.

normType

법칙의 타입(이하의 설명을 참조).

mask

옵션의 처리 마스크.

함수 cvNorm 하 arr2 하지만 NULL 의 경우, 이하와 같이 arr1 의 절대치 법칙을 계산한다.

$norm = ||arr1||_C = max_I abs(arr1(I))$, (normType = CV_C 의 경우)

$norm = ||arr1||_{L1} = sum_I abs(arr1(I))$, (normType = CV_L1 의 경우)

$norm = ||arr1||_{L2} = sqrt(sum_I arr1(I)^2)$, (normType = CV_L2 의 경우)

이 함수는,arr2 하지만 NULL (이)가 아닌 경우, 이하와 같이 절대치 혹은 상대치 법칙을 계산한다.

$norm = ||arr1-arr2||_C = max_I\ abs(arr1(I)-arr2(I))$, (normType = CV_C 의 경우)

$norm = ||arr1-arr2||_{L1} = sum_I\ abs(arr1(I)-arr2(I))$, (normType = CV_L1 의 경우)

$norm = ||arr1-arr2||_{L2} = sqrt(sum_I\ (arr1(I)-arr2(I))^2)$, (normType = CV_L2 의 경우)

혹은

$norm = ||arr1-arr2||_C/||arr2||_C$, (normType = CV_RELATIVE_C 의 경우)

$norm = ||arr1-arr2||_{L1}/||arr2||_{L1}$, (normType = CV_RELATIVE_L1 의 경우)

$norm = ||arr1-arr2||_{L2}/||arr2||_{L2}$, (normType = CV_RELATIVE_L2 의 경우)

함수 cvNorm (은)는 계산한 법칙을 돌려준다.멀티 채널 배열은 싱글 채널로서 취급한다.즉, 모든 채널의 결과가 통합된다.

Reduce

행렬을 벡터에 축소한다

```
void cvReduce( const CvArr* src, CvArr* dst, int op=CV_REDUCE_SUM );
```

src

입력 행렬.

dst

1행(또는1열)의 출력 벡터(모든 행/열로부터 지정된 방법으로 계산된다).

dim

배열을 어떻게 축소하는지를 나타내는 인덱스. 0 하행열을 1행 벡터에 축소한다. 1 하행열을 1 열에 축소한다. -1 하dst 의 사이즈로부터 차원을 해석해, 자동적으로 선택한다.

op

축소 처리의 종류.이하의 값의 언젠가.

CV_REDUCE_SUM - 출력은 각 행(또는 각 렬)의 총화

CV_REDUCE_AVG - 출력은 각 행(또는 각 렬)의 평균 벡터

CV_REDUCE_MAX - 출력은 각 행(또는 각 렬)에 있어서의 최대치

CV_REDUCE_MIN - 출력은 각 행(또는 각 렬)에 있어서의 최소치

함수 cvReduce (은)는, 행렬의 각 행(각 렬)을 1 차원 벡터의 집합으로서 취급해, 그 집합에 대해서 지시받은 처리를 실시하는 것에 의해서, 입력 행렬을 벡터에 축소한다. 예를 들면, 래스터 이미지의 수평 방향(혹은 수직 방향)에의 투영을 계산하기 위해서 사용할 수 있다.CV_REDUCE_SUM (와)과 CV_REDUCE_AVG 의 경우는, 정도를 유지하기 위해서 출력

요소의 빗트데프스를 크게 취해야 하는 것이다.또, 멀티 채널 배열에 대해도, 이것들 두 개의 모드가 서포트된다.

2-7 선형대수(Linear Algebra)

DotProduct

Euclid 거리에 근거한다 2 개의 배열의 내적을 계산한다

```
double cvDotProduct( const CvArr* src1, const CvArr* src2 );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

함수 cvDotProduct(은)는 Euclid 기하학에 있어서의 내적을 계산해, 그 결과를 돌려준다.

$src1 \bullet src2 = sum_I(src1(I)*src2(I))$
멀티 채널 배열의 경우, 모든 채널의 결과가 누산된다.특히 a(을)를 복소벡터로 하면,cvDotProduct(a,a) 하 $||a||^2$(을)를 돌려준다.이 함수는, 다차원 배열을 1 행,1 층씩 처리할 수도 있다.

Normalize

지정의 법칙이 되도록(듯이), 혹은 값이 지정의 범위가 되도록(듯이), 배열을 정규화한다

```
void cvNormalize( const CvArr* src, CvArr* dst,
                  double a=1, double b=0, int norm_type=CV_L2,
                  const CvArr* mask=NULL );
```

src

입력 배열.

dst

출력 배열.인프레이스 처리가 가능.

a

출력 배열의 최소치 또는 최대치, 혹은 출력 배열의 법칙.

b

출력 배열의 최대치 또는 최소치.

norm_type

정규화의 타입.이하중 하나를 이용할 수 있다.

CV_C - 배열의C-norm(절대치의 최대치)(을)를 정규화

CV_L1 - 배열의L_1-norm(절대치의 합계)(을)를 정규화

CV_L2 - 배열의L_2-norm(Euclid 거리)(을)를 정규화

CV_MINMAX - 배열의 값이 지정의 범위에 들어가도록(듯이) 슬캘링과 시프트를 실시한다.

mask

조작 마스크.특정의 배열 요소만을 정규화하기 위한 마스크.

함수 cvNormalize(은)는, 입력 배열을 그 법칙, 또는 값이 특정의 값이나 범위가 되도록(듯이) 정규화한다.

norm_type==CV_MINMAX 때,

dst(i,j)=(src(i,j)-min(src))*(b'-a')/(max(src)-min(src)) + a', (mask(i,j)!=0 의 경우)

dst(i,j)=src(i,j) (그 이외의 경우)

여기서 b'=MAX(a,b),a'=MIN(a,b).min(src) (와)과 max(src) (은)는 각각 입력 배열의 전체, 또는 지정된 부분 집합에 대해 계산한 전체의 최소치와 최대치이다.

norm_type!=CV_MINMAX 때,

dst(i,j)=src(i,j)*a/cvNorm(src,0,norm_type,mask), (mask(i,j)!=0 의 경우)

dst(i,j)=src(i,j) (그 이외의 경우)

이하에, 간단한 예를 나타낸다.

```
float v[3] = { 1, 2, 3 };
CvMat V = cvMat( 1, 3, CV_32F, v );

// 단위벡터의 생성
// 이것은, 이하와 동일하다.
// for(int i=0;i<3;i++) v[i]/=sqrt(v[0]*v[0]+v[1]*v[1]+v[2]*v[2]);
cvNormalize( &V, &V );
```

CrossProduct

둘의 3 차원 벡터의 외적을 계산한다

```
void cvCrossProduct( const CvArr* src1, const CvArr* src2, CvArr* dst );
```

src1

1번째의 입력 벡터.

src2

2번째의 입력 벡터.

dst

출력 벡터.

함수 cvCrossProduct (은)는, 둘의 3 차원 벡터의 외적을 계산한다.

dst = src1 × src2,

($dst_1 = src1_2src2_3 - src1_3src2_2$, $dst_2 = src1_3src2_1 - src1_1src2_3$, $dst_3 = src1_1src2_2 - src1_2src2_1$).

ScaleAdd

슬캘링 된 배열과 또 하나의 배열의 화를 계산한다

```
void cvScaleAdd( const CvArr* src1, CvScalar scale, const CvArr* src2, CvArr* dst );
#define cvMulAddS cvScaleAdd
```

src1

1번째의 입력 배열

scale

1번째의 배열을 위한 스케일 팩터.

src2

2번째의 입력 배열.

dst

출력 배열.

함수 cvScaleAdd(은)는, 슬캘링 된 배열과 또 하나의 배열의 화를 계산한다.

dst(I)=src1(I)*scale + src2(I)
모든 배열 파라미터는 같은 타입으로 같은 사이즈가 아니면 안 된다.

GEMM

범용적인 행렬의 곱셈을 실시한다

```
void cvGEMM( const CvArr* src1, const CvArr* src2, double alpha,
             const CvArr* src3, double beta, CvArr* dst, int tABC=0 );
#define cvMatMulAdd( src1, src2, src3, dst ) cvGEMM( src1, src2, 1, src3, 1, dst, 0 )
#define cvMatMul( src1, src2, dst ) cvMatMulAdd( src1, src2, 0, dst )
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

alpha

곱셈 결과에 대한 슬캘링 계수.이하의 설명을 참조.

src3

3번째의 입력 배열(시프트용).만약 시프트 하지 않는 경우는NULL(으)로 할 수 있다.

beta

3번째의 입력 배열에 대한 슬캘링 계수.이하의 설명을 참조.

dst

출력 배열.

tABC

조작 플래그.0또는 이하의 값의 편성.

CV_GEMM_A_T - src1(을)를 전치

CV_GEMM_B_T - src2(을)를 전치

CV_GEMM_C_T - src3(을)를 전치

례 : CV_GEMM_A_T+CV_GEMM_C_T(은)는, alpha*src1^T*src2 + beta*src3^{T}에 대응한다.

함수 cvGEMM(은)는 범용적인 행렬의 곱셈을 실시한다.

dst = alpha*op(src1)*op(src2) + beta*op(src3), 여기서, op(X) 하 X 혹은 X^T
모든 행렬은 같은 데이터 타입, 같은 사이즈일 필요가 있다.실수 혹은 복소수의 부동 소수점형의 행렬이 서포트되고 있다.

Transform

모든 배열 요소를 행렬에 의해 변환한다

void cvTransform(const CvArr* src, CvArr* dst, const CvMat* transmat, const CvMat* shiftvec=NULL);

src

1번째의 입력 배열.

dst

출력 배열.

transmat

변환 행렬.

shiftvec

옵션의 시프트 벡터.

함수 cvTransform(은)는,src 의 모두 요소에 행렬 변환을 실시해, 그 결과를 dst 에 보존한다.

dst(I)=transmat*src(I) + shiftvec 또는 $dst(I)_k=sum_j(transmat(k,j)*src(I)_j)$ + shiftvec(k)
즉,Nsrc 개의 요소를 가지는 벡터로서 취급해, M×N 행렬 transmat(와)과 시프트 벡터 shiftvec (을)를 이용하고, M 채널 배열 dst 의 하나의 요소로 변환한다. shiftvec (을)를 transmat 에 묻는다고 하는 선택도 있다. 이 경우,transmat 하 M×N+1 의 행렬이 아니면 안되어, 최우열은 시프트 벡터로서 다루어진다.

입력 배열과 출력 배열은 모두 데프스가 동일하고, 사이즈 혹은 선택되었다 ROI 의 사이즈가 같지 않으면 안 된다.transmat(와)과 shiftvec(은)는 부동 소수점형의 실수 행렬이 아니면 안 된다.

이 함수는,N 차원 점집합의 기하 변환, 임의의 색공간(color space)의 선형 변환, 채널의 상홀등에 사용된다.

PerspectiveTransform

벡터의 투시 투영 변환을 실시한다

void cvPerspectiveTransform(const CvArr* src, CvArr* dst, const CvMat* mat);

src

3채널의 부동 소수점형 입력 배열.

dst

3채널의 부동 소수점형 출력 배열.

mat

3×3 또는 4×4 의 변환 행렬.

함수 cvPerspectiveTransform(은)는, 이하와 같이,src 의 모든 요소를 2 차원 혹은 3 차원 벡터로서 취급해, 이것을 변환한다.

(x, y, z) -> (x' /w, y' /w, z' /w) 또는
(x, y) -> (x' /w, y' /w),

여기서,
(x' , y' , z' , w') = mat4x4*(x, y, z, 1) 또는
(x' , y' , w') = mat3x3*(x, y, 1)

w = w' (w' !=0 의 경우)
    inf (그렇지 않은 경우)

MulTransposed

행렬과 전치행렬의 곱셈을 실시한다

void cvMulTransposed(const CvArr* src, CvArr* dst, int order, const CvArr* delta=NULL);

src

입력 행렬.

dst

출력 행렬.

order

전치 한 행렬을 걸치는 차례.

delta

옵션 배열, 곱셈하기 전에src(으)로부터 끌린다.

함수 cvMulTransposed 하 src(와)과 그 전치행렬과의 곱셈을 계산한다.

이 함수는, 이하와 같이 계산을 실시한다.

$dst=(src-delta)*(src-delta)^T$
(order=0 의 경우)

$dst=(src-delta)^T*(src-delta)$
(그렇지 않은 경우).

Trace

행렬의 트레이스를 돌려준다

CvScalar cvTrace(const CvArr* mat);

mat

입력 행렬.

함수 cvTrace(은)는, 행렬 mat 의 대각 성분의 화를 돌려준다.

tr(src1)=sum$_i$mat(i,i)

Transpose

행렬의 전치를 실시한다

void cvTranspose(const CvArr* src, CvArr* dst);
#define cvT cvTranspose

src

입력 행렬.

dst

출력 행렬.

함수 cvTranspose 하행열 src(을)를 전치 한다.

dst(i,j)=src(j,i)
주석 : 복소행열의 경우, 복소수의 공역화는 실시하지 않는다.공역은 별로 계산 되는 것이 당연하다.예로서 cvXorS 의 샘플 코드를 참조하는 것.

Det

행렬식을 돌려준다

double cvDet(const CvArr* mat);

mat

입력 행렬.

함수 cvDet(은)는, 서방 행렬 mat 의 행렬식을 돌려준다. 작은 행렬에 대해서는 직접법이, 큰 행렬에 대해서는 Gauss 의 소거법이 사용된다. 정의 행렬식을 가지는 대칭 행렬에 대해서는,U=V=NULL(으)로서 SVD(을)를 실행해, 그 후 W 의 대각 요소의 내적으로서 행렬식을 계산하는 것도 가능하다.

Invert

역행열 또는 의사 역행열을 요구한다

double cvInvert(const CvArr* src, CvArr* dst, int method=CV_LU);
#define cvInv cvInvert

src

입력 행렬.

dst

출력 행렬.

method

역행열을 요구하는 수법.

CV_LU - 최적인 비 보트 선택에 의한 Gauss의 소거법

CV_SVD - 특이치 분해(SVD)

CV_SVD_SYM - 대칭정정치 행렬을 위한 특이치 분해

함수 cvInvert 하 src 의 역행열을 계산해, 그 결과를 dst 에 넣는다.

LU 의 경우, 이 함수는 src 의 행렬식을 돌려준다(src(은)는 서방 행렬이 아니어 않은들 없다). 행렬식이 0 의 경우, 역행열은 계산하지 못하고 dst 하 0 그리고 묻힌다.

SVD 의 경우, 이 함수는 src 의 조건수의 역수(큰 특이치에 대한 작은 특이치의 비)를 돌려준다. 또,src 하지만 모두 0 때는 0(을)를 돌려준다. 만약,src 하지만 비마사노리 행렬의 경우,SVD(을)를 이용하는 방법에서는 의사 역행열을 계산한다.

Solve

선형 문제 또는 최소 이승법 문제를 푼다

```
int cvSolve( const CvArr* A, const CvArr* B, CvArr* X, int method=CV_LU );
```

A

입력 행렬.

B

선형 시스템의 우변.

X

출력해.

method

역행열의 해법.

CV_LU - 최적인 비 보트 선택에 의한 Gauss의 소거법

CV_SVD - 특이치 분해

CV_SVD_SYM - 대칭정정치 행렬을 위한 특이치 분해.

함수 cvSolve(은)는, 선형 문제 또는 최소 제곱 문제를 푼다(후자는 특이치 분해를 이용해 푸는 것이 가능하다).

dst = arg $\min_X$||A*X-B||
method 에 CV_LU (을)를 지정했을 경우,A 하지만 마사노리 행렬이면 1 을 돌려주어, 그렇지 않으면 0(을)를 돌려준다. 후자의 경우는,dst 의 값은 유효하지 않다.

SVD

부동 소수점형의 실수 행렬의 특이치 분해를 실시한다

```
void cvSVD( CvArr* A, CvArr* W, CvArr* U=NULL, CvArr* V=NULL, int flags=0 );
```

A

입력M×N행렬.

W

특이치 행렬의 결과 (M×N 또는 N×N)또는 벡터(N×1).

U

임의의 왼쪽 직교 행렬 (M×M 또는 M×N). 만약CV_SVD_U_T하지만 지정되었을 경우, 위에서 말한, 행과 열의 수는 바뀐다.

V

임의의 오른쪽 직교 행렬(N×N).

flags

조작 플래그.0또는 이하의 값의 편성.

- CV_SVD_MODIFY_A (을)를 지정하면, 계산중에 행렬 A 의 변경을 실시할 수 있다. 이 플래그의 지정은 처리 속도를 향상시킨다.
- CV_SVD_U_T means that the tranposed matrix U is returned. Specifying the flag 하 U 의 전치행렬을 돌려주는 것을 의미한다.이 플래그의 지정은 처리 속도를 향상시킨다.
- CV_SVD_V_T 하 V 의 전치행렬을 돌려주는 것을 의미한다.이 플래그의 지정은 처리 속도를 향상시킨다.

함수 cvSVD(은)는, 행렬 A(을)를 두 개의 직교 행렬과 하나의 대일본 장기 말열의 적 으로 분해한다.

$A=U*W*V^T$

여기서,W(은)는 특이치의 대일본 장기 말열이며, 특이치의 1 차원의 벡터로서 encode 된다.모든 특이치는 비부의 값을 가져, 내림차순에 소트 된다 (U(와)과 V 의 행도 모두).

SVD 알고리즘은 수치적으로 로바스트이며, 그 전형적인 응용은, 이하와 같이 된다.

- 행렬 A 하지만 서방 행렬로 대칭정정치 행렬로서 정의되고 있는(예를 들면, 그것이 공변동 행렬이다)라고 해 엄밀한 고유치 문제의 해법이다. 이 케이스에 대해 W(은)는 고유치의 벡터가 되어,U=V(은)는 고유 벡터의 행렬이 된다. (따라서, 고유 벡터가 필요한 경우는,U 또는 V 의 다른 한쪽만을 계산하면 좋다).
- 방정식보다 미지수가 많은 선형 문제(과소 시스템)에 있어서의 엄밀한 해법.
- 미지수보다 방정식이 많은 선형 문제(과다 시스템)에 있어서의 최소 이승법에 따르는 해법. 이것과 전술의 경우에는,method 에 CV_SVD(을)를 지정한 함수 cvSolve 하지만 실행된다.
- 랭크(0 이 아닌 특이치의 수), 조건수(최대의 특이치와 최소의 특이치와의 비율), 행렬식(특이치의 적에 동일한 행렬식의 절대치)등의, 여러가지 행렬 특징량의 고정밀의 계산.여기에 리스트 된 모든 항목은 U 행렬과 V 행렬의 계산을 필요로 하지 않는다.

SVBkSb

특이치의 후퇴 대입을 실시한다

```
void cvSVBkSb( const CvArr* W, const CvArr* U, const CvArr* V,
```

```
                const CvArr* B, CvArr* X, int flags );
```

W

특이치의 행렬 또는 벡터.

U

왼쪽 직교 행렬(전치 되고 있을지도 모르다)

V

오른쪽 직교 행렬(전치 되고 있을지도 모르다)

B

행렬A의 의사 역행열에 타기 위한 행렬.옵션의 파라미터. 생략 되고 있는 경우, 그것은 적절한 사이즈의 단위행렬이라고 가정된다(그 때문에X하A의 재구성된 의사 역행열이 된다).

X

출력 행렬.후퇴 대입의 결과.

flags

조작 플래그.cvSVD(으)로의flags(와)과 일치하고 있지 않으면 안 된다.

함수 cvSVBkSb(은)는, 분해되는 행렬 A(cvSVD(을)를 참조)와 행렬 B(을)를 위한 후퇴 대입을 계산한다.

$X=V*W^{-1}*U^{T}*B$
여기서,

$W^{-1}(i,i)=1/W(i,i)$ (W(i,i) > epsilon•sum_iW(i,i) 의 경우)
0 (그 이외의 경우)
또,epsilon 하행열의 데이터 타입에 의존하는 작은 수치이다.

이 함수는cvSVD(와)과 함께,cvInvert,cvSolve의 내부에서 이용된다.이것들(svd (와)과 bksb) 의 저레벨 함수를 사용하는 이유는, 고레벨 함수(inv (와)과 solve)에 두어 일시적인 행렬의 확보를 피하기 때문에 있다.

EigenVV

대칭 행렬의 고유치와 고유 벡터를 계산한다

```
void cvEigenVV( CvArr* mat, CvArr* evects, CvArr* evals, double eps=0 );
```

mat

입력 대칭 서방 행렬.처리중으로 변경된다.

evects

고유 벡터의 출력 행렬.연속한 행으로서 보존된다.

evals

고유치 벡터의 출력 벡터.내림차순에 보존된다(물론 고유치와 고유 벡터의 차례는 일치한다).

eps

대각화의 정도(일반적으로,DBL_EPSILON=≈10^{-15} 그리고 충분하다).

함수 cvEigenVV(은)는, 행렬 mat 의 고유치와 고유 벡터를 계산한다.

mat*evects(i,:)' = evals(i)*evects(i,:)' (MTALAB 표기)
행렬 mat 의 내용은 이 함수에 의해서 파괴된다.

현재, 이 함수는 cvSVD 보다 늦고, 정도도 낮다. 거기서,mat 하지만 정정치 행렬인 것이 기존 때(예를 들면 공변동 행렬)은, 행렬 mat 의 고유치, 고유 벡터를 요구하기 위해서(특히, 고유 벡터가 필요하게 되지 않는 경우), cvSVD(을)를 사용하는 것이 추천 된다. 이것은,

```
cvEigenVV(mat, eigenvals, eigenvects);
```
대신에
```
cvSVD(mat, eigenvals, eigenvects, 0, CV_SVD_U_T + CV_SVD_MODIFY_A);
```
(을)를 부르는 것이다.

CalcCovarMatrix

벡터 집합의 공변동 행렬을 계산한다

```
void cvCalcCovarMatrix( const CvArr** vects, int count, CvArr* cov_mat, CvArr* avg, int flags );
```

vects

입력 벡터.이것들은 모두 같은 타입으로 같은 사이즈가 아니면 안 된다. 벡터는1차원일 필요는 없고,2차원(예를 들면, 이미지)등에서도 상관없다.

count

입력 벡터의 수.

cov_mat

부동 소수점형의 정방인 출력 공변동 행렬.

avg

입력 또는 출력 배열(플래그에 의존한다) - 입력 벡터의 평균 벡터.

flags

조작 플래그.이하의 값의 편성.

CV_COVAR_SCRAMBLED - 출력 공변동 행렬은 다음과 같이 계산된다.

$scale*[vects[0]-avg,vects[1]-avg,...]^T*[vects[0]-avg,vects[1]-avg,...]$

즉, 공변동 행렬은count×count이다. 그러한 일반적이지 않은 공변동 행렬은, 매우 큰 벡터 집합에 대한 고속의 주성분 분석을 위해서 사용된다(예를 들면, 얼굴 인식을 위한 고유얼굴). 이 「스크럼블 되었다」행렬의 고유치는, 진공변동 행렬의 고유치와 일치해, 그리고 「진정한」고유 벡터는 「스크럼블 되었다」공변동 행렬의 고유 벡터로부터 용이하게 계산할 수 있다.

CV_COVAR_NORMAL - 출력 공변동 행렬은 다음과 같이 계산된다.

$scale*[vects[0]-avg,vects[1]-avg,...]*[vects[0]-avg,vects[1]-avg,...]^T$

즉,cov_mat(은)는 모든 입력 벡터의 요소의 합계와 같은 사이즈의 일반적인 공변동 행렬이 된다. CV_COVAR_SCRAMBLED(와)과CV_COVAR_NORMAL의 어느 쪽인지 하나는 반드시 지정되지 않으면 안 된다.

CV_COVAR_USE_AVG - 이 플래그가 지정되었을 경우, 함수는 입력 벡터로부터 평균을 계산하지 않고, 인수로 지정된 평균 벡터를 사용한다. 평균이 어떠한 방법으로 이미 계산되고 있는 경우, 또는 공변동 행렬이 부분적으로 계산되고 있는 경우 (이 경우,avg(은)는 입력 벡터의 일부의 평균이 아니고, 모든 평균 벡터이다)에 유용하다.

CV_COVAR_SCALE - 이 플래그가 지정되었을 경우, 공변동 행렬은 입력 벡터의 수에 의해서 슬캘링 된다.

CV_COVAR_ROWS - 모든 입력 벡터는 단일의 행렬(vects[0])의 행으로서 보존되는 것을 의미한다. 이 경우,count(은)는 무시된다.그리고avg(은)는 적절한 크기의1행의 벡터가 아니면 안된다.

CV_COVAR_COLS - 모든 입력 벡터는 단일의 행렬(vects[0])의 열로서 보존되는 것을 의미한다. 이 경우,count 하무시된다.그리고avg(은)는 적절한 크기의1열의 벡터가 아니면 안된다.

함수 cvCalcCovarMatrix(은)는, 공변동 행렬과 옵션으로 입력 벡터 집합의 평균을 계산한다. 이 함수는, 주성분 분석이나 마하라노비스 거리에 의한 벡터의 비교등에 이용된다.

Mahalanobis

두 개의 벡터의 마하라노비스 거리를 계산한다

```
double cvMahalanobis( const CvArr* vec1, const CvArr* vec2, CvArr* mat );
```

vec1

1번째의1차원 입력 벡터.

vec2

2번째의1차원 입력 벡터.

mat

역공변동 행렬.

함수 cvMahalanobis(은)는, 두 개의 벡터간의 중량감 거리를 계산해, 그것을 돌려준다.

$d(vec1,vec2)=sqrt(sum_{i,j} {mat(i,j)*(vec1(i)-vec2(i))*(vec1(j)-vec2(j))})$

cvCalcCovarMatrix(을)를 이용해 공변동 행렬이 계산되어 한층 더 cvInvert 함수(행렬이 비마사노리 행렬일 가능성이 있으므로,method 에 CV_SVD 하지만 선택된 것)를 이용해 역행열을 요구한다.

CalcPCA

벡터 집합의 주성분 분석을 실시한다

```
void cvCalcPCA( const CvArr* data, CvArr* avg,
                CvArr* eigenvalues, CvArr* eigenvectors, int flags );
```

data

입력 데이터.각각의 벡터는 단일행(CV_PCA_DATA_AS_ROW)인가, 단일열(CV_PCA_DATA_AS_COL)이다.

avg

평균 벡터.함수내에서 계산되는지, 유저에 의해서 주어진다.

eigenvalues

출력인 공변동 행렬의 고유치.

eigenvectors

출력인 공변동 행렬의 고유 벡터(즉, 주성분).하나의 행이 하나의 벡터를 의미한다.

flags

조작 플래그.이하의 값의 편성.

CV_PCA_DATA_AS_ROW - 행으로서 벡터가 보존된다(즉, 어느 벡터의 모든 요소는 연속적으로 보존된다)

CV_PCA_DATA_AS_COL - 열로서 벡터가 보존된다(즉, 어느 벡터 성분에 속하는 값은 연속적으로 보존된다)

(위의 2개의 플래그는 서로 배타적이다)

CV_PCA_USE_AVG - 사전에 계산된 평균 벡터를 이용한다

함수 cvCalcPCA(은)는, 벡터 집합의 주성분 분석을 실시한다. 우선,cvCalcCovarMatrix(을)를 이용해 공변동 행렬을 계산해, 고유치, 고유 벡터를 요구한다. 고유치, 고유 벡터의 출력 개수는,MIN(rows(data),cols(data))(와)과 동일한지, 또는 적다.

ProjectPCA

지정된 부분 공간에 벡터를 투영 한다

```
void cvProjectPCA( const CvArr* data, const CvArr* avg,
                   const CvArr* eigenvectors, CvArr* result );
```

data

입력 데이터.각각의 벡터는 단일행인가, 단일열이다.

avg

평균 벡터.단일행 벡터의 경우, 그것은data의 행으로서 입력 벡터가 보존되고 있는 것을 의미한다. 그렇지 않은 경우는, 단일 열이며, 그 때의 벡터는data의 열로서 보존되고 있다.

eigenvectors

고유 벡터(주성분).하나의 행이 하나의 벡터를 의미한다.

result

출력인 분해 계수의 행렬.행의 수는 벡터의 수와 같지 않으면 안 된다. 열의 수는eigenvectors의 열의 수보다 작은가 같지 않으면 안 된다. 열의 수가 적은 경우, 입력 벡터는, 제cols(result)주성분까지를 기저로 하는 부분 공간에 투영 된다.

함수 cvProjectPCA(은)는, 입력 벡터를 정규 직교 기저(eigenvectors)(으)로 표현되는 부분 공간에 투영 한다. 내적을 계산하기 전에,avg 벡터가 입력 벡터로부터 감산된다.

```
result(i,:)=(data(i,:)-avg)*eigenvectors' // CV_PCA_DATA_AS_ROW 배치의 경우
```

BackProjectPCA

투영 계수로부터 원래의 벡터를 재구축 한다

```
void cvBackProjectPCA( const CvArr* proj, const CvArr* avg,
                       const CvArr* eigenvects, CvArr* result );
```

proj

입력 데이터.cvProjectPCA의result(와)과 같은 포맷.

avg

평균 벡터.만약 단일행 벡터의 경우, 출력 벡터가result의 행으로서 보존되고 있는 것을 의미한다. 그렇지 않은 경우는, 단일 열이며, 그 때는result의 열로서 보존된다.

eigenvectors

고유 벡터(주성분).하나의 행이 하나의 벡터를 의미한다.

result

출력인 재구축 된 벡터의 행렬.

함수 cvBackProjectPCA(은)는, 투영 계수로부터 벡터를 재구축 한다.

result(i,:)=proj(i,:)*eigenvectors + avg // CV_PCA_DATA_AS_ROW 배치의 경우

2-8 수학 함수(Math Functions)

Round, Floor, Ceil

부동 소수점형의 변수를 정수형으로 변환한다

```
int cvRound( double value );
int cvFloor( double value );
int cvCeil( double value );
```

value

부동 소수점형의 입력치.

함수 cvRound, cvFloor,cvCeil(은)는, 부동 소수점형의 입력치를 둥근 수법(Round, Floor, Ceil)의 하나를 이용해 정수형으로 변환한다.cvRound(은)는, 인수에 가장 가까운 정수치를 돌려준다.cvFloor(은)는,인수보다 크지 않은 최대의 정수치를 돌려준다.cvCeil(은)는, 인수보다 작지 않은 최소의 정수치를 돌려준다.몇개의 아키텍쳐에 대하고, 이러한 함수는 C 언어의 표준적인 캐스트 연산보다*비상*에 고속으로 있다. 인수의 절대치가 2^{31} 보다 큰 경우, 결과는 구해지지 않는다.또, 특별한 값(±Inf, NaN)(은)는 취급할 수 없다.

Sqrt

평방근을 계산한다

```
float cvSqrt( float value );
```

value

부동 소수점형의 입력치.

함수 cvSqrt(은)는, 인수의 평방근을 계산한다. 인수가 부의 값의 경우, 결과는 구해지지 않는다.

InvSqrt

평방근의 역수를 계산한다

```
float cvInvSqrt( float value );
```

value

부동 소수점형의 입력치.

함수 cvInvSqrt(은)는, 인수의 평방근의 역수를 계산한다.이것은, 통상,1./sqrt(value)(을)를 계산한다보다 고속으로 있다. 인수가 0 또는 부의 값 때, 결과는 구해지지 않는다.또, 특별한 값(±Inf, NaN)(은)는 취급할 수 없다.

Cbrt

입방근을 계산한다

```
float cvCbrt( float value );
```

value

부동 소수점형의 입력치.

함수 cvCbrt(은)는, 인수의 입방근을 계산한다.이것은, 통상,pow(value,1./3)(을)를 계산하는 것보다도 고속으로 있다. 한편, 부의 인수도 올바르게 처리된다.또, 특별한 값(±Inf, NaN)(은)는 취급할 수 없다.

FastArctan

2 차원의 벡터의 각도를 계산한다

```
float cvFastArctan( float y, float x );
```

x

2차원 벡터의x좌표.

y

2차원 벡터의y좌표.

함수 cvFastArctan(은)는 입력되었다 2 차원 벡터의 각도를 계산한다. 각도는 번(degree) 단위로 다루어져 0°(으)로부터 360°의 범위에서 변화한다.정도는 ~0.1°.

IsNaN

인수가 수치가 아닌지 어떤지를 확인한다

```
int cvIsNaN( double value );
```

value

부동 소수점형의 입력치.

함수 cvIsNaN(은)는, 인수가 수치(IEEE754 standard 에 정의되고 있다)가 아니면 1(을)를 돌려주어, 그 외의 경우는 0(을)를 돌려준다.

IsInf

인수가 무한대일지를 확인한다

```
int cvIsInf( double value );
```

value

부동 소수점형의 입력치.

함수 cvIsInf(은)는, 인수가±무한대(IEEE754 standard 에 정의되고 있다)이면 1(을)를 돌려주어, 그 외의 경우는 0(을)를 돌려준다.

CartToPolar

2 차원 벡터의 각도와 크기를 계산한다

```
void cvCartToPolar( const CvArr* x, const CvArr* y, CvArr* magnitude,
                    CvArr* angle=NULL, int angle_in_degrees=0 );
```

x

x좌표의 배열.

y

y좌표의 배열.

magnitude

크기의 출력 배열.필요하지 않으면NULL하지만 세트 된다.

angle

각도의 출력 배열.필요하지 않으면NULL하지만 세트 된다.각도는 라디안(0..2π), 또는 번(0..360°)(으)로 측정된다.

angle_in_degrees

각도를 나타내기 위해서 라디안(디폴트치), 또는 번의 어느 쪽을 이용하는지를 나타내는 플래그.

함수 cvCartToPolar(은)는,2 차원 벡터(x(I),y(I))의 크기와 각도의 어느 쪽인지, 또는 그 양쪽 모두를 계산한다.

$magnitude(I)=sqrt(x(I)^2+y(I)^2)$,
$angle(I)=atan(y(I)/x(I))$
각도는≈0.1°의 정도로 계산된다.(0,0)의 경우, 각도는 0 에 세트 된다.

PolarToCart

극좌표 형식에서 표현되었다 2 차원 벡터의 데카르트 좌표를 계산한다

```
void cvPolarToCart( const CvArr* magnitude, const CvArr* angle,
                    CvArr* x, CvArr* y, int angle_in_degrees=0 );
```

magnitude

크기의 배열.NULL의 경우, 크기는 모두1(와)과 가정된다.

angle

각도의 배열.단위는 라디안, 또는 번이다.

x

x좌표의 출력 배열로, 필요하지 않으면NULL하지만 세트 된다.

y

y좌표의 출력 배열로, 필요하지 않으면NULL하지만 세트 된다.

angle_in_degrees

이 플래그는 각도를 나타내기 위해서, 라디안(디폴트치) 또는 번의 어느 쪽을 이용하는지를 나타낸다.
함수 cvPolarToCart(은)는, 모든 벡터의 x 좌표와 y 좌표의 어느 쪽인지, 또 그 양쪽 모두의 magnitude(I)*exp(angle(I)*j), j=sqrt(-1)(을)를 계산한다.

x(I)=magnitude(I)*cos(angle(I)),
y(I)=magnitude(I)*sin(angle(I))

Pow

모든 배열 요소를 누승한다

void cvPow(const CvArr* src, CvArr* dst, double power);

src

입력 배열.

dst

출력 배열.입력과 같은 타입이 아니면 안된다.

power

누승의 지수.

함수 cvPow(은)는, 입력 배열의 모든 요소를 이하와 같이 p 승 한다.

$dst(I)=src(I)^p$, (p 하지만 정수의 경우),
$dst(I)=abs(src(I))^p$, (그 이외의 경우)
누승의 지수가 정수가 아닌 경우는, 입력 배열 요소의 절대치가 이용된다. 그러나, 몇개의 추가 처리에 의해서 부의 값에 대해서도 올바른 값을 얻을 수 있다.이하에, 배열 요소의 입방근을 계산하는 예를 나타낸다.

```
CvSize size = cvGetSize(src);
CvMat* mask = cvCreateMat( size.height, size.width, CV_8UC1 );
cvCmpS( src, 0, mask, CV_CMP_LT ); /* 부의 요소를 검출 */
cvPow( src, dst, 1./3 );
cvSubRS( dst, cvScalarAll(0), dst, mask ); /* 부의 입력에 대한 결과를 반전한다 */
cvReleaseMat( &mask );
```

정수나 0.5,-0.5 등 일부의 power 의 값에 대해서는, 특히 고속의 알고리즘이 이용된다.

Exp

모든 배열 요소에 대해 자연대수의 바닥(네이피아수)e 말해 나무승을 요구한다

void cvExp(const CvArr* src, CvArr* dst);

src

입력 배열.

dst

출력 배열.배정도의 부동 소수점형(double), 또는 입력 배열과 같은 타입이 아니면 안된다.

함수 cvExp 하입력 배열의 모든 요소에 대해서, 그것을 지수로 하는 자연대수의 바닥 e 말해 나무승을 요구한다.

dst(I)=exp(src(I))
최대 상대오차는≈7e-6.현재, 이 함수는, 지수 표현되지 않는다(denormalize) 값을 출력시에 0(으)로 변환한다.

Log

모든 배열 요소의 절대치의 자연대수를 계산한다

```
void cvLog( const CvArr* src, CvArr* dst );
```

src

입력 배열.

dst

출력 배열.배정도의 부동 소수점형(double), 또는 입력 배열과 같은 타입이 아니면 안된다
함수 cvLog(은)는, 입력 배열의 모든 요소의 절대치의 자연대수를 계산한다.

dst(I)=log(abs(src(I))), (src(I)!=0 의 경우)
dst(I)=C, (src(I)=0 의 경우)
여기서,C (은)는 큰 부의 수이다 (현재의 실장에서는≈-700).

SolveCubic

3 다음 방정식의 무로네를 요구한다

```
int cvSolveCubic( const CvMat* coeffs, CvMat* roots );
```

coeffs

식의 계수로,3개 또는4개의 요소를 가지는 배열.

roots

무로네의 출력 배열.세 개의 요소를 가진다.
함수 cvSolveCubic(은)는,3 다음 방정식의 무로네를 요구한다.

$coeffs[0]*x^3 + coeffs[1]*x^2 + coeffs[2]*x + coeffs[3] = 0$
(coeffs 하지만 4 요소의 벡터의 경우)

or

$x^3 + coeffs[0]*x^2 + coeffs[1]*x + coeffs[2] = 0$
(coeffs 하지만 3 요소의 벡터의 경우)
이 함수는 요구한 무로네의 수를 돌려주어, 배열 roots 에 뿌리를 격납한다. 하나의 뿌리 밖에 없는 경우, 출력 배열은에는 0 하지만 추가된다.

2-9 난수 생성(Random Number Generation)

RNG

난수 생성기 상태를 초기화한다

```
CvRNG cvRNG( int64 seed=-1 );
```

seed

랜덤시퀀스를 개시하기 위해서 사용된다64비트의 수치.

함수 cvRNG(은)는 난수 생성기를 초기화해, 그 상태를 돌려준다. 상태에의 포인터는 함수 cvRandInt,cvRandReal,cvRandArr 에게 건네진다.현재의 실장에서는,multiply-with-carry RNG 하지만 이용되고 있다.

RandArr

배열을 난수로 묻어 RNG 상태를 갱신한다

```
void cvRandArr( CvRNG* rng, CvArr* arr, int dist_type, CvScalar param1, CvScalar
param2 );
```

rng

cvRNG에 의해서 초기화되었다RNG상태.

arr

출력 배열.

dist_type

분포의 타입.

CV_RAND_UNI - 일 모양 분포

CV_RAND_NORMAL - 정규 분포(Gauss 분포)

param1

분포의 제일 파라미터.일 모양 분포에서는, 발생하는 난수의 하한치(이 값을 포함한다)이다. 정규 분포에서는, 난수의 평균치이다.

param2

분포의 제2 파라미터.일 모양 분포에서는, 발생하는 난수의 상한치(이 값은 포함하지 않는다)이다. 정규 분포에서는, 난수의 표준 편차이다.

함수 cvRandArr(은)는, 일 모양 또는 정규 분포의 난수로 출력 배열을 묻는다. 정규 분포의 부동 소수점형의 값을 2 차원 배열중의 랜덤인 위치에 가산하는 예를 이하에 나타낸다.

```
/* noisy_screen(을)를, 「엄청」부동 소수점형의 2 차원 배열로 한다 */
CvRNG rng_state = cvRNG(0xffffffff);
int i, pointCount = 1000;
/* 점의 좌표의 배열을 확보한다 */
CvMat* locations = cvCreateMat( pointCount, 1, CV_32SC2 );
/* 랜덤인 점의 값의 배열 */
```

```
CvMat* values = cvCreateMat( pointCount, 1, CV_32FC1 );
CvSize size = cvGetSize( noisy_screen );

cvRandInit( &rng_state,
            0, 1, /* 현재는 더미 파라미터를 사용해, 한층 더 그것들을 조정한다 */
            0xffffffff /* 여기에서는, 고정된 종을 사용한다 */,
            CV_RAND_UNI /* 일 모양 분포의 지정 */ );

/* 위치의 초기화 */
cvRandArr( &rng_state, locations, CV_RAND_UNI,
           cvScalar(0,0,0,0), cvScalar(size.width,size.height,0,0) );

/* 정규 분포치를 생성한다 RNG(을)를 만들기 위해서 RNG(을)를 수정한다 */
rng_state.disttype = CV_RAND_NORMAL;
cvRandSetRange( &rng_state,
                30 /* 편차 */,
                100 /* 점의 밝기의 평균 */,
                -1 /* 모든 차원을 초기화한다 */ );
/* 값을 생성 */
cvRandArr( &rng_state, values, CV_RAND_NORMAL,
           cvRealScalar(100), // average intensity
           cvRealScalar(30) // deviation of the intensity
           );

/* 점을 세트 한다 */
for( i = 0; i < pointCount; i++ )
{
    CvPoint pt = *(CvPoint*)cvPtr1D( locations, i, 0 );
    float value = *(float*)cvPtr1D( values, i, 0 );
    *((float*)cvPtr2D( noisy_screen, pt.y, pt.x, 0 )) += value;
}

/* 일시적인 배열을 해방하는 것을 잊지 않게 */
cvReleaseMat( &locations );
cvReleaseMat( &values );

/* RNG 상태는 해방의 필요는 없다 */
```

RandInt

32 비트 부호 없음 정수를 돌려주어,RNG(을)를 갱신한다

```
unsigned cvRandInt( CvRNG* rng );
```

rng

RandInit에 의해서 초기화되어 옵션으로RandSetRange에 의해서 커스터마이즈 되었다RNG상태(후자의 함수는, 이 함수의 결과에 영향을 미치지 않는다).

함수 cvRandInt(은)는, 일 모양 분포했다 32 비트 부호 없음 정수형의 난수를 돌려주어,RNG 상태를 갱신한다. 이것은,C 언어의 런 타임 라이브러리로 말할 곳의 rand() 함수에 유사하고 있다. 그러나,rand()하지만 0(으)로부터 RAND_MAX(2**16 또는 2**32, 플랫폼에 의존한다)까지의 수를 돌려주는 것 대하고, 이 함수에서는 항상 32 비트의 수를 생성한다. 이 함수는 점좌표나 팟치사이즈, 테이블 인덱스등의 스칼라의 난수를 생성하는데 도움이 된다. 여기서, 어느 범위의 정수는 잉여 연산에 의해서 생성되어 부동 소수점형의 수는, 지정 범위를 0..1 에 슬캘링 하는 것으로 생성할 수 있다. cvRandInt(을)를 이용하고, 전술의 예를 고쳐 쓴 것을 이하에 나타낸다.

```
/* 입력과 태스크는 전의 샘플과 같다. */
CvRNG rng_state = cvRNG(0xffffffff);
int i, pointCount = 1000;
/* ... - 여기서 배열은 확보되지 않는다 */
CvSize size = cvGetSize( noisy_screen );
/* 오바헷드를 줄이기 위해, 정규 분포 한 수치를 위한 버퍼를 생성한다 */
#define bufferSize 16
float normalValueBuffer[bufferSize];
CvMat normalValueMat = cvMat( bufferSize, 1, CV_32F, normalValueBuffer );
int valuesLeft = 0;

for( i = 0; i < pointCount; i++ )
{
    CvPoint pt;
    /* 랜덤인 점의 생성 */
    pt.x = cvRandInt( &rng_state ) % size.width;
    pt.y = cvRandInt( &rng_state ) % size.height;

    if( valuesLeft <= 0 )
    {
        /* 버퍼가 하늘의 경우, 정규 분포 한 수치로 버퍼를 묻는다 */
        cvRandArr( &rng_state, &normalValueMat,
                   CV_RAND_NORMAL, cvRealScalar(100), cvRealScalar(30) );
        valuesLeft = bufferSize;
    }
    *((float*)cvPtr2D( noisy_screen, pt.y, pt.x, 0 ) = normalValueBuffer[--valuesLeft];
}

/* 행렬의 헤더와 데이터는 스택에 가지고 있기 위해,normalValueMat(을)를 해방할 필요는 없다.
   이것은 작은 고정 사이즈의 행렬로 실행할 때에, 잘 알려져 있는 효과적인 방법이다*/
```

RandReal

부동 소수점형의 난수를 돌려주어,RNG(을)를 갱신한다

```
double cvRandReal( CvRNG* rng );
```

rng

cvRNG에 의해서 초기화된,RNG상태.

함수 cvRandReal(은)는,0(으)로부터 1(1(은)는 포함되지 않는다)의 범위에 일 모양 분포하는 부동 소수점형의 난수를 돌려준다.

2-10 이산 변환(Discrete Transforms)

DFT

1 차원 혹은 2 차원 부동 소수점형 배열에 대해서 이산 푸리에 변환(DFT), 역이산 푸리에 변환(IDFT)(을)를 실시한다

```
#define CV_DXT_FORWARD  0
#define CV_DXT_INVERSE  1
#define CV_DXT_SCALE    2
#define CV_DXT_ROWS     4
#define CV_DXT_INV_SCALE (CV_DXT_SCALE|CV_DXT_INVERSE)
#define CV_DXT_INVERSE_SCALE CV_DXT_INV_SCALE

void cvDFT( const CvArr* src, CvArr* dst, int flags, int nonzero_rows=0 );
```

src

입력 배열(실수 또는 복소수).

dst

입력 배열과 같은 사이즈·타입의 출력 배열.

flags

변환 플래그.이하의 값의 편성.

CV_DXT_FORWARD - 1차원 또는2차원의 순서 변환을 실시한다.결과의 슬캘링은 실시하지 않는다.

CV_DXT_INVERSE - 1차원 또는2차원의 역변환을 실시한다.결과의 슬캘링은 실시하지 않는다. CV_DXT_FORWARD (와)과 CV_DXT_INVERSE (은)는, 물론 동시에는 지정할 수 없다.

CV_DXT_SCALE - 결과를 배열 요소수로 나누어, 슬캘링 한다.통상은 CV_DXT_INVERSE 그와 동시에 이용한다. 쇼트 컷으로서 CV_DXT_INV_SCALE (을)를 이용해도 좋다.

CV_DXT_ROWS - 입력 배열의 각각의 행에 대해서 독립에, 순서 변환 혹은 역변환을 실시한다. 이 플래그는 복수의 벡터의 동시 변환을 허가해, 오버헤드(하나의 계산의 몇배도 커지기도 한다)를 줄이기 위해나,3차원 이상의 고차원에 대해서 변환을 실시하기 위해서 사용된다.

nonzero_rows

입력 배열의 비0인 행의 수(2차원순서 변환의 경우), 혹은 출력 배열로 주목하는 행의 수(2차원역변환의 경우). 이 값이 부,0, 혹은 행의 수보다 큰 경우는 무시된다. 이 파라미터에 의해,DFT(을)를 이용해2차원의 다다미 포함이나 상관 연산을 실시할 때의 계산 속도가 향상한다.자세한 것은, 이하의 샘플을 참조.

함수 cvDFT (은)는, 이하에 나타내도록(듯이)1 차원 혹은 2 차원 부동 소수점형 배열의 순서 변환·역변환을 실시한다.

N 개의 요소를 가진다 1 차원 벡터의 푸리에 변환 :

$y = F^{(N)} \cdot x$, 여기서, $F^{(N)}_{jk}=\exp(-i \cdot 2Pi \cdot j \cdot k/N)$, $i=\sqrt{-1}$

N 개의 요소를 가진다 1 차원 벡터의 역푸리에 변환 :
$x' = (F^{(N)})^{-1} \cdot y = conj(F^{(N)}) \cdot y$
$x = (1/N) \cdot x$

M × N 개의 요소를 가진다 2 차원 벡터의 푸리에 변환 :
$Y = F^{(M)} \cdot X \cdot F^{(N)}$

M × N 개의 요소를 가진다 2 차원 벡터의 역푸리에 변환 :
$X' = conj(F^{(M)}) \cdot Y \cdot conj(F^{(N)})$
$X = (1/(M \cdot N)) \cdot X'$
IPL 그리고 이용되는, 실수(싱글 채널)packed 포맷의 데이터가, 푸리에 변환의 결과 혹은 역푸리에 변환의 입력을 표현하기 위해서 이용된다.

$Re\ Y_{0,0}$	$Re\ Y_{0,1}$	$Im\ Y_{0,1}$	$Re\ Y_{0,2}$	$Im\ Y_{0,2}$	...	$Re\ Y_{0,N/2-1}$	$Im\ Y_{0,N/2-1}$	$Re\ Y_{0,N/2}$
$Re\ Y_{1,0}$	$Re\ Y_{1,1}$	$Im\ Y_{1,1}$	$Re\ Y_{1,2}$	$Im\ Y_{1,2}$	...	$Re\ Y_{1,N/2-1}$	$Im\ Y_{1,N/2-1}$	$Re\ Y_{1,N/2}$
$Im\ Y_{1,0}$	$Re\ Y_{2,1}$	$Im\ Y_{2,1}$	$Re\ Y_{2,2}$	$Im\ Y_{2,2}$	...	$Re\ Y_{2,N/2-1}$	$Im\ Y_{2,N/2-1}$	$Im\ Y_{2,N/2}$
...								
$Re\ Y_{M/2-1,0}$	$Re\ Y_{M-3,1}$	$Im\ Y_{M-3,1}$	$Re\ Y_{M-3,2}$	$Im\ Y_{M-3,2}$	...	$Re\ Y_{M-3,N/2-1}$	$Im\ Y_{M-3,N/2-1}$	$Re\ Y_{M-3,N/2}$
$Im\ Y_{M/2-1,0}$	$Re\ Y_{M-2,1}$	$Im\ Y_{M-2,1}$	$Re\ Y_{M-2,2}$	$Im\ Y_{M-2,2}$	...	$Re\ Y_{M-2,N/2-1}$	$Im\ Y_{M-2,N/2-1}$	$Im\ Y_{M-2,N/2}$
$Re\ Y_{M/2,0}$	$Re\ Y_{M-1,1}$	$Im\ Y_{M-1,1}$	$Re\ Y_{M-1,2}$	$Im\ Y_{M-1,2}$	...	$Re\ Y_{M-1,N/2-1}$	$Im\ Y_{M-1,N/2-1}$	$Im\ Y_{M-1,N/2}$

주석 : N 하지만 짝수라면 최종열이 존재해,M 하지만 짝수라면 맨 마지막 줄이 존재한다.

1 차원 실수 변환의 경우, 결과는 위의 행렬의 1 행목과 같이 된다.

DFT(을)를 이용했다2차원 다다미 포함의 계산

```
CvMat* A = cvCreateMat( M1, N1, CV_32F );
CvMat* B = cvCreateMat( M2, N2, A->type );

// 간직해 결과의 일부(abs(M2-M1)+1×abs(N2-N1)+1)만을 가질 가능성도 있다
CvMat* conv = cvCreateMat( A->rows + B->rows - 1, A->cols + B->cols - 1, A->type );

// A(와)과 B(을)를 초기화
...

int dft_M = cvGetOptimalDFTSize( A->rows + B->rows - 1 );
int dft_N = cvGetOptimalDFTSize( A->cols + B->cols - 1 );

CvMat* dft_A = cvCreateMat( dft_M, dft_N, A->type );
CvMat* dft_B = cvCreateMat( dft_M, dft_N, B->type );
CvMat tmp;

// A(을)를 dft_A 에 카피해,dft_A 의 우부 나머지를 0 그리고 묻는다
cvGetSubRect( dft_A, &tmp, cvRect(0,0,A->cols,A->rows));
cvCopy( A, &tmp );
cvGetSubRect( dft_A, &tmp, cvRect(A->cols,0,dft_A->cols - A->cols,A->rows));
```

```
   cvZero( &tmp );
   // 이하의 cvDFT()그럼, 파라미터 nonzero_rows(을)를 이용하고 있기 때문에,
   // dft_A 의 하부를 0 그리고 묻을 필요는 없다

   cvDFT( dft_A, dft_A, CV_DXT_FORWARD, A->rows );

   // 2 번째의 배열에 대해서도 이와 같이 반복한다
   cvGetSubRect( dft_B, &tmp, cvRect(0,0,B->cols,B->rows));
   cvCopy( B, &tmp );
   cvGetSubRect( dft_B, &tmp, cvRect(B->cols,0,dft_B->cols - B->cols,B->rows));
   cvZero( &tmp );
   // 이하의 cvDFT()그럼, 파라미터 nonzero_rows(을)를 이용하고 있기 때문에,
   // dft_B 의 하부를 0 그리고 묻을 필요는 없다

   cvDFT( dft_B, dft_B, CV_DXT_FORWBRD, B->rows );

   cvMulSpectrums( dft_A, dft_B, dft_A, 0 /* 다다미 포함은 아니고 상관을 얻기 위해서는
                                                CV_DXT_MUL_CONJ (을)를 지정한다
*/ );

   cvDFT( dft_A, dft_A, CV_DXT_INV_SCALE, conv->rows ); // 상부만을 계산한다
   cvGetSubRect( dft_A, &tmp, cvRect(0,0,conv->cols,conv->rows) );

   cvCopy( &tmp, conv );
```

GetOptimalDFTSize

주어진 벡터의 사이즈에 대한 최적인 DFT 의 사이즈를 돌려준다

```
int cvGetOptimalDFTSize( int size0 );
```

size0

벡터의 사이즈.

함수 cvGetOptimalDFTSize (은)는,size0 이상의 최소치 N (을)를 돌려주어, 얻을 수 있던 사이즈 N 의 벡터에 대한다 DFT(은)는 고속으로 계산할 수 있다. 현재의 실장에서는, 있다 p, q, r 에 대해서,$N=2^p \times 3^q \times 5^r$.

size0 하지만 매우 큰 값(INT_MAX 에 매우 가깝다)의 경우, 이 함수는 부의 값을 돌려준다.

MulSpectrums

두 개의 푸리에 스펙트럼의 요소마다의 곱셈을 실시한다

```
void cvMulSpectrums( const CvArr* src1, const CvArr* src2, CvArr* dst, int flags );
```

src1

1번째의 입력 배열.

src2

2번째의 입력 배열.

dst

입력 배열과 같은 타입·사이즈의 출력 배열.

flags

이하의 값의 편성.

CV_DXT_ROWS - 배열의 각 행을 개별의 스펙트럼으로서 취급한다 (cvDFT 의 파라미터를 참조).

CV_DXT_MUL_CONJ - 곱셈 전에2번째의 입력 배열의 공역을 계산한다

함수 cvMulSpectrums (은)는, 실푸리에 변환 혹은 복소푸리에 변환의 결과로서 얻을 수 있던 둘의 CCS-packed 행렬, 또는 복소행열의 요소마다의 곱셈을 실시한다.

이 함수와 cvDFT(을)를 모두 이용하는 것으로, 두 개의 배열의 고속의 간직해 계산이 가능하다.

DCT

1 차원 혹은 2 차원 부동 소수점형 배열의 순서 방향·역방향 이산 코사인 변환을 실시한다

```
#define CV_DXT_FORWARD  0
#define CV_DXT_INVERSE  1
#define CV_DXT_ROWS     4

void cvDCT( const CvArr* src, CvArr* dst, int flags );
```

src

입력 배열(실수의1차원 혹은2차원 배열).

dst

입력과 같은 사이즈·타입의 출력 배열.

flags

변환 플래그.이하의 값의 편성.

CV_DXT_FORWARD - 1차원 혹은2차원의 순서 변환.

CV_DXT_INVERSE - 1차원 혹은2차원의 역변환.

CV_DXT_ROWS - 입력 배열의 각각의 행에 대해서, 독립에 순서 변환 혹은 역변환을 실시한다. 이 플래그는 복수 벡터의 동시 변환을 허가해, 오버헤드(하나의 계산의 몇배도 커지기도 한다)를 줄이기 위해나,3차원 이상의 고차원에 대해서 변환을 실시하기 위해서 사용된다.

함수 cvDCT (은)는 이하와 같이,1 차원 혹은 2 차원 부동 소수점형 배열에 대해서 순서 방향·역방향 이산 코사인 변환을 실시한다.

N 개의 요소를 가진다 1 차원 벡터의 순서 방향 코사인 변환 :
$y = C^{(N)} \cdot x$, 여기서 $C^{(N)}_{jk} = \mathrm{sqrt}((j==0?1:2)/N) \cdot \cos(Pi \cdot (2k+1) \cdot j/N)$

N 개의 요소를 가진다 1 차원 벡터의 역방향 코사인 변환 :
$x = (C^{(N)})^{-1} \cdot y = (C^{(N)})^{T} \cdot y$

M×N 개의 요소를 가진다 2 차원 벡터의 순서 방향 코사인 변환 :

$Y = (C^{(M)}) \cdot X \cdot (C^{(N)})^T$

M×N 개의 요소를 가진다 2 차원 벡터의 역방향 코사인 변환 :
$X = (C^{(M)})^T \cdot Y \cdot C^{(N)}$

3.동적 구조체(Dynamic Structures)

3-1 메모리 저장(Memory Storages)

CvMemStorage

동적으로 확장 가능한 메모리스트레이지

```
typedef struct CvMemStorage
{
    struct CvMemBlock* bottom;/* 최초로 확보된 블록 */
    struct CvMemBlock* top; /* 새로운 블록을 확보하는 장소 */
    struct CvMemStorage* parent; /* 현재의 메모리브록크 - 스택의 선두 */
    int block_size; /* 블록의 크기 */
    int free_space; /* top 블록내의 자유 영역(아르바이트 단위) */
} CvMemStorage;
```

메모리스트레이지는 저레벨의 구조체로, 동적으로 확장 가능한 데이터 구조를 가져, 순서나 윤곽, 그래프, 세분화 등에 사용된다. 이것은 같은 사이즈의 메모리브록크의 리스트로서 편성되고 있다. bottom 필드는 블록의 리스트의 선두,top(은)는 현재 사용되고 있는 블록을 의미하지만, 리스트의 마지막 블록은 반드시 필요하지 않다.후자(여기에서는 top) 이외의 bottom(으)로부터 top 까지의 블록은 모두 점유 되고 있다. top(을)를 제외한다 top(으)로부터 끝까지의 모든 블록은 비어 영역에서,top 블록 자체는 부분적으로 점유 되고 있다. free_space(은)는,top 의 뒤로 남아 있는 아르바이트 단위로 나타난 자유 영역을 의미한다.

함수 cvMemStorageAlloc 그리고 명시적으로, 혹은 cvSeqPush(이)나 cvGraphAddEdge 등의 고레벨 함수에 의해서 간접적으로 확보된 새로운 메모리 영역은, 거기에 들어가는 경우에는*항상*현재의 블록의 마지막에 확보된다. 확보 후 free_space(은)는, 적절한 얼라이먼트를 보관 유지하기 위해서, 확보한 밧파사이즈에 패딩을 더한 만큼 두개 줄여진다. 확보된 버퍼가 top 의 이용 가능한 영역에 들어가지 않는 경우, 리스트의 다음의 스토리지 블록이 top(으)로서 이용되어 free_space(은)는 확보전의 전브록크사이즈에 리셋트 된다.

만약 비어 블록이 없는 경우, 새로운 블록이 확보되어 (또는 부모로부터 빌리고,cvCreateChildMemStorage(을)를 참조), 리스트의 마지막에 추가된다.이와 같이 이 스토리지는 스택으로서 행동해, bottom 하지만 스택의 바닥, top(와)과 free_space 의 페어가 스택의 선두를 나타낸다. 스택의 선두는 cvSaveMemStoragePos 그리고 보존, cvRestoreMemStoragePos 그리고 복원, cvClearMemStorage 그리고 리셋트 된다.

CvMemBlock

메모리스트레이지브록크

```
typedef struct CvMemBlock
{
    struct CvMemBlock* prev;
    struct CvMemBlock* next;
} CvMemBlock;
typedef struct CvMemBlock
{
    struct CvMemBlock* prev;
    struct CvMemBlock* next;
} CvMemBlock;
```

구조체 CvMemBlock(은)는, 메모리스트레이지의 1 블록을 표현한다. 실제의 메모리브록크의 데이터는 헤더의 뒤에 있어, 메모리브록크의 i 번째의 아르바이트는((char*)(mem_block_ptr+1))[i]그리고 꺼낼 수 있다. 그러나 통상, 직접 액세스 할 필요는 없다.

CvMemStoragePos

메모리스트레이지의 위치

```
typedef struct CvMemStoragePos
{
    CvMemBlock* top;
    int free_space;
} CvMemStoragePos;
```

이 구조체는 스택의 선두 위치를 보관 유지하고 있어, cvSaveMemStoragePos 그리고 보존, cvRestoreMemStoragePos 그리고 복원할 수 있다.

CreateMemStorage

메모리스트레이지를 생성한다

```
CvMemStorage* cvCreateMemStorage( int block_size=0 );
```

block_size

스토리지 블록의 아르바이트 단위의 사이즈.0의 경우, 디폴트치(현재는≈64K)(이)가 사용된다.

함수 cvCreateMemStorage(은)는 메모리스트레이지를 생성해, 그 포인터를 돌려준다.초기 상태에서는 스토리지는 하늘이다. block_size(을)를 제외하는 헤더의 필드는 모두 0(으)로 설정되어 있다.

CreateChildMemStorage

아이 메모리스트레이지를 생성한다

CvMemStorage* cvCreateChildMemStorage(CvMemStorage* parent);

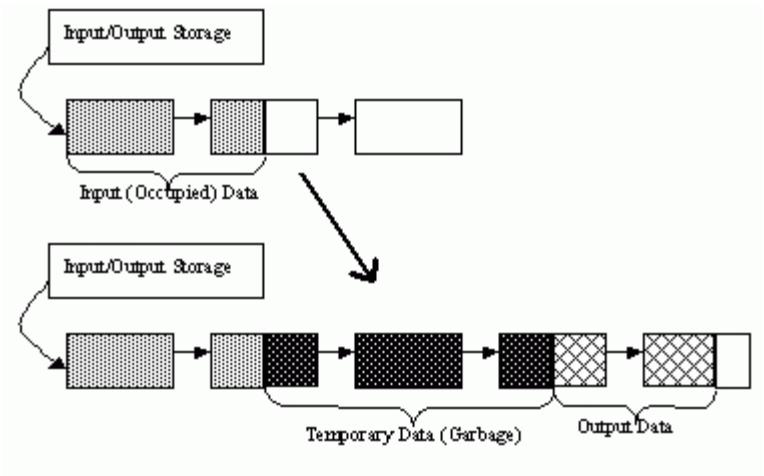
parent

친메모리스트레이지.

함수 cvCreateChildMemStorage(은)는, 메모리 확보/해방기구의 차이를 제외하고, 단순한 메모리스트레이지를 닮은 아이 메모리스트레이지를 생성한다. 아이 스토리지가 새로운 블록을 블록 리스트에 추가할 필요가 있는 경우, 아이는 부모로부터 블록을 꺼내는 것을 시도한다. 부모에게 점유 되어 있지 않은 이용 가능한 최초의 블록이 꺼내져 그 부분은 부모의 블록 리스트로부터 배제된다. 이용 가능한 블록이 없는 경우, 만약 부모의 부모가 존재하면, 부모는 거기로부터 블록을 확보할까 빌린다. 즉, 각각의 스토리지가 다른 스토리지의 부모나 아이가 되는, 최인이나 한층 더 복잡한 구조의 메모리스트레이지가 가능하게 된다. 아이 스토리지가 해방될까 클리어 되었을 경우, 그 모든 블록은 부모에게 돌려주어진다.다른 견해를 하면, 아이 스토리지는 단순한 스토리지와 같다.

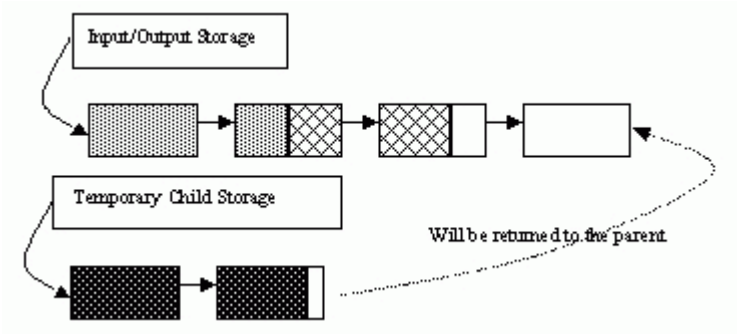
아이 스토리지는 이하의 상황에 있어 유용하다.몇개의 스토리지로 나누어져 존재하는 동적 데이터를 처리하고, 그 결과를 같은 스토리지에 둘 필요가 있는 상황을 상상해 주었으면 한다. 가장 단순하게, 일시적인 데이터를 입출력 데이터와 같은 스토리지에 두는 방법에서는, 처리 후의 스토리지는 이하와 같이 된다.

아이 스토리지를 이용하지 않고 동적인 데이터 처리를 실시했을 경우



이와 같이 스토리지의 중앙에 쓰레기가 남아 버린다. 그러나 처리의 시작해에 아이 메모리스트레이지를 생성하고, 일시적인 데이터를 거기에 써, 마지막에 해방했을 경우, 쓰레기는 입력/출력 스토리지에 남지 않는다.

아이 스토리지를 이용해 동적인 데이터 처리를 실시했을 경우



ReleaseMemStorage

메모리스트레이지를 해방한다

```
void cvReleaseMemStorage( CvMemStorage** storage );
```

storage

해방하는 스토리지의 포인터.

함수 cvReleaseMemStorage(은)는, 스토리지의 전메모리브록크를 해방해, 만약 부모가 있으면 부모에게 돌려준다. 그 때에 스토리지 헤더도 해방되어 스토리지에의 포인터도 클리어 된다. 모든 아이 스토리지는 부모를 해방하기 전에 해방해 둘 필요가 있다.

ClearMemStorage

메모리스트레이지를 클리어 한다

```
void cvClearMemStorage( CvMemStorage* storage );
```

storage

메모리스트레이지.

함수 cvClearMemStorage(은)는, 스토리지의 선두(빈영역의 경계)를, 제일 최초로 되돌린다. 이 함수는 메모리를 해방하지 않는다.만약 스토리지가 부모를 가지는 경우, 이 함수는 부모에게 모든 블록을 돌려준다.

MemStorageAlloc

스토리지내에 메모리밧파를 확보한다

```
void* cvMemStorageAlloc( CvMemStorage* storage, size_t size );
```

storage

메모리스트레이지.

size

밧파사이즈.

함수 cvMemStorageAlloc(은)는, 스토리지내에 메모리밧파를 확보한다. 밧파사이즈는, 스토리지의 브록크사이즈를 넘어선 안 된다. 넘었을 경우에는 런타임 에러가 발생한다. 버퍼 주소는,CV_STRUCT_ALIGN(현시점에서는=sizeof(double)) 아르바이트에 얼라이먼트 된다.

MemStorageAllocString

스토리지내에 텍스트 문자열을 확보한다

```
typedef struct CvString
{
    int len;
    char* ptr;
}
CvString;

CvString cvMemStorageAllocString( CvMemStorage* storage, const char* ptr, int len=-1 );
```

storage

메모리스트레이지.

ptr

문자열.

len

문자열의 길이(종단의'₩0'(은)는 세지 않는다).부의 경우, 이 함수가 길이를 계산한다.

함수 cvMemStorageAllocString(은)는, 메모리스트레이지내에 문자열의 카피를 생성한다. 이 함수는 유저가 건네준, 혹은 계산된 문자열의 길이와 카피된 문자열에의 포인터를 가지는 구조체를 돌려준다.

SaveMemStoragePos

메모리스트레이지의 위치를 보존한다

```
void cvSaveMemStoragePos( const CvMemStorage* storage, CvMemStoragePos* pos );
```

storage

메모리스트레이지.

pos

출력하는 스토리지 선두의 위치.

함수 cvSaveMemStoragePos(은)는, 스토리지의 선두의 현재 위치를 파라미터 pos 에 보존한다. 함수 cvRestoreMemStoragePos 그리고, 이 정도치를 복원할 수 있다.

RestoreMemStoragePos

메모리스트레이지의 위치를 복원한다

```
void cvRestoreMemStoragePos( CvMemStorage* storage, CvMemStoragePos* pos );
```

storage

메모리스트레이지.

pos

새로운 스토리지의 선두 위치.

함수 cvRestoreMemStoragePos(은)는, 파라미터 pos(으)로부터 스토리지의 선두를 복원한다. 이 함수와 함수 cvClearMemStorage(은)는, 메모리브록크내에서 점유 된 메모리를 해방하는 유일한 방법이다. 스토리지의 점유 부분의 도중의 메모리를 해방하는 방법은 없는 것에 충분히 주의하는 것.

3-2 순서(Sequences)

CvSeq

확장 가능한 요소의 순서

```
#define CV_SEQUENCE_FIELDS() ₩
    int flags; /* 여러가지 플래그 */ ₩
    int header_size; /* 순서의 헷다사이즈 */ ₩
    struct CvSeq* h_prev; /* 하나전의 순서에의 포인터 */ ₩
    struct CvSeq* h_next; /* 하나 후의 순서에의 포인터 */ ₩
    struct CvSeq* v_prev; /* 하나전의 순서에의 포인터(세칸다리, 구조에 따라서 의미가
다르다) */ ₩
    struct CvSeq* v_next; /* 하나 후의 순서에의 포인터(세칸다리, 구조에 따라서 의미가
다르다) */ ₩
    int total; /* 요소의 총수 */ ₩
    int elem_size;/* 순서 요소의 사이즈(아르바이트 단위) */ ₩
    char* block_max;/* 최신의 블록의 최대치 */ ₩
    char* ptr; /* 현재의 기입 포인터 */ ₩
    int delta_elems; /* 순서를 확장시킬 때에, 파티션 하는 요소수(순서의 입도) */ ₩
    CvMemStorage* storage; /* seq 하지만 보존되는 영역 */ ₩
    CvSeqBlock* free_blocks; /* 빈블록 리스트 */ ₩
    CvSeqBlock* first; /* 선두 순서 블록에의 포인터 */


typedef struct CvSeq
{
    CV_SEQUENCE_FIELDS()
} CvSeq;
```

구조체 CvSeq (은)는,OpenCV 의 동적 데이터 구조 모든 기본이 되는 것이다.

상기의 보조 매크로를 개입시킨 특수한 정의에 의해, 추가 파라미터를 수반하는 구조체 CvSeq 의 확장이 용이하게 실시할 수 있다. CvSeq (을)를 확장하기 위해서, 새로운 구조체를 정의해, 매크로 CV_SEQUENCE_FIELDS()에 의해서 열거된다 CvSeq 의 필드의 뒤에 유저 정의 필드를 두어도 괜찮다.

순서에는, 조밀한 순서와 드문드문한 순서의 2 종류가 존재한다. 조밀한 순서의 기본 타입은 CvSeq (이어)여, 이 순서는 확장 가능한 1 차원 배열 (벡터(vectors), 스택(stacks), 큐(queues), DEC(deques))를 표현하는데 이용된다. 이러한 타입에는 데이터의 중간 부분에 공백부가 없다.즉, 순서의 중간 부분에 있어서의 요소의 삭제나 추가 시에는, 가장 가까운 종단의 요소로부터 시프트 된다. 드문드문한 순서는,CvSet(을)를 그 기본 클래스로서 가진다.자세한 것은 후술 한다. 그것들은, 노드 플래그에 의해서 「데이터가 있다」인가 「빈 곳」가가 나타나고 있는 노드 순서이다. 이 타입의 순서는, 요소의 집합(sets), 그래프(graph), 해시 테이블(hash tables) 등의 순서가 없는 데이터 구조에 이용된다.

필드 header_size (은)는, 순서 헤더의 실사이즈가 들어가 있어 그 사이즈는,sizeof(CvSeq)보다 큰가 동일하고안 된다.

필드 h_prev, h_next, v_prev, v_next (은)는, 다른 순서군으로부터 계층 구조를 생성하기 위해서 사용할 수 있다. 필드 h_prev (와)과 h_next (은)는, 동일 계층에서의 전후의 순서를 가리켜, 게다가 필드 v_prev (와)과 v_next (은)는 세로 방향으로의 전후의 순서(자신의 부모와 최초의 아이)를 가리킨다. 그러나, 이것들은 단지 이름에 지나지 않고, 이러한 포인터를 다른 의미로 사용하는 것이 가능하다.

필드 first (은)는, 블록의 선두 순서를 가리킨다.이 구조에 대해서는 이하로 말한다.

필드 total (은)는, 조밀한 순서에서는 실요소수, 드문드문한 순서에서는 파티션 된 노드수를 나타낸다.

필드 flags (은)는, 상위 16 비트로 개개의 동적 특성 (조밀한 순서에서는 CV_SEQ_MAGIC_VAL , 드문드문한 순서에서는 CV_SET_MAGIC_VAL)(와)과 순서에 관한 그 외의 잡다한 정보를 가진다. 하위 CV_SEQ_ELTYPE_BITS 비트는, 요소 타입의 ID(을)를 나타낸다. 대부분의 순서 처리 함수는 요소 타입이 아니고,elem_size 에 보존된 요소 사이즈를 사용한다. 순서가 CvMat 타입의 하나인 수치 데이터로부터 되는 경우, 순서의 요소 타입은 대응한다 CvMat 의 요소 타입과 일치한다 (예를 들면,2 차원의 점데이터의 순서로는 CV_32SC2 하지만, 부동 소수점형의 순서에는 CV_32FC1 하지만 이용되는 등). 매크로 CV_SEQ_ELTYPE(seq_header_ptr) (은)는, 순서의 요소의 타입을 꺼낸다. 수치 순서를 취급하는 함수에서는 elem_size 하지만, 그 요소 타입으로부터 계산된 것과 동일한지 어떤지가 체크된다. 게다가 CvMat 에 준거한 타입에서는, 헤더 cvtypes.h 그리고 정의된 이하의 추가 요소 타입이 존재한다 :

표준적인 순서 요소의 종류

```
    #define CV_SEQ_ELTYPE_POINT          CV_32SC2  /* (x,y) */
    #define CV_SEQ_ELTYPE_CODE           CV_8UC1   /* 후리만코드: 0..7 */
    #define CV_SEQ_ELTYPE_GENERIC        0 /* 일반적인 순서 요소 타입 */
    #define CV_SEQ_ELTYPE_PTR            CV_USRTYPE1 /* =6 */
    #define CV_SEQ_ELTYPE_PPOINT         CV_SEQ_ELTYPE_PTR  /* &elem: 다른 순서 요소에의
포인터 */
    #define CV_SEQ_ELTYPE_INDEX          CV_32SC1  /* #elem: 다른 순서 요소의 인덱스 */
    #define CV_SEQ_ELTYPE_GRAPH_EDGE     CV_SEQ_ELTYPE_GENERIC  /* &next_o, &next_d,
&vtx_o, &vtx_d */
    #define CV_SEQ_ELTYPE_GRAPH_VERTEX   CV_SEQ_ELTYPE_GENERIC  /* 선두의 옆, &(x,y) */
```

```
    #define CV_SEQ_ELTYPE_TRIAN_ATR      CV_SEQ_ELTYPE_GENERIC  /* 2 분목의 정점(노드)
*/
    #define CV_SEQ_ELTYPE_CONNECTED_COMP CV_SEQ_ELTYPE_GENERIC  /* 접속 성분  */
    #define CV_SEQ_ELTYPE_POINT3D        CV_32FC3  /* (x,y,z)  */
```
다음의,CV_SEQ_KIND_BITS 비트는, 순서의 종류를 지정한다.

표준적인 순서의 종류

```
    /* 일반적인(특별한 지정 없음의) 순서의 종류 */
    #define CV_SEQ_KIND_GENERIC     (0 << CV_SEQ_ELTYPE_BITS)

    /* 조밀한 순서의 아류형 */
    #define CV_SEQ_KIND_CURVE       (1 << CV_SEQ_ELTYPE_BITS)
    #define CV_SEQ_KIND_BIN_TREE    (2 << CV_SEQ_ELTYPE_BITS)

    /* 드문드문한 순서(또는 집합)의 아류형 */
    #define CV_SEQ_KIND_GRAPH       (3 << CV_SEQ_ELTYPE_BITS)
    #define CV_SEQ_KIND_SUBDIV2D    (4 << CV_SEQ_ELTYPE_BITS)
```
나머지의 비트는, 이외의 순서의 종류나 요소 타입을 지정하기 위해서 이용된다. 예를 들면, 점(CV_SEQ_KIND_CURVE|CV_SEQ_ELTYPE_POINT)(으)로부터 구성되는 곡선에서는, 타입 CV_SEQ_POLYGON 에 속하는 플래그 CV_SEQ_FLAG_CLOSED (이)나, 다른 플래그가 그 아류형을 표현하기 위해서 이용된다. 많은 윤곽 처리 함수는, 입력 순서의 타입을 체크해, 그 타입이 서포트외인 경우는 에러를 발생한다. 파일 cvtypes.h 에는, 서포트되고 있는 정의가 끝난 순서 타입이나 그 외의 속성을 취득하는 보조 매크로의 전리스트가 기술되고 있다. 이하는 순서를 구성하기 위한 기본 요소의 정의이다.

CvSeqBlock

연속한 순서 블록

```
typedef struct CvSeqBlock
{
    struct CvSeqBlock* prev; /* 전의 순서 블록 */
    struct CvSeqBlock* next; /* 다음의 순서 블록 */
    int start_index; /* 블록의 선두 요소의 인덱스 +
    sequence->first->start_index */
    int count; /* 블록내의 요소수 */
    char* data; /* 블록의 선두 요소에의 포인터 */
} CvSeqBlock;
```
순서 블록은, 순환 쌍방향 리스트를 구성한다. 그 때문에 포인터 prev,next (은)는 결코 NULL 에는 안되어, 순서중의 전후의 순서 블록을 가리킨다. 즉, 최종 블록의 next (은)는 선두 블록을, 선두 블록의 prev (은)는 최종 블록을 각각 가리키고 있다. 필드 start_index (와)과 count (은)는 순서속에서 블록의 위치를 추적할 때에 도움이 된다. 예를 들면, 순서가 10 요소로부터 되어 있어 그것이 3,5,2 요소로 나누어져 있어 선두 블록이 파라미터 start_index = 2 그리고 주어지고 있는 경우, 이 순서 블록의 (start_index, count) 의 페어는, 각각 (2,3), (5, 5), (10, 2) 된다. 선두

블록의 파라미터 start_index (은)는, 어느 요소가 순서의 선두에 추가되고 있는 경우를 제외하고, 통상 0 이다.

CvSlice

순서의 슬라이스

```
typedef struct CvSlice
{
    int start_index;
    int end_index;
} CvSlice;

inline CvSlice cvSlice( int start, int end );
#define CV_WHOLE_SEQ_END_INDEX 0x3fffffff
#define CV_WHOLE_SEQ  cvSlice(0, CV_WHOLE_SEQ_END_INDEX)

/* 순서의 슬라이스장을 계산한다 */
int cvSliceLength( CvSlice slice, const CvSeq* seq );
```

순서를 처리하는 함수 중 몇개인가는, 디폴트로 순서 전체(CV_WHOLE_SEQ)(이)가 세트 되고 있는 파라미터 CvSlice slice (을)를 사용한다. start_index 인가 end_index 하지만 부나 순서장을 넘고 있는 경우에서도, start_index(은)는 경계에 포함되어(슬라이스에 포함된다),end_index (은)는 경계에 포함되지 않는다(슬라이스에 포함되지 않는다). 이것들이 동일한 경우는, 슬라이스는 하늘인(즉 요소가 없다)로 간주해진다. 순서는 순환 구조로서 취급되기 위해, 순서 마지막 몇 개의 요소와 거기에 계속 되는 순서 최초의 몇 개의 요소를 슬라이스로서 선택할 수 있다. 예를 들면,10 개의 요소를 가지는 순서의 cvSlice(-2, 3) (은)는, 최종의 하나전(8th), 최종(9th), 선두(0th), 2 번째 (1th) 그리고 3 번째 (2nd)의 요소로부터 된다. 이 함수는 슬라이스의 인수를 이하의 방법으로 정규화한다. 우선 슬라이스장을 조사하기 위해서 cvSliceLength (을)를 호출해, cvGetSeqElem 의 인수와 같게, 슬라이스의 start_index 하지만 정규화된다(즉, 부의 인덱스를 사용할 수 있다). 실제의 슬라이스는, 정규화되었다 start_index 에 시작해, cvSliceLength 요소까지 처리된다(순서가 순환 구조이라고 가정하고 있는 것에 거듭해 주의).

슬라이스를 인수로서 취하지 않는 함수로 순서의 일부만 처리하고 싶은 경우는, 함수 cvSeqSlice (을)를 이용해 서브 순서를 추출하거나 cvCvtSeqToArray 그리고 연속 버퍼로서 보존하는 일로 가능하게 된다(옵션으로서 계속해 cvMakeSeqHeaderForArray (을)를 부른다).

CreateSeq

순서를 생성한다

```
CvSeq* cvCreateSeq( int seq_flags, int header_size,
                    int elem_size, CvMemStorage* storage );
```

seq_flags

생성된 순서의 플래그.생성된 순서가, 특정의 순서 타입을 인수로 취하는 함수에 일절 건네받지 않는 경우는,

이 값에0(을)를 지정해도 상관없다.그렇지 않은 경우는, 정의 끝난 순서 타입의 리스트로부터 적절한 타입이 선택되지 않으면 안 된다.

header_size

순서의 헷다사이즈.sizeof(CvSeq)이상이 아니면 안된다. 또, 특별한 타입인가 그 확장이 지시받고 있는 경우, 그 타입은 기본 타입의 헤더와 합치하고 있지 않으면 안 된다.

elem_size

순서의 요소 사이즈(아르바이트 단위).사이즈는 순서 타입과 합치해야 한다.예를 들면, 점군의 순서를 작성하는 경우, 요소 타입에CV_SEQ_ELTYPE_POINT(을)를 지정해, 파라미터 elem_size 하 sizeof(CvPoint) (와)과 동일하지 않으면 안 된다.

storage

순서가 보존되는 장소.

함수 cvCreateSeq (은)는, 순서를 작성해, 그 포인터를 돌려준다. 이 함수는, 순서 헤더를 스토리지내가 연속한 하나의 블록으로서 파티션 해, 인수로서 건네받은 값을 구조체의 필드 flags,elem_size, header_size,storage 에 인수로 건네준 값을 세트 한다. 그리고,delta_elems 에 디폴트치(함수 cvSetSeqBlockSize 그리고 재설정할 수 있다)를 세트 해, 그 외의 헤더 필드(선두로부터 sizeof(CvSeq) 아르바이트 이후의 스페이스도 포함한다)를 클리어 한다.

SetSeqBlockSize

순서의 브록크사이즈를 설정한다

```
void cvSetSeqBlockSize( CvSeq* seq, int delta_elems );
```

seq

순서.

delta_elems

순서 요소의 브록크사이즈.

함수 cvSetSeqBlockSize (은)는, 메모리 확보 사이즈를 지정한다. 순서 버퍼 건성나무 영역을 다 써 버렸을 경우, 함수는 delta_elems 순서 요소 분의 파티션을 실시한다. 확보한 블록이 전의 블록의 직후가 되는 경우, 두 개의 블록은 접속되지만, 그 이외에서는, 새로운 순서 블록이 생성된다. 그 때문에, 이 파라미터를 크게 하는 것에 의해서 순서 블록의 단편화는 누를 수 있지만, 스토리지중이 많은 스페이스가 낭비되게 된다. 순서 생성시에, 파라미터 delta_elems 에는 디폴트치≈1K 하지만 세트 된다. 순서 생성 후, 이 함수를 언제라도 부를 수 있 이후의 파티션 시에 그 값이 사용된다. 이 함수에 의해서, 건네받은 파라미터치를 메모리스트레이지의 제한에 맞추기 위해서 변경할 수 있다.

SeqPush

순서의 말미에 요소를 추가한다

```
char* cvSeqPush( CvSeq* seq, void* element=NULL );
```

seq

순서.

element

추가되는 요소.

함수 cvSeqPush (은)는, 순서의 말미에 요소를 추가해, 할당할 수 있었던 요소에의 포인터를 돌려준다. 입력의 element 하지만 NULL 의 경우, 이 함수는 단지 요소 하나 분의 영역을 확보한다.

이하의 코드는, 이 함수를 이용해 새로운 순서를 작성하는 방법을 나타낸다.

```
CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* seq = cvCreateSeq( CV_32SC1, /* 정수형 요소의 순서 */
                          sizeof(CvSeq), /* 헷다사이즈 - 확장 필드 없음 */
                          sizeof(int), /* 요소 사이즈 */
                          storage /* 보존 영역 */ );
int i;
for( i = 0; i < 100; i++ )
{
    int* added = (int*)cvSeqPush( seq, &i );
    printf( "%d is addedWn", *added );
}

...
/* 마지막에 메모리스트레이지를 해방한다 */
cvReleaseMemStorage( &storage );
```

함수 cvSeqPush 의 계산량은 O(1)이지만, 대규모 순서 데이터의 기입에는 보다 고속의 수법이 존재한다 (cvStartWriteSeq(와)과 관련하는 함수를 참조).

SeqPop

순서의 말미로부터 요소를 삭제한다

```
void cvSeqPop( CvSeq* seq, void* element=NULL );
```

seq

순서.

element

옵션 파라미터.포인터가 비0의 경우, 삭제한 요소를 이 장소에 카피한다.

함수 cvSeqPop (은)는 순서로부터 하나의 요소를 삭제한다.순서가 이미 하늘의 경우는, 에러를 돌려준다.이 함수의 계산량은 O(1) 이다.

SeqPushFront

순서의 선두에 요소를 추가한다.

```
char* cvSeqPushFront( CvSeq* seq, void* element=NULL );
```

seq

순서.

element

추가되는 요소.

함수 cvSeqPushFront 하 cvSeqPush (와)과 같지만, 새로운 요소는 순서의 선두에 추가된다.이 함수의 계산량은 O(1) 이다.

SeqPopFront

순서의 선두로부터 요소를 삭제한다

```
void cvSeqPopFront( CvSeq* seq, void* element=NULL );
```

seq

순서.

element

옵션 파라미터.포인터가 비0의 경우, 삭제 요소를 이 장소에 카피한다.

함수 cvSeqPopFront (은)는 순서의 선두로부터, 하나의 요소를 삭제한다.순서가 이미 하늘의 경우는, 에러를 돌려준다.이 함수의 계산량은 O(1) 이다.

SeqPushMulti

복수의 요소를 순서의 어느 쪽인가의단(선두나 말미)에 추가한다

```
void cvSeqPushMulti( CvSeq* seq, void* elements, int count, int in_front=0 );
```

seq

순서.

elements

추가되는 요소군.

count

추가되는 요소수.

in_front

변경하는 순서의 구석을 지정하는 플래그.

CV_BACK (=0) - 요소를 순서의 말미에 추가한다

CV_FRONT(!=0) - 요소를 순서의 선두에 추가한다

함수 cvSeqPushMulti (은)는, 복수의 요소를 순서의 어느 쪽인가의단에 추가한다. 요소군은 입력 배열중의 차례대로 순서에 추가되지만, 다른 순서 블록에 분할되기도 한다.

SeqPopMulti

복수의 요소를 순서의 어느 쪽인가의단(선두나 말미)으로부터 삭제한다

```
void cvSeqPopMulti( CvSeq* seq, void* elements, int count, int in_front=0 );
```

seq

순서.

elements

삭제되는 요소.

count

삭제되는 요소수.

in_front

변경하는 순서의 구석을 지정하는 플래그.

CV_BACK (=0) - 순서의 말미로부터 요소를 삭제한다

CV_FRONT(!=0) - 순서의 선두로부터 요소를 삭제한다

함수 cvSeqPopMulti (은)는, 복수의 요소를 순서의 어느 쪽인가의단으로부터 삭제한다. 삭제되는 요소수가 순서중의 요소수보다 큰 경우는, 함수는 가능한 한 많은 요소를 삭제한다.

SeqInsert

순서안에 요소를 삽입한다

```
char* cvSeqInsert( CvSeq* seq, int before_index, void* element=NULL );
```

seq

순서.

before_index

요소가 삽입되는 인덱스(이 인덱스의 전에 삽입된다). 0(지정 가능한 파라미터의 최소치)의 전에의 삽입은 cvSeqPushFront (와)과 같은 의미이며, seq->total(지정 가능한 파라미터의 최대치)의 전에의 삽입은 cvSeqPush (와)과 같은 의미이다.

element

삽입되는 요소.

함수 cvSeqInsert (은)는(element 에의) 포인터가 NULL(이)가 아닌 경우, 순서내의 요소를 지정한 삽입 위치에서 가까운 쪽의 순서의 구석에 시프트 해, element 의 내용을 그 위치에 카피한다.이 함수는, 삽입된 요소에의 포인터를 돌려준다.

SeqRemove

순서중에서 요소를 삭제한다

```
void cvSeqRemove( CvSeq* seq, int index );
```

seq

순서.

index

삭제되는 요소의 인덱스.

함수 cvSeqRemove (은)는, 주어진 인덱스를 가지는 요소를 삭제한다. 인덱스가 범위외의 경우, 이 함수는 에러를 발생한다.하늘의 순서로부터 요소를 삭제하려고 하는 것은, 이 상황의 일례이다. 이 함수는, 가까운 쪽의 순서의 구석과 index 번째 (이것은 삭제되지 않는다)의 위치동안에 존재하는 순서 요소를 시프트 하는 일에 의해서, 요소를 삭제한다.

ClearSeq

순서를 클리어 한다

```
void cvClearSeq( CvSeq* seq );
```

seq

순서.

함수 cvClearSeq (은)는, 순서의 모든 요소를 클리어 한다. 이 함수는 메모리를 스토리지에 돌려주지 않지만, 다음에 새로운 요소가 순서에 추가되었을 때, 이 메모리를 재이용한다. 이 함수의 시간 계산량은, O(1) 이다..

GetSeqElem

인덱스로 지정된 순서 요소의 포인터를 돌려준다

```
char* cvGetSeqElem( const CvSeq* seq, int index );
#define CV_GET_SEQ_ELEM( TYPE, seq, index )  (TYPE*)cvGetSeqElem( (CvSeq*)(seq),
(index) )
```

seq

순서.

index

요소의 인덱스.

함수 cvGetSeqElem (은)는, 주어진 인덱스를 가지는 요소를 순서중에서 요구해 그 포인터를 돌려준다. 요소가 발견되지 않는 경우,0(을)를 돌려준다. 이 함수에서는, 부의 인덱스의 지정도 가능하다.예를 들면,-1(은)는 순서의 마지막 요소,-2(은)는 마지막 하나전을 가리키는 등. 순서가 하나의 순서 블록으로 구성되어 있는지, 목적의 요소가 선두의 블록에 있는 경우는, 매크로 CV_GET_SEQ_ELEM(elemType, seq, index) (을)를 사용하는 것이 좋다.이 매크로의 파라미터 elemType(은)는 순서 요소의 타입(예를 들면,CvPoint),seq (은)는 순서, index (은)는 목적의 요소의 인덱스이다. 매크로는, 먼저 최초로 지정된 요소가 선두의 블록내에 있을지를 체크해, 있으면 그 위치를 돌려주어, 없으면 매크로로부터 메인 함수 GetSeqElem (을)를 호출한다. 부의 인덱스에 대해서는, 항상 cvGetSeqElem (을)를 호출한다. 블록수가 요소수보다 매우 작으면 가정했을 경우는, 함수의 시간 계산량은 O(1)(이)가 된다.

SeqElemIdx

포인터로 지정된 순서의 요소의 인덱스를 돌려준다

```
int cvSeqElemIdx( const CvSeq* seq, const void* element, CvSeqBlock** block=NULL );
```

seq

순서.

element

순서 요소에의 포인터.

block

옵션의 인수.NULL (이)가 아닌 경우, 요소를 포함한 순서 블록의 주소가 이 장소에 보존된다.

함수 cvSeqElemIdx (은)는, 지정된 순서 요소의 인덱스를 돌려준다.그 요소가 존재하지 않는 경우는 부의 값을 돌려준다.

CvtSeqToArray

순서를 메모리내가 연속한 하나의 블록에 카피한다

```
void* cvCvtSeqToArray( const CvSeq* seq, void* elements, CvSlice slice=CV_WHOLE_SEQ );
```

seq

순서.

elements

충분히 큰 영역을 가지는 출력 배열에의 포인터.데이터에의 포인터이며, 행렬의 헤더는 아니다.

slice

배열에 카피하는 순서내의 부분.

함수 cvCvtSeqToArray (은)는, 순서의 전체 또는 일부를 지정된 버퍼에 카피해, 그 버퍼에의 포인터를 돌려준다.

MakeSeqHeaderForArray

배열로부터 순서를 생성한다

```
CvSeq* cvMakeSeqHeaderForArray( int seq_type, int header_size, int elem_size,
                                void* elements, int total,
                                CvSeq* seq, CvSeqBlock* block );
```

seq_type

생성하는 순서의 타입.

header_size

순서 헤더의 사이즈.파라미터seq(은)는, 이 헤더와 같은 사이즈인가, 이것보다 큰 구조체에의 포인터를 가리키지 않으면 안 된다.

elem_size

순서 요소의 사이즈.

elements

순서를 구성하는 요소.

total

순서내의 요소수.배열의 요소수는 이 파라미터치와 동일하지 않으면 안 된다.

seq

순서 헤더로서 이용되는 로컬 변수에의 포인터.

block

단일 순서 블록의 헤더를 나타내는 로컬 변수에의 포인터.

함수 cvMakeSeqHeaderForArray (은)는, 배열용으로 순서 헤더를 초기화한다. 순서 헤더도 순서 블록도 양쪽 모두, 유저에 의해서 파티션 된다(예를 들면, 스택상에). 이 함수에서는 데이터의 카피는 행해지지 않는다. 결과적으로 얻을 수 있는 순서는, 단일 블록으로 구성되어 스토리지 포인터로서 NULL(을)를 가진다. 그 때문에, 그 요소를 읽는 것은 가능하지만, 이 순서에 요소를 추가하려고 하면, 많은 경우는 에러가 된다.

SeqSlice

순서 슬라이스를 위한 다른 헤더를 작성한다

```
CvSeq* cvSeqSlice( const CvSeq* seq, CvSlice slice,
                   CvMemStorage* storage=NULL, int copy_data=0 );
```

seq

순서.

slice

추출하는 순서의 일부분.

storage

새로운 순서 헤더와 카피된 데이터(만약 데이터가 있으면)를 보존하는 출력 스토리지. NULL의 경우, 이 함수는 입력 순서에 포함되는 스토리지를 사용한다.

copy_data

추출된 슬라이스의 요소를 카피한다(copy_data!=0)인가, 하지 않는다(copy_data=0)인지를 나타내는 플래그.

함수 cvSeqSlice (은)는, 입력 순서의 지정한 슬라이스에 상당하는 새로운 순서를 작성한다. 새로운 순서에서는, 그 요소를 원래의 순서와 공유하는지, 독자로 그 카피를 가질까의 언젠가이다. 순서의 일부만을 처리할 필요가 있지만, 그 처리 함수가 슬라이스 파라미터를 가지지 않는 경우에, 필요한 서브 순서를 이 함수로 추출할 수 있다.

CloneSeq

순서의 카피를 작성한다

```
CvSeq* cvCloneSeq( const CvSeq* seq, CvMemStorage* storage=NULL );
```

seq

순서.

storage

새로운 순서 헤더와 카피된 데이터(만약 데이터가 있으면)를 보존하는 출력 스토리지. NULL의 경우, 입력 순서에 포함되는 스토리지를 사용한다.

함수 cvCloneSeq (은)는, 입력 순서의 완전한 카피를 작성해, 그 포인터를 돌려준다. cvCloneSeq(seq, storage)하 cvSeqSlice(seq, CV_WHOLE_SEQ, storage, 1) 에 동일하다.

SeqRemoveSlice

순서 슬라이스를 삭제한다

```
void cvSeqRemoveSlice( CvSeq* seq, CvSlice slice );
```

seq

순서.

slice

삭제하는 순서의 일부분.

함수 cvSeqRemoveSlice (은)는, 지정된 순서로부터 슬라이스를 삭제한다.

SeqInsertSlice

순서내에 배열을 삽입한다

```
void cvSeqInsertSlice( CvSeq* seq, int before_index, const CvArr* from_arr );
```

seq

순서.

slice

배열이 삽입되는 장소에의 인덱스(인덱스의 전에 삽입된다).

from_arr

추가되는 요소의 배열.

함수 cvSeqInsertSlice (은)는, from_arr 배열의 모든 요소를 순서의 지정된 위치에 삽입한다. 배열 from_arr (은)는, 행렬 혹은, 다른 순서라도 좋다.

SeqInvert

순서 요소의 순서를 반전시킨다

```
void cvSeqInvert( CvSeq* seq );
```

seq

순서.

함수 cvSeqInvert (은)는, 순서내의 순서를 반전시킨다(선두의 요소는 말미에, 말미의 요소는 선두에, 와 같이).

SeqSort

순서의 요소를, 지정한 비교 함수를 이용해 소트 한다

```
/* a < b ? -1 : a > b ? 1 : 0 */
typedef int (CV_CDECL* CvCmpFunc)(const void* a, const void* b, void* userdata);

void cvSeqSort( CvSeq* seq, CvCmpFunc func, void* userdata=NULL );
```

seq

소트 되는 순서

func

요소의 관계에 따르고, 부·0·정의 값을 돌려주는 비교 함수(상기의 선언과 아래의 예를 참조.). 마지막 인수userdata하지만 사용되지 않는 것을 제외하면 같은 함수가,C의 함수 qsort 그렇지만 사용된다.

userdata

비교 함수에게 건네지는 유저 파라미터.글로벌 변수의 사용을 피하기 위해서 유효하다.

함수 cvSeqSort(은)는, 지정한 기준을 이용해 순서를 소트 한다.이하에, 이 함수의 사용예를 나타낸다.

```
/* 2 차원의 점을 위에서 아래, 왼쪽에서 오른쪽에 소트 한다 */
static int cmp_func( const void* _a, const void* _b, void* userdata )
{
    CvPoint* a = (CvPoint*)_a;
    CvPoint* b = (CvPoint*)_b;
    int y_diff = a->y - b->y;
    int x_diff = a->x - b->x;
    return y_diff ? y_diff : x_diff;
}

...

CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* seq = cvCreateSeq( CV_32SC2, sizeof(CvSeq), sizeof(CvPoint), storage );
int i;

for( i = 0; i < 10; i++ )
{
    CvPoint pt;
    pt.x = rand() % 1000;
    pt.y = rand() % 1000;
    cvSeqPush( seq, &pt );
}

cvSeqSort( seq, cmp_func, 0 /* 여기에서는 유저 데이터는 사용하지 않는다 */ );

/* 소트 된 데이터를 출력 */
for( i = 0; i < seq->total; i++ )
{
    CvPoint* pt = (CvPoint*)cvSeqElem( seq, i );
    printf( "(%d,%d)₩n", pt->x, pt->y );
}

cvReleaseMemStorage( &storage );
```

SeqSearch

순서로부터 지정 요소를 검색한다

```
/* a < b ? -1 : a > b ? 1 : 0 */
typedef int (CV_CDECL* CvCmpFunc)(const void* a, const void* b, void* userdata);

char* cvSeqSearch( CvSeq* seq, const void* elem, CvCmpFunc func,
                   int is_sorted, int* elem_idx, void* userdata=NULL );
```

seq

순서.

elem

검색하는 요소.

func

요소의 관계에 따르고, 부·0·정의 값을 돌려주는 비교 함수(cvSeqSort 도 참조).

is_sorted

순서가 소트가 끝난 상태인가 아닌가를 나타내는 플래그.

elem_idx

출력 파라미터.발견된 요소의 인덱스.

userdata

비교 함수에게 건네지는 유저 파라미터.글로벌 변수의 사용을 피하기 위해서 유효하다.

함수 cvSeqSearch (은)는 순서중에서 요소를 검색한다. 순서가 소트 되고 있으면,O(log(N))의 2 분 탐색법이 이용된다. 그 외의 경우는, 단순한 선형 탐색이 이용된다. 요소가 발견되지 않는 경우, 이 함수는 NULL 포인터를 돌려주어, 인덱스에는 순서의 요소수를 세트 한다. 선형 탐색을 이용하고 요소를 찾아냈을 경우, 인덱스에는 seq(i)>elem 인 최소의 인덱스 i 하지만 세트 된다.

StartAppendToSeq

순서에의 데이터 기입 처리를 초기화한다

```
void cvStartAppendToSeq( CvSeq* seq, CvSeqWriter* writer );
```

seq

순서에의 포인터.

writer

라이터(Writer) 상태.이 함수로 초기화된다.

함수 cvStartAppendToSeq (은)는, 순서에의 데이터 기입 처리를 초기화한다. 써지는 요소는, 매크로 CV_WRITE_SEQ_ELEM(written_elem, writer) 에 의해서 순서의 마지막에 추가된다. 기입 처리동안에 순서에 대한 다른 처리를 하면, 잘못된 결과가 일으켜지거나 순서 자체가 망가질 가능성이 있는 것에 주의 (이러한 문제를 회피하려면 cvFlushSeqWriter 의 설명을 참조).

StartWriteSeq

새로운 순서를 작성해, 라이터(writer)(을)를 초기화한다

```
void cvStartWriteSeq( int seq_flags, int header_size, int elem_size,
                      CvMemStorage* storage, CvSeqWriter* writer );
```

seq_flags

작성된 순서의 플래그. 생성된 순서가, 특정의 순서 타입을 인수로 취하는 함수에 일절 건네받지 않는 경우는, 이 값에0(을)를 지정해도 상관없다. 그렇지 않은 경우는, 정의 끝난 순서 타입의 리스트로부터 적절한 타입이 선택되지 않으면 안 된다.

header_size

순서 헤더의 사이즈.sizeof(CvSeq)이상이 아니면 안된다. 또, 특별한 타입이나 확장이 지시받고 있는 경우, 그 타입은 기본 타입의 헷다사이즈와 합치해야 한다.

elem_size

순서의 요소 사이즈(아르바이트 단위).사이즈는 순서 타입과 합치해야 한다. 예를 들면, 점군의 순서(요소 타입은 CV_SEQ_ELTYPE_POINT)(을)를 작성하는 경우, 파라미터 elem_size 하 sizeof(CvPoint) (와)과 동일하지 않으면 안 된다.

storage

순서의 위치.

writer

라이터 상태.이 함수로 초기화된다.

함수 cvStartWriteSeq (은)는,cvCreateSeq (와)과 cvStartAppendToSeq 의 편성이다. 생성된 순서에의 포인터는 writer->seq 에 보존되어 마지막에 불려 가야 할 함수 cvEndWriteSeq (으)로부터 돌려주어진다.

EndWriteSeq

순서 기입 처리를 종료한다

```
CvSeq* cvEndWriteSeq( CvSeqWriter* writer );
```

writer

라이터 상태.

함수 cvEndWriteSeq (은)는 기입 처리를 종료해, 써진 순서에의 포인터를 돌려준다. 이 함수는 비어 블록을 메모리스트레이지에 돌려주기 위해서 마지막 불완전한 순서 블록을 잘라 버린다. 이 처리에 의해서, 순서는 안전하게 읽고 쓰기할 수 있게 된다.

FlushSeqWriter

라이터 상태로부터 순서 헤더를 갱신한다

```
void cvFlushSeqWriter( CvSeqWriter* writer );
```

writer

라이터 상태.

함수 cvFlushSeqWriter (은)는, 기입 프로세스중에서도 요구(예를 들면, 특정 상태를 체크하는 등)가 있으면 언제라도, 유저가 순서 요소를 읽어낼 수 있도록(듯이) 한다.이 함수는, 순서로부터의 읽기를 가능하게 하기 위해서 헤더를 갱신한다. 그러나, 라이터를 닫지 않기 때문에, 언제라도 써 처리를 계속할 수 있다. 빈번히 flush (을)를 사용하는 알고리즘에 대해서는, 대신에 cvSeqPush(을)를 이용하는 일도 고려한다.

StartReadSeq

순서로부터의 연속 읽기 처리를 초기화한다

```
void cvStartReadSeq( const CvSeq* seq, CvSeqReader* reader, int reverse=0 );
```

seq

순서.

reader

리더(reader) 상태.이 함수로 초기화된다.

reverse

순서 주사 방향의 지정.reverse 하지만 0 의 경우, 리더는 선두의 순서 요소에 위치한다.그 이외는 마지막 요소에 위치한다.

함수 cvStartReadSeq (은)는, 리더 상태를 초기화한다. 이 후, 선두로부터 말미까지의 전순서 요소는, 계속 되어 불리는 매크로 CV_READ_SEQ_ELEM(read_elem, reader)(순서 방향 읽기의 경우) 혹은, 매크로 CV_REV_READ_SEQ_ELEM(read_elem, reader)(역방향 읽기의 경우)(을)를 이용하고, 읽기가 가능하게 된다. 어느 쪽의 매크로도 순서 요소를 read_elem 에 세트 해, 읽기 포인터를 다음의 요소 위치에 이동시킨다. 순서 블록의 순환 구조는, 읽기 처리에도 적용된다. 이것은 매크로 CV_READ_SEQ_ELEM 에 의해서 순서 말미의 요소가 이미 읽어내진 후, 매크로가 재차 불린다고 순서 선두의 요소가 읽어내지는 것이다.CV_REV_READ_SEQ_ELEM 냄새나도 같이이다. 읽기 처리는 순서의 변경도 텐포라리밧파의 생성도 하지 않기 때문에, 이 처리를 종료시키는 함수는 존재하지 않는다. 리더 구조체의 필드 ptr (은)는, 다음에 읽어내지는 순서의 현재의 요소를 가리키고 있다. 이하의 코드는, 순서의 라이터와 리더의 사용예이다.

```
CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* seq = cvCreateSeq( CV_32SC1, sizeof(CvSeq), sizeof(int), storage );
CvSeqWriter writer;
CvSeqReader reader;
int i;

cvStartAppendToSeq( seq, &writer );
for( i = 0; i < 10; i++ )
{
    int val = rand()%100;
    CV_WRITE_SEQ_ELEM( val, writer );
    printf("%d is written₩n", val );
}
cvEndWriteSeq( &writer );

cvStartReadSeq( seq, &reader, 0 );
for( i = 0; i < seq->total; i++ )
{
    int val;
#if 1
    CV_READ_SEQ_ELEM( val, reader );
    printf("%d is read₩n", val );
```

```
#else /* 대신 방법.순서의 요소수가 큰 경우나, 컴파일시에
         사이즈나 타입을 모르는 경우는 이쪽의 사용이 바람직하다 */
    printf("%d is readWn", *(int*)reader.ptr );
    CV_NEXT_SEQ_ELEM( seq->elem_size, reader );
#endif
}
...

cvReleaseStorage( &storage );
```

GetSeqReaderPos

현재의 리더의 위치를 돌려준다

```
int cvGetSeqReaderPos( CvSeqReader* reader );
```

reader

리더 상태.

함수 cvGetSeqReaderPos (은)는, 현재의 리더의 위치(0 ... reader->seq->total - 1 의 범위)를 돌려준다.

SetSeqReaderPos

리더를 지정의 위치로 이동한다

```
void cvSetSeqReaderPos( CvSeqReader* reader, int index, int is_relative=0 );
```

reader

리더 상태.

index

이동처의 위치.위치 결정 모드(다음의 파라미터is_relative(을)를 참조)가 사용되고 있는 경우, 실제의 위치는 index (을)를 reader->seq->total그리고 나눈 잉여가 된다.

is_relative

비0의 경우,index (은)는 현재 위치로부터의 상대치.

함수 cvSetSeqReaderPos (은)는, 읽기 위치를 절대 위치나 상대 위치에서 나타내진 위치로 이동한다.

3-3 설정 함수(Sets)

CvSet

노드의 집합

```
typedef struct CvSetElem
{
    int flags; /* 빈노드라면 부, 다른 경우 0 인가 정의 값 */
    struct CvSetElem* next_free; /* 빈노드의 경우, 다음의 빈노드에의 포인터 */
}
CvSetElem;

#define CV_SET_FIELDS()    ₩
    CV_SEQUENCE_FIELDS()   /* CvSeq(으)로부터 계승한다 */ ₩
    struct CvSetElem* free_elems; /* 빈노드의 리스트 */

typedef struct CvSet
{
    CV_SET_FIELDS()
} CvSet;
```

구조체 CvSet (은)는,OpenCV 의 드문드문한 데이터 구조의 기본이다.

상술의 선언대로,CvSet 하 CvSeq(을)를 계승해, 거기에 free_elems(빈노드 리스트)(을)를 더한 것이다. 빈 곳인가 그렇지 않은가가 기술된 세트의 각 노드는, 내부적인 순서의 요소이다. 조밀한 순서의 요소에 대하는 제한은 없지만, 세트(와 여기로부터 파생하는 구조체)의 요소는

정수형의 필드에서 시작할 필요가 있어, 구조체 CvSetElem 에 합치시킬 수가 할 수 없으면 안 된다. 왜냐하면, 이것들 두 개의 필드(정수로 거기에 계속 되는 포인터)는, 빈노드 리스트를 가지는 노드 세트의 구성에 필요하기 때문에이다. 빈노드의 경우,flags 필드는 부의 값이 된다(필드의 최상위비트(MSB)(이)가 세트 되고 있다). 그리고 next_free(은)는, 다음의 빈노드를 가리킨다(선두의 빈노드는,CvSet 의 free_elems 필드로부터 참조된다).노드가 사용되고 있는 경우,flags 필드는 0 이상의 값으로,(set_elem->flags & CV_SET_ELEM_IDX_MASK) 의 식을 이용해 꺼내지는 노드 인덱스를 가진다.노드의 남은 부분(next_free)(은)는 유저에 의해서 결정할 수 있다. 특히, 사용되고 있는 노드는 비어 노드와 같이 링크 되어 있지 않기 때문에,2 번째의 필드(next_free)(은)는, 이러한 링크나 다른 목적을 위해서 이용할 수 있다.
매크로 CV_IS_SET_ELEM(set_elem_ptr)(은)는, 지정한 노드가 사용되고 있는지 어떤지를 판단하기 위해서 이용할 수 있다.

최초, 세트와 리스트는 하늘이다.세트로부터 새로운 노드 요구가 있었을 경우, 빈노드 리스트로부터 노드가 꺼내져 그 후 리스트는 갱신된다. 빈노드 리스트가 하늘의 경우는, 새로운 순서 블록이 확보되어 블록내의 모든 노드가 비어 노드 리스트에 추가된다. 따라서, 세트의 total 필드에는, 사용되고 있는 노드수로 비어 노드의 수의 합계가 들어간다. 사용되고 있던 노드가 해방되었을 때에는, 빈노드 리스트에 더해진다.마지막에 해방되는 노드는, 최초로 사용된 것이다.

OpenCV 에 두어 CvSet (은)는, 그래프(CvGraph), 드문드문한 다차원 배열(CvSparseMat), 평면의 세분비율(CvSubdiv2D) 등을 표현하기 위해서 사용된다.

CreateSet

하늘세트를 생성한다

```
CvSet* cvCreateSet( int set_flags, int header_size,
                    int elem_size, CvMemStorage* storage );
```

set_flags
: 생성하는 세트의 타입.

header_size
: 세트의 헤더의 사이즈(sizeof(CvSet)이상).

elem_size
: 세트의 요소의 사이즈(CvSetElem 이상).

storage
: 세트를 위한 컨테이너.

함수 cvCreateSet(은)는 지정된 헤더의 사이즈와 요소의 사이즈를 가지는 하늘세트를 생성해, 그 세트에의 포인터를 돌려준다. 이 함수는 cvCreateSeq 에 다소의 처리를 추가한 것에 지나지 않는다.

SetAdd

세트에 새로운 노드(요소)를 추가한다

```
int cvSetAdd( CvSet* set_header, CvSetElem* elem=NULL, CvSetElem** inserted_elem=NULL );
```

set_header

세트.

elem

옵션의 입력 인수.삽입하는 요소. NULL(이)가 아닌 경우, 새롭게 확보한 노드에 데이터를 카피한다 (카피한 후, 선두의 정수 필드의 최상위비트는 클리어 된다).

inserted_elem

옵션의 출력 인수.할당할 수 있었던 요소에의 포인터.

함수 cvSetAdd(은)는, 새로운 노드를 할당한다. 옵션으로서 입력 요소 데이터를 노드에 카피해, 노드에의 포인터와 인덱스를 돌려준다. 인덱스치는, 노드의 flags 필드의 하위의 비트로부터 얻는다.이 함수는 O(1)의 계산량을 가지지만, 세트의 노드를 확보하기 위한보다 고속의 함수가 존재한다(cvSetNew(을)를 참조).

SetRemove

세트로부터 요소를 삭제한다

```
void cvSetRemove( CvSet* set_header, int index );
```

set_header

세트.

index

삭제하는 요소의 인덱스.

함수 cvSetRemove(은)는, 세트로부터 지정된 인덱스의 요소를 없앤다. 만약 지정된 장소의 노드가 없는 경우, 이 함수는 아무것도 하지 않는다.이 함수는 O(1)의 계산량을 가지는, 그러나, 이미 없애는 요소의 장소가 기존인 경우는,cvSetRemoveByPtr (분)편이 고속으로 있다.

SetNew

세트에 요소를 더한다(고속판)

```
CvSetElem* cvSetNew( CvSet* set_header );
```

set_header

세트.

함수 cvSetNew(은)는,cvSetAdd 의 고속 인 라인 처리 버젼이다. 이 함수는 새로운 노드를 확보해, 인덱스는 아니고 노드에의 포인터를 돌려준다.

SetRemoveByPtr

포인터로 지정한 세트의 요소를 삭제한다

```
void cvSetRemoveByPtr( CvSet* set_header, void* elem );
```

set_header

세트.

elem

삭제되는 요소.

함수 cvSetRemoveByPtr(은)는, 요소의 포인터를 이용한다 cvSetRemove 의 고속 인 라인 처리 버젼이다. 이 함수는 노드가 사용되고 있을까 그렇지 않은가를 확인하지 않는다.그 때문에, 유저는 주의해 사용해야 한다.

GetSetElem

인덱스에 의해서 세트의 요소를 검색한다

```
CvSetElem* cvGetSetElem( const CvSet* set_header, int index );
```

set_header

세트.

index

순서안세트의 요소의 인덱스.

함수 cvGetSetElem 하, 인덱스에 의해서 세트의 요소를 검색한다. 이 함수는 찾아낸 요소에의 포인터를 돌려주는, 인덱스가 무효인지, 대응하는 노드가 빈 곳의 경우는 0(을)를 돌려준다. 이 함수는 부의 인덱스를 서포트해, 이것은 노드의 장소를 얻기 위해서 cvGetSeqElem (을)를 이용할 때에 사용된다.

ClearSet

세트를 클리어 한다

```
void cvClearSet( CvSet* set_header );
```

set_header

클리어 되는 세트.

함수 cvClearSet(은)는, 세트로부터 모든 요소를 없앤다.이것은 O(1)의 계산량을 가진다.

3-4 그래프(Graphs)

CvGraph

중량감 첨부의 유향 또는 무향그래프

```
#define CV_GRAPH_VERTEX_FIELDS()    ₩
    int flags; /* 정점 플래그 */   ₩
    struct CvGraphEdge* first; /* 최초의 옆 */

typedef struct CvGraphVtx
{
    CV_GRAPH_VERTEX_FIELDS()
```

```
    }
    CvGraphVtx;

    #define CV_GRAPH_EDGE_FIELDS()      ₩
        int flags; /* 옆플래그 */     ₩
        float weight; /* 옆의 중량감 */ ₩
        struct CvGraphEdge* next[2]; /* 옆리스트에 있어서의 다음의 옆.시점(0), 종점(1) */
₩
        struct CvGraphVtx* vtx[2]; /* 시점(0), 종점(1) */

    typedef struct CvGraphEdge
    {
        CV_GRAPH_EDGE_FIELDS()
    }
    CvGraphEdge;

    #define  CV_GRAPH_FIELDS()                  ₩
        CV_SET_FIELDS() /* 정점세트 */   ₩
        CvSet* edges;   /* 옆세트*/

    typedef struct CvGraph
    {
        CV_GRAPH_FIELDS()
    }
    CvGraph;
```

구조체 CvGraph (은)는,OpenCV 그리고 사용되는 그래프의 기본 구조이다.

그래프 구조체는,CvSet(일반적인 그래프의 특성과 그래프 정점을 기술)(을)를 계승해, 한층 더 멤버로서 또 하나의 세트(그래프의 옆을 기술)를 포함하고 있다.

정점, 옆, 그래프 헤더의 구조체는, 다른 확장 가능한 OpenCV 의 구조체와 같은 방법(구조체의 확장이나 커스터마이즈를 심플하게 하는 매크로를 이용해)으로 선언된다. 정점과 옆의 구조체는 명시적으로는 CvSetElem (을)를 계승하고 있지 않지만, 그것들은 세트의 요소의 두 개의 조건(선두에 정수 필드 flags(을)를 가져,CvSetElem 구조체와 합치한다)를 만족하다. 필드 flags (은)는, 정점과 옆이 존재하고 있는 것을 나타낼 뿐만 아니라, 다른 목적 (예를 들면, 그래프의 주사(cvCreateGraphScanner 등을 참조))에도 이용되므로, 이 필드를 직접 사용하지 않는 편이 좋다.

그래프는 옆세트로서 표현되어 각변은 거기에 접속하는 옆의 리스트를 가진다.정보의 중복을 가능한 한 피하기 위해서, 다른 정점에의 접속 리스트가 교대로 배치된다.

그래프는 유향이나 무향인가의 언젠가이다.무향그래프의 경우, 정점 A(으)로부터 B 에의 접속과 정점 B(으)로부터 A 에의 접속의 구별은 없고, 그래프상에는 어느 쪽인지 한편 밖에 동시에 존재할 수 없다.이러한 접속은 각각 옆 <A, B> ,<B, A> (이)라고 표현된다.

CreateGraph

하늘의 그래프를 생성한다

```
CvGraph* cvCreateGraph( int graph_flags, int header_size, int vtx_size,
                        int edge_size, CvMemStorage* storage );
```

graph_flags

생성한 그래프의 타입.무향그래프의 경우,CV_SEQ_KIND_GRAPH, 유향그래프의 경우,CV_SEQ_KIND_GRAPH | CV_GRAPH_FLAG_ORIENTED.

header_size

그래프의 헷다사이즈 (sizeof(CvGraph)이상).

vtx_size

그래프의 정점 사이즈(커스터마이즈 된 정점의 구조체는 CvGraphVtx 그리고 시작되지 않으면 안 된다(CV_GRAPH_VERTEX_FIELDS() (을)를 사용)).

edge_size

그래프의 옆사이즈(커스터마이즈 된 옆의 구조체는 CvGraphEdge 그리고 시작되지 않으면 안 된다(CV_GRAPH_EDGE_FIELDS() (을)를 사용)).

storage

그래프 컨테이너.

함수 cvCreateGraph (은)는, 하늘의 그래프를 생성해, 그 포인터를 돌려준다.

GraphAddVtx

그래프에 정점을 추가한다

```
int cvGraphAddVtx( CvGraph* graph, const CvGraphVtx* vtx=NULL,
                   CvGraphVtx** inserted_vtx=NULL );
```

graph

그래프.

vtx

추가되는 정점의 초기화에 사용되는, 옵션의 입력 인수(sizeof(CvGraphVtx)의 영역을 넘은 유저 정의 필드만 카피된다).

inserted_vertex

옵션의 출력 인수.NULL(이)가 아닌 경우, 새로운 정점의 주소가 여기에 쓰여진다.

함수 cvGraphAddVtx (은)는 그래프에 정점을 추가해, 정점의 인덱스를 돌려준다.

GraphRemoveVtx

그래프로부터 정점을 삭제한다(인덱스 지정)

```
int cvGraphRemoveVtx( CvGraph* graph, int index );
```

graph

그래프.

vtx_idx

삭제되는 정점의 인덱스.

함수 cvGraphRemoveVtx (은)는, 지정한 정점과 그 정점으로 접속하는 모든 옆을 그래프로부터 삭제한다. 입력된 정점이 그래프에 존재하지 않는 경우는, 에러를 출력한다. 반환값은 삭제된 옆의 수, 혹은 정점이 그래프에 존재하지 않는 경우는-1 이다.

GraphRemoveVtxByPtr

그래프로부터 정점을 삭제한다(포인터 지정)

```
int cvGraphRemoveVtxByPtr( CvGraph* graph, CvGraphVtx* vtx );
```

graph

그래프.

vtx

삭제되는 정점에의 포인터.

함수 cvGraphRemoveVtxByPtr (은)는, 지정한 정점과 그 정점으로 접속하는 모든 옆을 그래프로부터 삭제한다. 입력된 정점이 그래프에 존재하지 않는 경우는, 에러를 출력한다. 반환값은 삭제된 옆의 수, 혹은 정점이 그래프에 존재하지 않는 경우는-1 이다.

GetGraphVtx

인덱스를 이용해 그래프의 정점을 검색한다

```
CvGraphVtx* cvGetGraphVtx( CvGraph* graph, int vtx_idx );
```

graph

그래프.

vtx_idx

정점의 인덱스.

함수 cvGetGraphVtx (은)는, 그래프의 정점을 인덱스로부터 검색해, 그 포인터를 돌려준다. 정점이 그래프내에 존재하지 않는 경우는,NULL(을)를 돌려준다.

GraphVtxIdx

그래프 정점의 인덱스를 돌려준다

```
int cvGraphVtxIdx( CvGraph* graph, CvGraphVtx* vtx );
```

graph

그래프.

vtx

그래프 정점에의 포인터.

함수 cvGraphVtxIdx (은)는, 포인터로 지정된 그래프 정점의 인덱스를 돌려준다.

GraphAddEdge

그래프에 옆을 추가한다(인덱스 지정)

```
int cvGraphAddEdge( CvGraph* graph, int start_idx, int end_idx,
                    const CvGraphEdge* edge=NULL, CvGraphEdge** inserted_edge=NULL );
```

graph

그래프.

start_idx

옆의 시점을 나타내는 인덱스.

end_idx

옆의 종점을 나타내는 인덱스.무향그래프의 경우, 순서는 어디라도 좋다.

edge

옵션의 입력 파라미터.옆의 초기화를 위한 데이터.

inserted_edge

입력된 옆의 주소를 보존하기 위한 , 옵션의 출력 파라미터.

함수 cvGraphAddEdge (은)는, 지정한 두 개의 정점을 접속한다. 이 함수는, 추가에 성공하면 1(을)를 돌려주어, 벌써 두 개의 정점을 접속하는 옆이 존재하는 경우는 0(을)를 돌려주어, 정점의 어느 쪽인지가 존재하지 않는, 시점과 종점이 같을이라고 해 다른 중대한 문제가 있을 때는-1(을)를 돌려준다. 후자의 경우(즉, 결과가 부 때), 이 함수는 디폴트로 에러도 출력한다.

GraphAddEdgeByPtr

그래프에 옆을 추가한다(포인터 지정)

```
int cvGraphAddEdgeByPtr( CvGraph* graph, CvGraphVtx* start_vtx, CvGraphVtx* end_vtx,
                         const CvGraphEdge* edge=NULL, CvGraphEdge**
inserted_edge=NULL );
```

graph

그래프.

start_vtx

옆의 시점을 나타내는 정점에의 포인터.

end_vtx

옆의 종점을 나타내는 정점에의 포인터.무향그래프의 경우, 순서는 어디라도 좋다.

edge

옵션의 입력 파라미터, 옆의 초기화 데이터.

inserted_edge

옆의 집합 중(안)에서 입력된 옆의 주소를 보존하기 위한 , 옵션의 출력 파라미터.

함수 cvGraphAddEdgeByPtr (은)는 지정한 두 개의 정점을 접속한다. 이 함수는, 추가에 성공하면 1(을)를 돌려주어, 벌써 두 개의 정점을 접속하는 옆이 존재하는 경우는 0(을)를 돌려주어, 정점의 어느 쪽인지가 존재하지 않는, 시점과 종점이 같을이라고 해 다른 중대한

문제가 있을 때는-1(을)를 돌려준다. 후자의 경우(즉, 결과가 부 때), 함수는 디폴트로 에러도 출력한다.

GraphRemoveEdge

그래프로부터 옆을 삭제한다(인덱스 지정)

```
void cvGraphRemoveEdge( CvGraph* graph, int start_idx, int end_idx );
```

graph

그래프.

start_idx

옆의 시점을 나타내는 정점의 인덱스.

end_idx

옆의 종점을 나타내는 정점의 인덱스.무향그래프의 경우, 순서는 어디라도 좋다.

함수 cvGraphRemoveEdge (은)는, 지정된 두 개의 정점을 접속하는 옆을 삭제한다. 정점이(이 순서로) 접속되어 있지 않은 경우, 이 함수는 아무것도 하지 않는다.

GraphRemoveEdgeByPtr

그래프로부터 옆을 삭제한다(포인터 지정)

```
void cvGraphRemoveEdgeByPtr( CvGraph* graph, CvGraphVtx* start_vtx, CvGraphVtx*
end_vtx );
```

graph

그래프.

start_vtx

옆의 시점을 나타내는 정점에의 포인터.

end_vtx

옆의 종점을 나타내는 정점에의 포인터.무향그래프의 경우, 순서는 어디라도 좋다.

함수 cvGraphRemoveEdgeByPtr (은)는, 지정된 두 개의 정점을 접속하는 옆을 삭제한다. 정점이(이 순서로) 접속되어 있지 않은 경우, 이 함수는 아무것도 하지 않는다.

FindGraphEdge

그래프로부터 옆을 검출한다(인덱스 지정)

```
CvGraphEdge* cvFindGraphEdge( const CvGraph* graph, int start_idx, int end_idx );
#define cvGraphFindEdge cvFindGraphEdge
```

graph

그래프.

start_idx

옆의 시점을 나타내는 정점의 인덱스.

end_idx

　　옆의 종점을 나타내는 정점의 인덱스.무향그래프의 경우, 순서는 어디라도 좋다.

함수 cvFindGraphEdge (은)는, 지정한 두 개의 정점을 접속하는 옆을 검출해, 그 포인터를 돌려준다. 옆이 존재하지 않는 경우는 NULL(을)를 돌려준다.

FindGraphEdgeByPtr

그래프로부터 옆을 검출한다(포인터 지정)

```
CvGraphEdge* cvFindGraphEdgeByPtr( const CvGraph* graph, const CvGraphVtx* start_vtx,
                                   const CvGraphVtx* end_vtx );
#define cvGraphFindEdgeByPtr cvFindGraphEdgeByPtr
```

graph

　　그래프.

start_vtx

　　옆의 시점을 나타내는 정점에의 포인터.

end_vtx

　　옆의 종점을 나타내는 정점에의 포인터.무향그래프의 경우, 순서는 어디라도 좋다.

함수 cvFindGraphEdgeByPtr (은)는, 지정한 두 개의 정점을 접속하는 옆을 검출해, 그 포인터를 돌려준다. 옆이 존재하지 않는 경우는 NULL(을)를 돌려준다.

GraphEdgeIdx

그래프의 옆의 인덱스를 돌려준다

```
int cvGraphEdgeIdx( CvGraph* graph, CvGraphEdge* edge );
```

graph

　　그래프.

edge

　　옆에의 포인터.

함수 cvGraphEdgeIdx (은)는, 그래프의 옆의 인덱스를 돌려준다.

GraphVtxDegree

정점으로 접속하고 있는 옆의 수를 센다(인덱스 지정)

```
int cvGraphVtxDegree( const CvGraph* graph, int vtx_idx );
```

graph

　　그래프.

vtx

　　정점의 인덱스.

함수 cvGraphVtxDegree (은)는, 지정한 정점으로 접속했다(들어간다/나오는 양방향의) 옆의 수를 돌려준다. 옆의 수를 세기 위해서, 이하의 코드가 이용된다.

```
    CvGraphEdge* edge = vertex->first; int count = 0;
    while( edge )
    {
        edge = CV_NEXT_GRAPH_EDGE( edge, vertex );
        count++;
    }
```

매크로 CV_NEXT_GRAPH_EDGE(edge, vertex) (은)는, vertex 에 접속하는,edge 의 다음의 옆을 돌려준다.

GraphVtxDegreeByPtr

정점으로 접속하고 있는 옆의 수를 센다(포인터 지정)

```
int cvGraphVtxDegreeByPtr( const CvGraph* graph, const CvGraphVtx* vtx );
```

graph

그래프.

vtx

정점에의 포인터.

함수 cvGraphVtxDegreeByPtr (은)는, 지정한 정점으로 접속했다(들어간다/나오는 양방향의) 옆의 수를 돌려준다.

ClearGraph

그래프를 클리어 한다

```
void cvClearGraph( CvGraph* graph );
```

graph

그래프.

함수 cvClearGraph (은)는, 그래프로부터 모든 정점과 옆을 삭제한다. 이 함수의 시간 계산량은 O(1)이다.

CloneGraph

그래프를 카피한다

```
CvGraph* cvCloneGraph( const CvGraph* graph, CvMemStorage* storage );
```

graph

카피하는 그래프.

storage

카피를 위한 컨테이너.

함수 cvCloneGraph (은)는, 그래프의 완전 카피를 작성한다. 그래프의 정점이나 옆이 외부 데이터에의 포인터를 가지는 경우, 그 포인터는 카피한 그래프에서도 공유된다. 이 함수는 정점이나 옆의 집합을 최적화(데후라그먼트)하기 위한(해), 새로운 그래프의 정점과 옆의 인덱스가 원래의 그래프의 것과 다를 가능성이 있다.

CvGraphScanner

그래프 주사 상태를 위한 구조체

```
typedef struct CvGraphScanner
{
    CvGraphVtx* vtx;        /* 현재의 정점(혹은 옆의 시점) */
    CvGraphVtx* dst;        /* 현재의 옆의 접속처 정점 */
    CvGraphEdge* edge;      /* 현재의 옆 */

    CvGraph* graph;         /* 주사중의 그래프 */
    CvSeq*   stack;         /* 주사중의 정점 스택 */
    int      index;         /* 주사 끝난 정점의 하한 인덱스 */
    int      mask;          /* 이벤트 마스크 */
}
CvGraphScanner;
```

구조체 CvGraphScanner (은)는, 깊이 우선 주사를 위해서 이용된다. 이하의 함수의 설명을 참조.

CreateGraphScanner

그래프가 깊이 우선 주사를 위한 구조체를 생성한다

```
CvGraphScanner* cvCreateGraphScanner( CvGraph* graph, CvGraphVtx* vtx=NULL,
                                      int mask=CV_GRAPH_ALL_ITEMS );
```

graph

그래프.

vtx

주사 개시 정점.NULL의 경우, 주사는 선두의 정점(정점 순서 중 최소의 인덱스를 가지는 정점)으로부터 시작된다.

mask

이벤트 마스크.어느 이벤트에 유저가 주목하고 싶은 것인지를 지정한다 (함수 cvNextGraphItem (은)는 유저에게 컨트롤을 돌려준다). CV_GRAPH_ALL_ITEMS(모든 이벤트에 주목)인가, 이하의 플래그의 편성.

- CV_GRAPH_VERTEX - 처음 액세스 한 정점에서 정지한다
- CV_GRAPH_TREE_EDGE - tree edge 그리고 정지한다 (tree edge (은)는 마지막에 액세스 한 정점과 다음에 액세스 되는 정점을 접속하는 옆)
- CV_GRAPH_BACK_EDGE - back edge 그리고 정지한다 (back edge (은)는 마지막에 액세스 한 정점과 탐색 키우치로의 그 부모의 어느쪽이든을 접속하는 옆)

- CV_GRAPH_FORWARD_EDGE - forward edge 그리고 정지한다 (forward edge (은)는 마지막에 액세스 한 정점과 탐색 키우치로의 그 아이의 어느쪽이든을 접속하는 옆. forward edge (은)는 유향그래프의 주사에 대해서만 유효)
- CV_GRAPH_CROSS_EDGE - cross edge 그리고 정지한다 (cross edge (은)는 다른 탐색목끼리인가 같은 나뭇가지끼리를 접속하는 옆. cross edges (은)는 유향그래프의 주사에 대해서만 유효)
- CV_GRAPH_ANY_EDGE - 임의의 옆에서 정지한다(tree, back, forward 그리고 cross edges)
- CV_GRAPH_NEW_TREE - 모든 새로운 탐색목의 선두에서 정지한다. 주사 프로세스가, 주사 개시 정점(initial vertex)(으)로부터 도달 가능한 모든 정점과 옆에 액세스 하는 경우 (액세스 된 정점은 옆과 함께 하나의 나무(tree)(을)를 구성한다), 그래프중에서 아직 액세스 되어 있지 않은 정점을 탐색해, 그 정점으로부터의 탐색을 재개한다.새로운 나무의 개시전(최초로 cvNextGraphItem 하지만 불린다, 완전히 첫 경우도 포함한다)에, CV_GRAPH_NEW_TREE 이벤트를 생성한다.
  무향그래프에 대해서는, 각 탐색목이, 그래프의 접속되는 하나의 성분에 대응한다.
- CV_GRAPH_BACKTRACKING - 역탐색(백 트랙킹, 이미 액세스 된 정점을 반대로 돌아오고 간다) 중에, 모든 액세스 끝난 정점에서 정지한다.

함수 cvCreateGraphScanner (은)는 깊이, 우선 주사/탐색을 위한 구조체를 생성한다. 초기화된 구조체는, 함수 cvNextGraphItem (순서대로 주사 처리)(으)로 이용된다.

NextGraphItem

그래프 주사 처리를 1 스텝, 혹은 수스텝 진행한다

```
int cvNextGraphItem( CvGraphScanner* scanner );
```

scanner

그래프 주사 상태.이 함수로 갱신된다.

함수 cvNextGraphItem (은)는, 유저가 주목한 이벤트 (cvCreateGraphScanner 의 mask 인수로 지정된다) 에 합치하는지, 모든 주사가 종료할 때까지, 그래프내를 순서에 주사 한다. 전자의 경우, 이 함수는 상기의 mask 파라미터로 지정된 이벤트를 돌려주어, 다음의 호출로 주사를 재개한다. 후자의 경우,CV_GRAPH_OVER (-1) (을)를 돌려준다. 이벤트가 CV_GRAPH_VERTEX,CV_GRAPH_BACKTRACKING,CV_GRAPH_NEW_TREE 의 경우, 현재의 주사중의 정점은 scanner->vtx 에 보존된다. 옆에 관계하고 있는 이벤트의 경우, 현재 주사 대상의 옆은 scanner->edge 에, 하나전에 액세스 한 정점은 scanner->vtx 에, 옆의 이제(벌써) 한편(종점)의 정점은 scanner->dst 에 각각 보존된다.

ReleaseGraphScanner

그래프의 주사 처리를 종료한다

```
void cvReleaseGraphScanner( CvGraphScanner** scanner );
```

scanner

스캐너에의 포인터의 포인터.

함수 cvGraphScanner (은)는, 그래프의 주사 처리를 종료해, 스캐너를 해방한다.

3-5 트리 함수(Trees)

CV_TREE_NODE_FIELDS

트리 노드의 종류를 선언하기 위한 보조 매크로

```
#define CV_TREE_NODE_FIELDS(node_type)                          ₩
    int       flags;         /* 여러가지 플래그 */              ₩
    int       header_size;   /* 순서 헤더의 사이즈 */      ₩
    struct    node_type* h_prev; /* 하나전의 순서에의 포인터 */       ₩
    struct    node_type* h_next; /* 하나 후의 순서에의 포인터 */        ₩
    struct    node_type* v_prev; /* 하나전의 순서에의 포인터(세칸다리, 구조에 따라서
의미가 다르다) */ ₩
    struct    node_type* v_next; /* 하나 후의 순서에의 포인터(세칸다리, 구조에 따라서
의미가 다르다) */
```

매크로 CV_TREE_NODE_FIELDS()(은)는, 모든 동적 구조의 기본 타입이다 CvSeq(와)과 같이, 계층 구조(트리)로서 조직화할 수 있는 데이터 구조를 선언하기 위해서 사용된다. 이 매크로를 이용해 선언된 노드로부터 구성되는 트리는, 이 섹션으로 말하는 각 함수를 이용해 처리할 수 있다.

CvTreeNodeIterator

트리 노드의 이테레이타를 위한 구조체

```
typedef struct CvTreeNodeIterator
{
    const void* node;
    int level;
    int max_level;
}
CvTreeNodeIterator;
```

구조체 CvTreeNodeIterator (은)는, 트리를 주사 하기 위해서 이용된다. 트리의 노드의 선언에는, 매크로 CV_TREE_NODE_FIELDS(...)(을)를 이용할 필요가 있다.

InitTreeNodeIterator

트리 노드의 이테레이타를 초기화한다

```
void cvInitTreeNodeIterator( CvTreeNodeIterator* tree_iterator,
                             const void* first, int max_level );
```

tree_iterator

초기화되는 트리의 이테레이타.

first

선두 노드.여기로부터 주사를 개시한다.

max_level

트리 주사 범위의 최대 레벨(first 노드가 제1레벨이라고 가정한다). 예를 들면,1 하first(와)과 같은 레벨의 노드만이 처리되는 것을 의미하는, 또,2하first(와)과 같은 레벨의 노드와 그 아이 노드가 처리된다.

함수 cvInitTreeNodeIterator (은)는, 트리의 이테레이타를 초기화한다.트리는 깊이 우선으로 주사된다.

NextTreeNode

현재의 노드를 돌려주어, 이테레이타를 다음의 노드에 이동시킨다.

```
void* cvNextTreeNode( CvTreeNodeIterator* tree_iterator );
```

tree_iterator

초기화되는 트리의 이테레이타.

함수 cvNextTreeNode (은)는, 현재 대상이 되고 있는 노드를 돌려주어, 그 후 이테레이타를 갱신한다(다음의 노드에 이동시킨다). 즉, 이 함수의 동작은, 통상의 C 언어의 포인터로의 *p++ 표현, 혹은 C++의 코레크션이테레이타와 거의 같다. 더 이상의 노드가 없는 경우, 이 함수는 NULL(을)를 돌려준다.

PrevTreeNode

현재의 노드를 돌려주어, 이테레이타를 전의 노드에 이동시킨다.

```
void* cvPrevTreeNode( CvTreeNodeIterator* tree_iterator );
```

tree_iterator

초기화되는 트리의 이테레이타.

함수 cvPrevTreeNode (은)는 현재 대상이 되고 있는 노드를 돌려주어, 그 후 이테레이타를 갱신한다(하나전의 노드에 이동시킨다). 즉, 이 함수의 동작은, 통상의 C 의 포인터로의 *p-- 표현, 혹은 C++의 코레크션이테레이타와 거의 같다. 더 이상의 노드가 없는 경우, 함수는 NULL(을)를 돌려준다.

TreeToNodeSeq

모든 노드에의 포인터를 하나의 순서에 모은다

```
CvSeq* cvTreeToNodeSeq( const void* first, int header_size, CvMemStorage* storage );
```

first

트리의 선두 노드.

header_size

작성한 순서의 헷다사이즈(sizeof(CvSeq) 하지만 이용되는 것이 많다).

storage

순서를 위한 컨테이너.

함수 cvTreeToNodeSeq (은)는, 노드 first (으)로부터 도달 가능한 모든 노드에의 포인터를 하나의 순서에 정리한다. 포인터는 깊이 우선순서에 차례차례 써진다.

InsertNodeIntoTree

트리에 새로운 노드를 추가한다

```
void cvInsertNodeIntoTree( void* node, void* parent, void* frame );
```

node

삽입되는 노드.

parent

트리내에 이미 존재하고 있는 친노드.

frame

톱 레벨 노드.parent (와)과 frame 하지만 같은 경우, node의v_prev필드에는,parent (이)가 아니고,NULL하지만 세트 된다.

함수 cvInsertNodeIntoTree (은)는, 트리에 다른 노드를 추가한다. 이 함수는 메모리의 파티션을 실시하지 않고, 단지 트리 노드의 링크를 변경하는 것만으로 있다.

RemoveNodeFromTree

트리로부터 노드를 삭제한다

```
void cvRemoveNodeFromTree( void* node, void* frame );
```

node

삭제되는 노드.

frame

톱 레벨 노드.node->v_prev = NULL 한편 node->h_prev = NULL (즉,node 하지만 frame의 최초의 아이 노드이다)인 경우, frame->v_next 하 node->h_next 에 세트 된다 (즉, 최초의 아이 노드인가frame하지만 변경된다).

함수 cvRemoveNodeFromTree (은)는, 트리로부터 노드를 삭제한다. 이 함수는 메모리의 영역 해방을 실시하지 않고, 단지 트리 노드의 링크를 변경하는 것만으로 있다.

4.그리기 함수(Drawing Functions)

그리기 함수는 행렬이나 이미지, 임의의 카라데프스로 사용된다. 에일리어징 제거 처리는 8 비트 이미지에게만 실장되고 있다. 모든 함수는 파라미터 color(CV_RGB 매크로인가,cvScalar 함수를

이용해 작성된다)를 가져, 이것은 칼라 이미지에서는 rgb 값, 그레이 스케일 이미지에서는 휘도를 의미한다.

그리는 이미지의 일부, 혹은 모두가 이미지의 영역외에 있는 경우에는, 그 부분은 잘라내진다. 칼라 이미지의 경우, 채널의 차례는청,록,적. . .이다. 차례를 변경하고 싶은 경우는,cvScalar(을)를 이용해 차례를 지정하는지,cvCvtColor 혹은 cvTransform(을)를 이용하고, 그리기 전, 혹은 그린 후에 이미지를 변환하면 좋다.

4-1 곡선과 형상(Curves and Shapes)

CV_RGB

칼라치를 작성한다

```
#define CV_RGB( r, g, b )  cvScalar( (b), (g), (r) )
```

Line

2 점을 묶는 선분을 그리기 한다

```
void cvLine( CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color,
             int thickness=1, int line_type=8, int shift=0 );
```

img

이미지.

pt1

선분의1번째의 단 점.

pt2

선분의2번째의 단 점.

color

선분의 색.

thickness

선분의 굵기.

line_type

선분의 종류.

8 (또는0) - 8연결에 의한 선분.

4 - 4연결에 의한 선분.

CV_AA - 에일리어징 제거 된 선분.

shift

좌표의 소수점 이하의 자리수를 나타내는 비트수.

함수 cvLine(은)는, 점 pt1(와)과 점 pt2(을)를 묶는 선분을 이미지상에 그리기 한다. 그리기 되는 선분은 이미지나 ROI 에 의해 잘라내진다. 정수치 좌표로 에일리어징 제거 되어 있지 않은 선분에서는,8 연결, 또는 4 연결의 브레젠함(Bresenham) 알고리즘이 사용된다. 굵은 선분의

구석은 둥글고 그리기 된다.에일리어징 제거 된 선분은, 가우시안(Gaussian) 필터를 이용해 그려진다. 선분의 색을 지정하려면 , 매크로 CV_RGB(r, g, b)(을)를 이용한다.

Rectangle

테두리만, 혹은 전부 칠해진 구형을 그리기 한다

```
void cvRectangle( CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color,
                  int thickness=1, int line_type=8, int shift=0 );
```

img
: 구형이 그리기 되는 이미지.

pt1
: 구형의 하나의 정점.

pt2
: 구형의 반대측의 정점.

color
: 선의 색(RGB), 혹은 휘도(그레이 스케일 이미지).

thickness
: 구형을 그리는 선의 굵기.부의 값, 예를 들면CV_FILLED(을)를 지정했을 경우는 전부 칠해진다.

line_type
: 선의 종류, 자세한 것은cvLine(을)를 참조.

shift
: 좌표의 소수점 이하의 자리수를 나타내는 비트수.

함수 cvRectangle(은)는, 점 pt1(와)과 점 pt2(을)를 대각으로 하는 구형을 그리기 한다.

Circle

엔을 그리기 한다

```
void cvCircle( CvArr* img, CvPoint center, int radius, CvScalar color,
               int thickness=1, int line_type=8, int shift=0 );
```

img
: 엔이 그리기 되는 이미지.

center
: 엔의 중심.

radius
: 엔의 반경.

color
: 엔의 색.

thickness
: 선의 폭.부의 값을 지정했을 경우는 전부 칠해진다.

line_type

선의 종류, 자세한 것은cvLine(을)를 참조.

shift

중심 좌표와 반경의 소수점 이하의 자리수를 나타내는 비트수.

함수 cvCircle(은)는, 준 중심과 반경에 따라 테두리만, 혹은 전부 칠해진 엔을 그린다.그리기 되는 엔은 ROI 에 의해서 잘라내진다. 엔의 색은 매크로 CV_RGB (r, g, b)그리고 지정할 수 있다.

Ellipse

테두리만의 타원, 타원호, 혹은 전부 칠해진 선형의 타원을 그리기 한다

```
void cvEllipse( CvArr* img, CvPoint center, CvSize axes, double angle,
                double start_angle, double end_angle, CvScalar color,
                int thickness=1, int line_type=8, int shift=0 );
```

img

타원이 그리기 되는 이미지.

center

타원의 중심.

axes

타원의 축의 길이.

angle

회전 각도.

start_angle

타원호의 개시 각도.

end_angle

타원호의 종료 각도.

color

타원의 색.

thickness

타원호의 선의 폭.

line_type

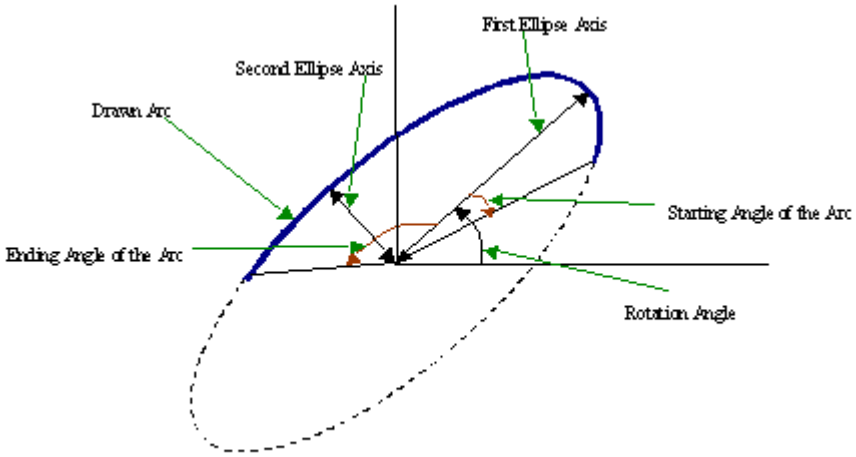
타원호의 선의 종류.자세한 것은cvLine(을)를 참조.

shift

중심 좌표와 축의 길이의 소수점 이하의 자리수를 나타내는 비트수.

함수 cvEllipse(은)는, 테두리만의 타원, 타원호, 혹은 전부 칠한 타원을 그리기 한다. 그리기 되는 타원은 ROI 에 의해서 잘라내진다. 에일리어징 제거 된 선이나 굵은 선을 지정했을 경우에는, 선형 근사가 이용된다.각도는 번으로 지정한다. 파라미터의 의미를 이하의 이미지로 설명한다.

타원호를 그리기하기 위한 파라미터



EllipseBox

테두리만의 타원, 혹은 전부 칠해진 타원을 그리기 한다

```
void cvEllipseBox( CvArr* img, CvBox2D box, CvScalar color,
                   int thickness=1, int line_type=8, int shift=0 );
```

img

타원이 그려지는 이미지.

box

그리기 하고 싶은 타원을 둘러싸는 구형 영역.

thickness

타원 경계선의 폭.

line_type

타원 경계선의 종류.자세한 것은cvLine(을)를 참조.

shift

구형 영역의 정점 좌표의 소수점 이하의 자리수를 나타내는 비트수.

함수 cvEllipseBox(은)는 테두리만의 타원, 혹은 전부 칠해진 타원을 그리기 한다. 이 함수는,cvCamShift(이)나 cvFitEllipse 그리고 얻을 수 있던 결과로부터 타원을 그리기 할 때에 편리하다. 그리기 되는 타원은 ROI 에 의해서 잘라내진다.에일리어징 제거 된 선이나 굵은 선을 지정했을 경우에는, 선형 근사가 이용된다.

FillPoly

다각형 내부를 전부 칠한다

```
void cvFillPoly( CvArr* img, CvPoint** pts, int* npts, int contours,
                 CvScalar color, int line_type=8, int shift=0 );
```

img

다각형이 그려지는 이미지.

pts

다각형에의 포인터 배열.

npts

다각형 정점수의 배열.

contours

전부 칠해진 영역을 단락짓는 윤곽의 수.

color

다각형의 색.

line_type

선의 종류.자세한 것은cvLine(을)를 참조.

shift

정점 좌표의 소수점 이하의 자리수를 나타내는 비트수.

함수 cvFillPoly(은)는, 몇개의 다각형에 의해 단락지어진 영역을 전부 칠한다. 이 함수에서는 복잡한 영역, 예를 들면 영역내에 구멍을 가지는 것이나, 자기 교차한 것 등도 전부 칠한다.

FillConvexPoly

철다각형을 전부 칠한다

```
void cvFillConvexPoly( CvArr* img, CvPoint* pts, int npts,
                       CvScalar color, int line_type=8, int shift=0 );
```

img

다각형이 그려지는 이미지.

pts

하나의 다각형에의 포인터의 배열.

npts

다각형의 정점수.

color

다각형의 색.

line_type

선의 종류.자세한 것은cvLine(을)를 참조.

shift

정점 좌표의 소수점 이하의 자리수를 나타내는 비트수.

함수 cvFillConvexPoly(은)는 철다각형의 내부를 전부 칠한다. 이 함수는, 함수 cvFillPoly 보다 고속으로 동작한다. 또, 철다각형 뿐만이 아니라, 그 윤곽이 수평인 스캔 라인과 2 회이하 밖에 교차하지 않는 듯한 단순한 다각형은 모두 전부 칠할 수 있다.

PolyLine

polyline(테두리만의 다각형)를 그리기 한다

```
void cvPolyLine( CvArr* img, CvPoint** pts, int* npts, int contours, int is_closed,
                 CvScalar color, int thickness=1, int line_type=8, int shift=0 );
```

img
polyline가 그려지는 이미지.
pts
polyline에의 포인터의 배열.
npts
polyline의 정점수의 배열.
contours
polyline의 개수.
is_closed
polyline를 닫을지를 지정한다.닫는 경우, 각각의 영역의 마지막 정점과 최초의 정점을 묶는 선분을 그리기 한다.
color
선의 색.
thickness
선의 굵기.
line_type
선의 종류.자세한 것은cvLine(을)를 참조.
shift
정점 좌표의 소수점 이하의 자리수를 나타내는 비트수.
함수 cvPolyLine(은)는, 하나, 혹은 복수의 polyline 를 그리기 한다.

4-2 문자열 (Text)

InitFont

폰트 구조체를 초기화한다

```
void cvInitFont( CvFont* font, int font_face, double hscale,
                 double vscale, double shear=0,
                 int thickness=1, int line_type=8 );
```

font
이 함수로 초기화되는 폰트 구조체에의 포인터
font_face
폰트명의 식별자.현재는,Hershey fonts (http://sources.isc.org/utils/misc/hershey-font.txt) 의 일부만 서포트되고 있다.
CV_FONT_HERSHEY_SIMPLEX - 보통 사이즈의sans-serif 폰트
CV_FONT_HERSHEY_PLAIN - 작은 사이즈의sans-serif 폰트
CV_FONT_HERSHEY_DUPLEX - 보통 사이즈의sans-serif 폰트 (CV_FONT_HERSHEY_SIMPLEX 보다 복잡)
CV_FONT_HERSHEY_COMPLEX - 보통 사이즈의serif 폰트

CV_FONT_HERSHEY_TRIPLEX - 보통 사이즈의serif 폰트 (CV_FONT_HERSHEY_COMPLEX 보다 복잡)
CV_FONT_HERSHEY_COMPLEX_SMALL - CV_FONT_HERSHEY_COMPLEX (이)가 작은 버젼
CV_FONT_HERSHEY_SCRIPT_SIMPLEX - 자필 스타일의 폰트
CV_FONT_HERSHEY_SCRIPT_COMPLEX - CV_FONT_HERSHEY_SCRIPT_SIMPLEX 의 복잡한 버젼
파라미터는 상기의 값과 이탤릭 혹은 사자를 의미하는 옵션CV_FONT_ITALIC플래그를 합성할 수 있다.

hscale

폭의 비율.1.0f(으)로 했을 경우, 문자는 각각의 폰트에 의존하는 원래의 폭으로 표시된다. 0.5f(으)로 했을 경우, 문자는 원래의 반의 폭으로 표시된다.

vscale

높이의 비율.1.0f(으)로 했을 경우, 문자는 각각의 폰트에 의존하는 원래의 높이로 표시된다. 0.5f(으)로 했을 경우, 문자는 원래의 반의 높이로 표시된다.

shear

수직선으로부터의 문자의 상대적인 각도.제로의 경우는 비이탤릭 폰트로, 예를 들면,1.0f하≈45°(을)를 의미한다.

thickness

문자의 굵기.

line_type

선의 종류.자세한 것은cvLine(을)를 참조.

함수 cvInitFont(은)는, 문자 그리기 함수에게 건네지는 폰트 구조체를 초기화한다.

PutText

문자열을 그리기 한다

```
void cvPutText( CvArr* img, const char* text, CvPoint org, const CvFont* font, CvScalar color );
```

img

입력 이미지.

text

그리기 하는 문자열.

org

최초의 문자의 좌하의 좌표.

font

폰트 구조체에의 포인터.

color

문자의 색.

함수 cvPutText(은)는, 지정한 폰트와 색으로 문자열을 이미지중에 그리기 한다. 그리기 된 문자는 ROI 에 의해서 잘라내진다.지정한 폰트에 포함되지 않는 기호는 사각으로 옮겨놓을 수 있다.

GetTextSize

문자열의 폭과 높이를 취득한다

```
void cvGetTextSize( const char* text_string, const CvFont* font, CvSize* text_size, int* baseline );
```

text_string

입력 문자열.

font

폰트 구조체에의 포인터.

text_size

결과적으로 얻을 수 있는 문자열의 사이즈.문자의 높이에는, baseline보다 아래의 부분의 높이는 포함되지 않는다.

baseline

문자의 최하점에서 본 baseline의y좌표.

함수 cvGetTextSize(은)는, 지정한 폰트로의 입력 문자열을 포함 하는 구형을 계산한다.

4-3 점집합과 윤곽(Point Sets and Contours)

DrawContours

이미지의 외측 윤곽선, 또는 안쪽 윤곽선을 그리기 한다

```
void cvDrawContours( CvArr *img, CvSeq* contour,
                     CvScalar external_color, CvScalar hole_color,
                     int max_level, int thickness=1,
                     int line_type=8, CvPoint offset=cvPoint(0,0) );
```

img

윤곽을 그리기 하는 원이미지.윤곽은ROI그리고 잘라내진다.

contour

최초의 윤곽에의 포인터.

external_color

외측 윤곽선의 색.

hole_color

안쪽 윤곽선(구멍)의 색.

max_level

그리기 되는 윤곽의 최대 레벨. 0(으)로 했을 경우,contour만이 그리기 된다. 1(으)로 했을 경우, 선두의 윤곽과 동레벨의 모든 윤곽이 그리기 된다. 2(으)로 했을 경우, 선두의 윤곽과 동레벨의 모든 윤곽과 선두의 윤곽의 하나하의 레벨의 모든 윤곽이 그리기 된다.이하 같이. 부의 값으로 했을 경우, 이 함수는contour의 후에 계속 되는 동레벨의 윤곽을 그리기 하지 않지만, contour의 아이의 윤곽을abs(max_level)-1의 레벨까지 그리기 한다.

thickness

그리기 되는 윤곽선의 굵기.부(예를 들면=CV_FILLED)(으)로 했을 경우에는, 내부를 전부 칠한다.

line_type

선의 종류.자세한 것은cvLine(을)를 참조.

offset

모든 좌표를 지정한 값만 시프트 한다.이것은, 윤곽을ROI안의 이미지로부터 생성하고,ROI의 오프셋을 고려할 필요가 있는 경우에 편리하다.

함수 cvDrawContours(은)는,thickness>=0 의 경우에는 외측 윤곽선을 이미지상에 그리기 해, thickness<0 의 경우에는 윤곽의 내부를 전부 칠한다.

(예) 윤곽 함수를 이용한 연결 요소의 검출

```
#include "cv.h"
#include "highgui.h"

int main( int argc, char** argv )
{
    IplImage* src;
    // 커멘드 라인의 제 1 파라미터는 2 치(흑백)이미지의 파일명을 지정한다
    if( argc == 2 && (src=cvLoadImage(argv[1], 0))!= 0)
    {
        IplImage* dst = cvCreateImage( cvGetSize(src), 8, 3 );
        CvMemStorage* storage = cvCreateMemStorage(0);
        CvSeq* contour = 0;

        cvThreshold( src, src, 1, 255, CV_THRESH_BINARY );
        cvNamedWindow( "Source", 1 );
        cvShowImage( "Source", src );

        cvFindContours( src, storage, &contour,
                        sizeof(CvContour), CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE );
        cvZero( dst );

        for( ; contour != 0; contour = contour->h_next )
        {
            CvScalar color = CV_RGB( rand()&255, rand()&255, rand()&255 );
            /* 윤곽을 보기 위해서는,CV_FILLED(을)를 1(으)로 한다 */
            cvDrawContours( dst, contour, color, color, -1, CV_FILLED, 8 );
        }

        cvNamedWindow( "Components", 1 );
        cvShowImage( "Components", dst );
        cvWaitKey(0);
    }
}
```

윤곽을 보기 위해서는, 상술의 샘플의 CV_FILLED (을)를 1 에 옮겨놓는다.

InitLineIterator

라인이테레이타를 초기화한다

```
int cvInitLineIterator( const CvArr* image, CvPoint pt1, CvPoint pt2,
                        CvLineIterator* line_iterator, int connectivity=8,
                        int left_to_right=0 );
```

image

대상 이미지.

pt1

선분의 첫째의 단 점.

pt2

선분의 니 번째의 단 점.

line_iterator

라인이테레이타 상태 구조체에의 포인터.

connectivity

주사 한 선분의 접속성.4또는8.

left_to_right

pt1(와)과pt2(와)는 관계없는 것으로 선분을 언제나 왼쪽에서 오른쪽으로 주사 한다(left_to_right≠0)인가, pt1(으)로부터pt2에의 정해진 방향으로 주사 할까(left_to_right=0)(을)를 지정하는 플래그.

함수 cvInitLineIterator(은)는 라인이테레이타를 초기화해,2 단 점간의 픽셀수를 돌려준다. 양쪽 모두의 점은 모두 이미지내에 존재해야 한다. 라인이테레이타가 초기화된 후,2 단 점을 묶는 선분상의 모든 점은 CV_NEXT_LINE_POINT 에 의해 취득된다. 선상의 점은,4 연결 또는 8 연결 브레젠함(Bresenham) 알고리즘을 이용해 한개씩 계산된다.

(예) 라인이테레이타를 이용한 색이 붙은 선상의 픽셀치의 총화의 계산

```
CvScalar sum_line_pixels( IplImage* image, CvPoint pt1, CvPoint pt2 )
{
    CvLineIterator iterator;
    int blue_sum = 0, green_sum = 0, red_sum = 0;
    int count = cvInitLineIterator( image, pt1, pt2, &iterator, 8, 0 );

    for( int i = 0; i < count; i++ ){
        blue_sum += iterator.ptr[0];
        green_sum += iterator.ptr[1];
        red_sum += iterator.ptr[2];
        CV_NEXT_LINE_POINT(iterator);

        /* 픽셀의 좌표를 표시 : 어떻게 좌표를 계산하는지, 의 데먼스트레이션 */
        {
        int offset, x, y;
        /* ROI(은)는 설정되어 있지 않으면 가정하고 있다.만약 설정되어 있는 경우에는
고려할 필요가 있다.*/
        offset = iterator.ptr - (uchar*)(image->imageData);
```

```
            y = offset/image->widthStep;
            x = (offset - y*image->widthStep)/(3*sizeof(uchar) /* 픽셀의 사이즈 */);
            printf("(%d,%d)₩n", x, y );
            }
        }
        return cvScalar( blue_sum, green_sum, red_sum );
    }
```

ClipLine

선분을 이미지 영역에서 잘라낸다

```
int cvClipLine( CvSize img_size, CvPoint* pt1, CvPoint* pt2 );
```

img_size

이미지의 크기.

pt1

선분의1번째의 단 점.이 값은 이 함수에 의해서 변경된다.

pt2

선분의2번째의 단 점.이 값은 이 함수에 의해서 변경된다.

함수 cvClipLine (은)는, 이미지 영역내에 존재하는 선분의 영역을 계산한다. 만약 선분이 완전하게 이미지외라면 0(을)를 돌려주어, 그렇지 않으면 1(을)를 돌려준다.

Ellipse2Poly

타원호를 polyline 로 근사 한다

```
int cvEllipse2Poly( CvPoint center, CvSize axes,
                    int angle, int arc_start,
                    int arc_end, CvPoint* pts, int delta );
```

center

호의 중심.

axes

타원의 축의 길이.cvEllipse(을)를 참조.

angle

타원의 회전 각도.cvEllipse(을)를 참조.

start_angle

타원호의 개시 각도.

end_angle

타원호의 종료 각도.

pts

이 함수로 전부 칠해지는 점의 배열.

delta

polyline가 연속한 정점간의 각도, 근사 정도.출력되는 점의 총수는 최대로 ceil((end_angle - start_angle)/delt

a) + 1.

함수 cvEllipse2Poly(은)는, 지정한 타원호를 근사 하는 polyline 의 정점을 계산한다. 이것은 cvEllipse 그리고 이용된다.

5.데이타 영속성과 실행시간형 정보(Data Persistence and RTTI)

5-1 파일 저장(File Storage)

CvFileStorage

파일 스토리지

```
typedef struct CvFileStorage
{
    ...       // 은폐 필드
} CvFileStorage;
```

구조체 CvFileStorage(은)는, 디스크내에 있는 파일과 관련지을 수 있었던 파일 스토리지의 「블랙 박스」적인 표현이다. 후술 하는 여러가지 함수는 CvFileStorage(을)를 인수에 가져, 스칼라치로 구성되는 계층적인 콜렉션이나, 행렬이나 순서, 그래프등의 표준적인 CXCore 오브젝트, 유저 정의 오브젝트등의 세이브와 로드를 유저에게 제공한다.

CXCore(은)는,XML(http://www.w3c.org/XML)(이)나 YAML (http://www.yaml.org)형식의 데이터의 읽고 쓰기가 가능하다. 이하는 XML(와)과 YAML 에 의한 부동 소수점형의 3×3 단위행렬 A 의 표현이다.CXCore 의 함수를 이용하면 다음과 같이 된다.

XML:

```
<?xml version="1.0">
<opencv_storage>
<A type_id="opencv-matrix">
  <rows>3</rows>
  <cols>3</cols>
  <dt>f</dt>
  <data>1. 0. 0. 0. 1. 0. 0. 0. 1.</data>
</A>
</opencv_storage>
```

YAML:

```
%YAML:1.0
A: !!opencv-matrix
```

```
        rows: 3
        cols: 3
        dt: f
        data: [ 1., 0., 0., 0., 1., 0., 0., 0., 1.]
```
이 예로 볼 수 있도록(듯이),XML 그럼 계층 구조의 표현에 네스트(상자)된 태그를
이용해 YAML 그럼(Python(와)과 유사한) 인덴트를 이용한다.

같다 CXCore 함수를 이용해 양데이터 형식의 읽고 쓰기가 가능하고, 형식은 열린 파일의
확장자(extension)로 판별된다. 확장자(extension)는 XML 의 경우.xml,YAML 의 경우는.yml 이다.

CvFileNode

파일 스토리지 노드

```
/* 파일 노드 타입 */
#define CV_NODE_NONE        0
#define CV_NODE_INT         1
#define CV_NODE_INTEGER     CV_NODE_INT
#define CV_NODE_REAL        2
#define CV_NODE_FLOAT       CV_NODE_REAL
#define CV_NODE_STR         3
#define CV_NODE_STRING      CV_NODE_STR
#define CV_NODE_REF         4 /* 사용되지 않는다 */
#define CV_NODE_SEQ         5
#define CV_NODE_MAP         6
#define CV_NODE_TYPE_MASK   7

/* 옵션의 플래그 */
#define CV_NODE_USER        16
#define CV_NODE_EMPTY       32
#define CV_NODE_NAMED       64

#define CV_NODE_TYPE(tag)  ((tag) & CV_NODE_TYPE_MASK)

#define CV_NODE_IS_INT(tag)        (CV_NODE_TYPE(tag) == CV_NODE_INT)
#define CV_NODE_IS_REAL(tag)       (CV_NODE_TYPE(tag) == CV_NODE_REAL)
#define CV_NODE_IS_STRING(tag)     (CV_NODE_TYPE(tag) == CV_NODE_STRING)
#define CV_NODE_IS_SEQ(tag)        (CV_NODE_TYPE(tag) == CV_NODE_SEQ)
#define CV_NODE_IS_MAP(tag)        (CV_NODE_TYPE(tag) == CV_NODE_MAP)
#define CV_NODE_IS_COLLECTION(tag) (CV_NODE_TYPE(tag) >= CV_NODE_SEQ)
#define CV_NODE_IS_FLOW(tag)       (((tag) & CV_NODE_FLOW) != 0)
#define CV_NODE_IS_EMPTY(tag)      (((tag) & CV_NODE_EMPTY) != 0)
#define CV_NODE_IS_USER(tag)       (((tag) & CV_NODE_USER) != 0)
#define CV_NODE_HAS_NAME(tag)      (((tag) & CV_NODE_NAMED) != 0)
```

```
#define CV_NODE_SEQ_SIMPLE 256
#define CV_NODE_SEQ_IS_SIMPLE(seq) (((seq)->flags & CV_NODE_SEQ_SIMPLE) != 0)

typedef struct CvString
{
    int len;
    char* ptr;
}
CvString;

/* 읽어들인 파일 스토리지에 포함되는 요소의 모든 키(names)(은)는,
   룩 업 작업의 고속화를 위해 해시에 기억된다 */
typedef struct CvStringHashNode
{
    unsigned hashval;
    CvString str;
    struct CvStringHashNode* next;
}
CvStringHashNode;

/* 파일 스토리지의 기본 요소 - 스칼라, 또는 콜렉션 */
typedef struct CvFileNode
{
    int tag;
    struct CvTypeInfo* info; /* 형태 정보(유저 정의 오브젝트용, 그렇지 않은 경우는 0) */
    union
    {
        double f; /* 부동 소수점의 스칼라치 */
        int i;    /* 정수의 스칼라치 */
        CvString str; /* 문자열 */
        CvSeq* seq; /* 순서(정렬된 파일 노드의 콜렉션) */
        struct CvMap* map; /* 맵(이름 붙여 된 파일 노드의 콜렉션) */
    } data;
}
CvFileNode;
```

이 구조체는, 파일 스토리지로부터의 데이터 읽어들이기에게만 이용된다(예를 들면 파일로부터의 데이터의 로드). 데이터가 파일에 써지는 경우, 최소한의 버퍼링으로 연속적으로 실행된다.이 파일 스토리지에는 일절의 데이터가 보존되지 않는다.

반대로, 파일로부터 데이터가 읽혔을 경우, 모든 파일은 해석되고, 메모리내에서는 나무 구조로서 표현된다. 나무 구조의 각각의 노드는 CvFileNode 그리고 표현된다. 파일 노드 N 의 형태는,CV_NODE_TYPE(N->tag)(으)로서 취득할 수 있다. 몇개의 파일 노드(잎)는, 문자열이나, 정수형치, 부동 소수점형치등의 스칼라이다. 다른 파일 노드는 파일 노드의 콜렉션이며, 그것은 스칼라치인가 그 노드로의 콜렉션이다. 콜렉션에는 순서와 맵의 2 개의 형태가 있다(여기에서는 YAML 의 표기법을 사용하지만,XML 스트림에서도 같이이다). 순서(CvSeq (와)과 혼동 하지 않게)는 이름 붙여 되어 있지 않은 파일 노드의 정렬된 콜렉션으로, 맵은 이름 붙여

된 파일 노드의 정렬되어 있지 않은 콜렉션이다. 이와 같이, 순서의 요소에는, 인덱스(cvGetSeqElem)에 의해 액세스 되어 맵의 요소에는 이름(cvGetFileNodeByName)(으)로 액세스 된다. 이하의 겉(표)는 파일 노드의 형태의 차이를 나타낸다.

형	CV_NODE_TYPE(node->tag)	치
정수	CV_NODE_INT	node->data.i
부동 소수점	CV_NODE_REAL	node->data.f
문자열	CV_NODE_STR	node->data.str.ptr
순서	CV_NODE_SEQ	node->data.seq
맵	CV_NODE_MAP	node->data.map*

*

map 필드에 직접 액세스 할 필요는 없다(CvMap(은)는 숨겨진 구조체이다). 이 맵의 요소는, 「맵」파일 노드의 포인터를 인수에 가지는 함수 cvGetFileNodeByName(을)를 사용해 취득할 수 있다.

유저(커스텀) 오브젝트는 CvMat(이)나 CvSeq 등의 표준적인 CxCore 의 형태인가, cvRegisterType 그리고 등록된 형태의 인스턴스이다. 이러한 오브젝트는, 파일 스토리지가 열려 해석된 후, 초기 상태에서는 파일내에서 맵(상술의 XML(와)과 YAML 의 샘플 파일과 같이)로서 표현되고 있다. 그 후, 오브젝트는 리퀘스트(함수 cvRead(이)나 cvReadByName(을)를 사용했을 경우) 에 따라 디코드(원래의 표현에 변환)된다.

CvAttrList

속성의 리스트

```
typedef struct CvAttrList
{
    const char** attr; /* (attribute_name,attribute_value)의 페어로부터 된다 NULL 그리고 끝나는 배열 */
    struct CvAttrList* next; /* 속성 리스트의 다음의 덩어리에의 포인터 */
}
CvAttrList;

/* 구조체 CvAttrList 의 초기화 */
inline CvAttrList cvAttrList( const char** attr=NULL, CvAttrList* next=NULL );

/* 속성의 값을 돌려주는지, 만약 그러한 속성이 존재하지 않는 경우에는 0(NULL)(을)를 돌려준다 */
const char* cvAttrValue( const CvAttrList* attr, const char* attr_name );
```

현재의 실장에서는, 속성은 유저 오브젝트를 쓸 때의 특별한 파라미터를 건네주기 위해서 사용된다(cvWrite(을)를 참조). 태그의 안에 있다 XML 의 속성이나 오브젝트의 형태 지정(type_id 속성)은 서포트되지 않는다.

OpenFileStorage

데이터 읽고 쓰기를 위한 파일을 오픈한다

```
CvFileStorage* cvOpenFileStorage( const char* filename, CvMemStorage* memstorage, int flags );
```

filename

스토리지에 관련 지을 수 있었던 파일의 이름.

memstorage

일시적인 데이터나,CvSeq(이)나 CvGraph등의 동적 구조체의 보존에 사용되는 메모리스트레이지.NULL의 경우, 일시적인 메모리가 확보되어 사용된다.

flags

이하 중의 하나를 지정.

CV_STORAGE_READ - 데이터 읽을 유익의 파일 오픈

CV_STORAGE_WRITE - 데이터 기입을 위한 파일 오픈

함수 cvOpenFileStorage(은)는, 데이터의 읽고 쓰기를 위한 파일을 연다. 기입의 경우는, 새로운 파일이 작성되는지, 파일이 존재하는 경우에는 덧쓰기된다.또, 읽고 쓰기되는 파일의 종류는 확장자(extension)로 판별된다. *XML* 의 경우는.xml, *YAML* 의 경우는.yml 또는.yaml 이다. 이 함수의 반환값은 구조체 CvFileStorage 에의 포인터이다.

ReleaseFileStorage

파일 스토리지의 해방

```
void cvReleaseFileStorage( CvFileStorage** fs );
```

fs

해방하는 파일에의 포인터의 포인터.

함수 cvReleaseFileStorage(은)는 파일을 닫고, 일시적인 구조체를 모두 해방한다. 이 함수는, 모든 입출력 조작이 종료한 후에 실행되지 않으면 안 된다.

5-2 데이터의 기입(Writing Data)

StartWriteStruct

새로운 구조체의 기입을 개시한다

```
void cvStartWriteStruct( CvFileStorage* fs, const char* name,
                         int struct_flags, const char* type_name=NULL,
```

CvAttrList attributes=cvAttrList());

fs

파일 스토리지.

name

쓰는 구조체의 이름.읽어들이는 경우는, 이 이름으로 구조체에 액세스 할 수 있다.

struct_flags

다음의 값의 편성.

CV_NODE_SEQ - 쓰는 구조체는 순서(CvFileStorage(을)를 참조), 즉 요소는 이름을 가지지 않는다.

CV_NODE_MAP - 쓰는 구조체는 맵(CvFileStorage(을)를 참조), 즉 모든 요소가 이름을 가진다.

이것들 두 개중 하나만을 지정해야 한다.

CV_NODE_FLOW - YAML스트림 에 대해서만 의미를 가지는 옵션 플래그. 구조체는, 보다 컴팩트한flow(block(은)는 아니고)로서 보존된다. 이 플래그는 전요소가 스칼라인 구조체나 배열에 대해서 이용하는 것이 추천된다.

type_name

옵션 파라미터 - 오브젝트의 형태의 이름. XML의 경우, 구조체 개시 태그의type_id속성으로서 쓰여진다. YAML의 경우, 구조체명에 계속 되는 코론의 뒤에 쓰여진다(CvFileStorage의 설명안에 있는 예를 참조). 주로 유저 오브젝트와 함께 사용된다.스토리지가 읽혀졌을 때, encode 된 형명이 오브젝트의 형태를 결정한다 (CvTypeInfo(와)과cvFindType(을)를 참조).

attributes

이 파라미터는 현재는 사용되지 않았다.

함수 cvStartWriteStruct(은)는, 순서나 맵이 복합한 구조체(콜렉션)의 기입을 개시한다. 스칼라나 구조체로 구성된 모든 구조체의 필드가 써진 다음에,cvEndWriteStruct(을)를 실행해야 한다. 이 함수는, 몇개의 오브젝트를 그룹화 하는 경우나, 유저 오브젝트(CvTypeInfo(을)를 참조)의 기입 함수를 실장하는 경우에도 사용되는 일이 있다.

EndWriteStruct

구조체의 기입을 종료한다

```
void cvEndWriteStruct( CvFileStorage* fs );
```

fs

파일 스토리지.

함수 cvEndWriteStruct(은)는, 구조체의 기입을 종료한다.

WriteInt

정수형의 값을 쓴다

```
void cvWriteInt( CvFileStorage* fs, const char* name, int value );
```

fs

파일 스토리지.

name

써지는 값의 이름.부모의 구조체가 순서의 경우는,NULL(으)로 하지 않으면 안 된다.

value

써지는 값.

함수 cvWriteInt(은)는,1 개의 정수치(이름 있어, 또는 없음)를 파일에 쓴다.

WriteReal

부동 소수점형의 값을 쓴다

```
void cvWriteReal( CvFileStorage* fs, const char* name, double value );
```

fs

파일 스토리지.

name

써지는 값의 이름.부모의 구조체가 순서의 경우는,NULL(으)로 하지 않으면 안 된다.

value

써지는 값.

함수 cvWriteReal(은)는, 단정도 부동 소수점형의 값(이름 있어, 또는 없음)을 파일에 쓴다. 특별한 값은 encode 된다 : Not A Number 하 NaN 에,±Infinity 하 +.Inf (-.Inf) (이)가 된다.

이하의 예에서는 새로운 형태의 등록을 행하지 않고, 종료 조건과 같은 독자적인 구조체를 보존하는 저레벨의 기입 함수의 사용법을 나타낸다.

```
void write_termcriteria( CvFileStorage* fs, const char* struct_name,
                         CvTermCriteria* termcrit )
{
    cvStartWriteStruct( fs, struct_name, CV_NODE_MAP, NULL, cvAttrList(0,0));
    cvWriteComment( fs, "termination criteria", 1 ); // 단순한 코멘트
    if( termcrit->type & CV_TERMCRIT_ITER )
        cvWriteInt( fs, "max_iterations", termcrit->max_iter );
    if( termcrit->type & CV_TERMCRIT_EPS )
        cvWriteReal( fs, "accuracy", termcrit->epsilon );
    cvEndWriteStruct( fs );
}
```

WriteString

문자열을 쓴다

```
void cvWriteString( CvFileStorage* fs, const char* name,
                    const char* str, int quote=0 );
```

fs

파일 스토리지.

name

써지는 문자열의 이름.부모의 구조체가 순서의 경우는,NULL(으)로 하지 않으면 안 된다.

str

써지는 문자열.

quote

0이외의 경우, 써지는 문자열은 필요할지에 관련되지 않고 인용부호로 낀다. 0의 경우, 필요한 경우에게만 인용부호가 사용된다(예를 들면, 문자열이 숫자로 시작되어 있거나, 스페이스를 포함한 경우).

함수 cvWriteString(은)는, 문자열을 파일 스토리지에 쓴다.

WriteComment

코멘트를 쓴다

```
void cvWriteComment( CvFileStorage* fs, const char* comment, int eol_comment );
```

fs

파일 스토리지.

comment

일행 또는 복수행의, 써지는 코멘트.

eol_comment

0이외의 경우, 이 함수는 현재의 행의 마지막에 코멘트를 넣으려고 시도한다. 플래그가0그리고, 코멘트가 복수, 또는 현재의 행의 마지막에 들어가지 않는 경우는, 코멘트는 새로운 행으로부터 시작할 수 있다.

함수 cvWriteComment(은)는, 파일 스토리지에 코멘트를 쓴다. 이 코멘트는 디버그나 설명을 기술하기 위해서 사용되는 것으로, 읽기시에는 읽어 날아간다.

StartNextStream

다음의 스트림을 개시한다

```
void cvStartNextStream( CvFileStorage* fs );
```

fs

파일 스토리지.

함수 cvStartNextStream(은)는, 파일 스토리지내의 다음의 스트림을 개시한다. YAML(와)과 XML(은)는 어느쪽이나 복수의 「스트림」을 서포트하고 있다. 이것은 파일의 연결이나 써 프로세스의 재개에 도움이 된다.

Write

유저 오브젝트를 쓴다

```
void cvWrite( CvFileStorage* fs, const char* name,
              const void* ptr, CvAttrList attributes=cvAttrList() );
```

fs

파일 스토리지.

name

써지는 오브젝트의 이름.부모의 구조체가 순서의 경우는,NULL(으)로 하지 않으면 안 된다.

ptr

오브젝트에의 포인터.

attributes

오브젝트의 속성.이것은 특정의 형태에 대해서 고유하다(이하를 참조).

함수 cvWrite(은)는, 오브젝트를 파일 스토리지에 쓴다. 우선 cvTypeOf(을)를 이용해 적절한 형태 정보를 찾아낸다.그리고, 형태 정보의 write 메소드가 불려 간다.

속성은 써 수속을 커스터마이즈 하기 위해서 사용된다.표준의 형태에서는 이하의 속성을 서포트하고 있다 (모든*dt 속성은,cvWriteRawData(와)과 같은 포맷을 가진다).

CvSeq

- header_dt - CvSeq, 또는 CvChain(순서가 프리 맨 최인의 경우), 또는 CvContour (순서가 윤곽이나 점렬의 경우)에 계속 된다, 순서 헤더의 유저 필드의 설명.
- dt - 순서 요소의 설명.
- recursive - 속성이 존재해,「0」에서도「false」도 아닌 경우, 순서(윤곽)의 모든 나무는 보존된다.

CvGraph

그래프 정점의 유저 필드의 설명.

- header_dt - CvGraph 에 계속 되는, 그래프 헤더의 유저 필드의 설명.
- vertex_dt - 그래프 정점의 유저 필드의 설명.
- edge_dt - 그래프 엣지의 유저 필드의 설명(엣지의 중량감은 항상 쓰여지기 위해, 명시적으로 지정할 필요가 없는 것에 주의).

이하는 CvFileStorage 의 해설에 있다 YAML(을)를 생성하는 코드이다.

```
#include "cxcore.h"

int main( int argc, char** argv )
{
    CvMat* mat = cvCreateMat( 3, 3, CV_32F );
    CvFileStorage* fs = cvOpenFileStorage( "example.yml", 0, CV_STORAGE_WRITE );

    cvSetIdentity( mat );
    cvWrite( fs, "A", mat, cvAttrList(0,0) );

    cvReleaseFileStorage( &fs );
    cvReleaseMat( &mat );
    return 0;
}
```

WriteRawData

복수의 수치를 쓴다

```
void cvWriteRawData( CvFileStorage* fs, const void* src,
                     int len, const char* dt );
```

fs

파일 스토리지.

src

쓰는 배열에의 포인터.

len

쓰는 배열의 요소수.

dt

다음에 나타내는 포맷을 가지는 배열의 개개의 요소의 사양. ([count]{'u'|'c'|'w'|'s'|'i'|'f'|'d'})..., 그리고, 이러한 기호는 기본적인C언어의 형태와 같다.

- 'u' - 8 비트 부호 없음수
- 'c' - 8 비트 부호 있어 수
- 'w' - 16 비트 부호 없음수
- 's' - 16 비트 부호 있어 수
- 'i' - 32 비트 부호 있어 수
- 'f' - 단정도 부동 소수점형수
- 'd' - 배정도 부동 소수점형수
- 'r' - 포인터.하위 32 비트는, 부호 있어 정수로서 써진다. 이 형태는, 써지는 요소끼리가 링크 구조를 가지는 구조체를 격납하는데 이용할 수 있다.

count(은)는 있는 형태의 값의 옵션 카운터이다. 예를 들면dt='2if'(은)는, 각각의 배열 요소가2개의 정수, 그 후에 하나의 단정도 부동 소수점수(실수)가 계속 되는 구조인 것을 의미한다. 이 사양과 동등의 표기로서는,'if','2i1f' 등이 있다. 다른 예로서 dt='u'(은)는, 배열이 아르바이트열인 것을 의미해, dt='2d'(은)는, 배열이 배정도 부동 소수점형수의 페어로 구성되는 것을 의미한다.

함수 cvWriteRawData(은)는, 하나 하나의 요소가 복수의 수치의 모임인 배열을 쓴다. 이 함수는 cvWriteInt(와)과 cvWriteReal 의 반복으로 옮겨놓을 수 있지만, 단일의 실행 쪽이 효율적이다. 이름을 가지는 요소가 하나도 없기 때문에, 맵보다는 순서에 써야 하는 것인 것에 주의한다.

WriteFileNode

파일 노드를 다른 파일 스토리지에 쓴다

```
void cvWriteFileNode( CvFileStorage* fs, const char* new_node_name,
                      const CvFileNode* node, int embed );
```

fs

기입처의 파일 스토리지.

new_file_node

기입처 파일 스토리지내의 파일 노드가 새로운 이름. 전의 이름을 유지하기 위해서는,cvGetFileNodeName(node)(을)를 이용한다.

node

써지는 노드.

embed

써지는 노드가 콜렉션으로, 이 파라미터가0(이)가 아닌 경우, 계층의 여분의 레벨은 생성되지 않는다. 그 대신에,node의 모든 요소는 현재 써지고 있는 구조체에 써진다. 당연, 맵 요소는 맵에게만 써져 순서 요소는 순서에게만 써진다.

함수 cvWriteFileNode(은)는, 파일 스토리지에 파일 노드의 카피를 쓴다. 이 함수의 용도로서 몇개의 파일 스토리지를 하나로 정리하는,XML(와)과 YAML 의 포맷을 교환하는, 등을 들 수 있다.

5-3 데이터의 읽기(Reading Data)

데이터는 파일 스토리지로부터 두 개의 단계를 거쳐 받아들여진다.우선 요구된 데이터를 포함한 파일 노드를 탐색해, 그 후에 그 노드로부터 수동, 혹은 특별한 read 메소드를 이용해 수중에 넣는다.

GetRootFileNode

파일 스토리지의 톱 레벨 노드의 하나를 수중에 넣는다

```
CvFileNode* cvGetRootFileNode( const CvFileStorage* fs, int stream_index=0 );
```

fs

파일 스토리지.

stream_index

0(으)로부터 시작되는 스트림의 인덱스. cvStartNextStream(을)를 참조. 많은 경우, 파일중에 존재하는 것은 하나의 스톰이지만, 복수로도 될 수 있다.

함수 cvGetRootFileNode(은)는, 톱 레벨 파일 노드의 하나를 돌려준다. 톱 레벨 노드는 이름을 가지지 않고, 그것들은 스트림에 상당해, 파일 스토리지내에 차례차례로 보존되고 있다. 인덱스가 범위외의 경우, 이 함수는 NULL 포인터를 돌려준다. 이 함수에 stream_index=0,1,...(을)를 순서로 지정해 호출해,NULL 포인터가 돌려주어질 때까지 반복하는 것으로, 모든 톱 레벨 노드를 얻을 수 있다.이 함수는 파일 스토리지의 재귀적인 주사를 위해서 사용된다.

GetFileNodeByName

맵내 또는 파일 스토리지내로부터 노드를 탐색한다

```
CvFileNode* cvGetFileNodeByName( const CvFileStorage* fs,
                                 const CvFileNode* map,
                                 const char* name );
```

fs

파일 스토리지.

map

친맵.NULL의 경우, 이 함수는 제일 최초의 것으로부터 개시하고, 모든 톱 레벨 노드(스트림)안을 탐색한다.

name

파일 노드명.

함수 cvGetFileNodeByName(은)는,name 의 파일 노드를 탐색한다. 노드의 탐색은 map 안에서 행해져 포인터가 NULL 이면, 스토리지의 톱 레벨 파일 노드를 포함해 행해진다. 맵에 대해서 이 함수를, 순서에 대해서 cvGetSeqElem (또는 순서 리더)(을)를 이용하는 것으로, 파일 스토리지의 주사가 가능하게 된다. 어느 키에 대한 복수의 쿠에리를 고속으로 처리하는(예를 들면 구조체의 배열등) 위해(때문에)는,cvGetHashedKey(와)과 cvGetFileNode(을)를 조합해 사용하면 좋다.

GetHashedKey

준 이름에 대한 독특한 포인터를 돌려준다

```
CvStringHashNode* cvGetHashedKey( CvFileStorage* fs, const char* name,
                                  int len=-1, int create_missing=0 );
```

fs

파일 스토리지.

name

노드명.

len

이름의 길이(사전에 알고 있는 경우), 또는 계산할 필요가 있는 경우는-1.

create_missing

absent key(을)를 해시 테이블에 추가할지를 지정하는 플래그.

함수 cvGetHashedKey(은)는, 특정의 파일 노드명에 대한 유일한 포인터를 돌려준다. 이 포인터는 함수 cvGetFileNodeByName 보다 고속의 함수 cvGetFileNode 에 건네줄 수 있다. 후자의 함수는 문자열의 내용을 비교하는 것이 아니라, 포인터의 비교에 의해 문자열을 비교하기 위해(때문에)이다.

다음의 예에서는, 점렬이 두 개의 엔트리를 가지는 맵의 순서로서 나타내지고 있다.예를 들면,

```
%YAML:1.0
points:
  - { x: 10, y: 10 }
  - { x: 20, y: 20 }
  - { x: 30, y: 30 }
  # ...
```

거기서, 해시화 되었다"x"(와)과"y"의 포인터를 얻는 것으로, 점의 디코드를 고속화하는 것이 가능하게 된다.

(예) 파일 스토리지로부터 구조체의 배열을 읽어들인다

```
#include "cxcore.h"

int main( int argc, char** argv )
{
    CvFileStorage* fs = cvOpenFileStorage( "points.yml", 0, CV_STORAGE_READ );
    CvStringHashNode* x_key = cvGetHashedNode( fs, "x", -1, 1 );
    CvStringHashNode* y_key = cvGetHashedNode( fs, "y", -1, 1 );
    CvFileNode* points = cvGetFileNodeByName( fs, 0, "points" );
```

```
    if( CV_NODE_IS_SEQ(points->tag) )
    {
        CvSeq* seq = points->data.seq;
        int i, total = seq->total;
        CvSeqReader reader;
        cvStartReadSeq( seq, &reader, 0 );
        for( i = 0; i < total; i++ )
        {
            CvFileNode* pt = (CvFileNode*)reader.ptr;
#if 1 /* 고속 버젼 */
            CvFileNode* xnode = cvGetFileNode( fs, pt, x_key, 0 );
            CvFileNode* ynode = cvGetFileNode( fs, pt, y_key, 0 );
            assert( xnode && CV_NODE_IS_INT(xnode->tag) &&
                    ynode && CV_NODE_IS_INT(ynode->tag));
            int x = xnode->data.i; // 혹은 x = cvReadInt( xnode, 0 );
            int y = ynode->data.i; // 혹은 y = cvReadInt( ynode, 0 );
#elif 1 /* 저속 버젼.x_key(와)과 y_key(을)를 사용하지 않는다 */
            CvFileNode* xnode = cvGetFileNodeByName( fs, pt, "x" );
            CvFileNode* ynode = cvGetFileNodeByName( fs, pt, "y" );
            assert( xnode && CV_NODE_IS_INT(xnode->tag) &&
                    ynode && CV_NODE_IS_INT(ynode->tag));
            int x = xnode->data.i; // 혹은 x = cvReadInt( xnode, 0 );
            int y = ynode->data.i; // 혹은 y = cvReadInt( ynode, 0 );
#else /* 초저속이지만 사용하기 쉬운 버젼 */
            int x = cvReadIntByName( fs, pt, "x", 0 /* 디폴트치 */ );
            int y = cvReadIntByName( fs, pt, "y", 0 /* 디폴트치 */ );
#endif
            CV_NEXT_SEQ_ELEM( seq->elem_size, reader );
            printf("%d: (%d, %d)₩n", i, x, y );
        }
    }
    cvReleaseFileStorage( &fs );
    return 0;
}
```

맙헤의 액세스 방법으로 관련되지 않고, 단순한 순서를 사용하는 것에 비해, 이것은 아직 꽤저속인 것에 주의하면 좋겠다. 예를 들면 위의 예에 대해서는, 점을 하나의 순서에 있어서의 정수의 페어로서 encode 하는 것(분)이 보다 효율적이다.

GetFileNode

맵 또는 파일 스토리지내의 노드를 찾아낸다

```
CvFileNode* cvGetFileNode( CvFileStorage* fs, CvFileNode* map,
                           const CvStringHashNode* key, int create_missing=0 );
```

fs

파일 스토리지.

map

친맵.NULL의 경우, 이 함수는 톱 레벨 노드를 찾는다.만약map(와)과key의 양쪽 모두가NULL의 경우에는, 이 함수는 톱 레벨 노드를 가지는 맵인 루트 파일 노드를 돌려준다.

key

cvGetHashedKey그리고 취득되는 노드명헤의 유일한 포인터.

create_missing

absent node(을)를 맵에 추가할지를 지정하는 플래그.

함수 cvGetFileNode(은)는, 파일 노드를 찾아낸다. 이것은 cvGetFileNodeByName (cvGetHashedKey(을)를 참조)의 고속 버젼이다. 맵내에 없는 경우, 이 함수는(퍼스를 실시하는 함수가 이용해) 새로운 노드를 삽입하는 것이 가능하다.

GetFileNodeName

파일 노드의 이름을 돌려준다

```
const char* cvGetFileNodeName( const CvFileNode* node );
```

node

파일 노드.

함수 cvGetFileNodeName 파일 노드의 이름을 돌려준다. 파일 노드가 이름을 가지지 않는가,node 하지만 NULL 의 경우에는 NULL(을)를 돌려준다.

ReadInt

파일 노드로부터 정수치를 읽어들인다

```
int cvReadInt( const CvFileNode* node, int default_value=0 );
```

node

파일 노드.

default_value

node하지만NULL의 경우의 반환값.

함수 cvReadInt(은)는, 파일 노드로 표현된 정수치를 돌려준다. 파일 노드가 NULL 의 경우,default_value(을)를 돌려준다 (즉,cvGetFileNode 의 직후에 NULL 포인터의 체크를 실시하지 않고, 이 함수를 사용하면 편리하다). 파일 노드가 CV_NODE_INT 형태를 가지는 경우,node->data.i(을)를 돌려준다. 파일 노드가 CV_NODE_REAL 형태를 가지는 경우,node->data.f(을)를 정수로 변환해 돌려준다. 그 이외의 경우, 반환값은 부정이다.

ReadIntByName

파일 노드를 탐색해, 그 값을 돌려준다

```
int cvReadIntByName( const CvFileStorage* fs, const CvFileNode* map,
                     const char* name, int default_value=0 );
```

fs

파일 스토리지.

map

친맵.NULL의 경우, 이 함수는 톱 레벨 노드를 탐색한다.

name

노드명.

default_value

파일 노드가 발견되지 않는 경우의 반환값.

함수 cvReadIntByName(은)는,cvGetFileNodeByName (와)과 cvReadInt 의 단순한 합성이다.

.

ReadReal

파일 노드로부터 부동 소수점형의 값을 수중에 넣는다

```
double cvReadReal( const CvFileNode* node, double default_value=0. );
```

node

파일 노드.

default_value

node하지만NULL의 경우의 반환값.

함수 cvReadReal(은)는, 파일 노드로 표현되는 부동 소수점형의 값을 돌려준다. 파일 노드가 NULL 의 경우,default_value(을)를 돌려준다 (즉,cvGetFileNode 의 직후에 NULL 포인터의 체크를 행하지 않고, 이 함수를 사용하면 편리하다). 파일 노드가 CV_NODE_REAL 형태를 가지는 경우,node->data.f(을)를 돌려준다. 파일 노드가 CV_NODE_INT 형태를 가지는 경우,node->data.f(을)를 부동 소수점형으로 변환해 돌려준다. 그 이외의 경우, 반환값은 부정이다.

ReadRealByName

파일 노드를 찾아 그 값을 돌려준다

```
double cvReadRealByName( const CvFileStorage* fs, const CvFileNode* map,
                         const char* name, double default_value=0. );
```

fs

파일 스토리지.

map

친맵.NULL의 경우, 이 함수는 톱 레벨 노드를 찾는다.

name

노드명.

default_value

파일 노드가 발견되지 않는 경우의 반환값.

함수 cvReadRealByName(은)는,cvGetFileNodeByName (와)과 cvReadReal 의 단순한 합성이다.

ReadString

파일 노드로부터 문자열을 꺼낸다

```
const char* cvReadString( const CvFileNode* node, const char* default_value=NULL );
```

node

파일 노드.

default_value

node하지만NULL의 경우의 반환값.

함수 cvReadString(은)는, 파일 노드로 표현된 문자열을 돌려준다. 파일 노드가 NULL 의 경우에는 default_value(을)를 돌려준다 (즉,cvGetFileNode 의 직후에 NULL 포인터의 체크를 행하지 않고, 이 함수를 사용하면 편리하다). 파일 노드가 CV_NODE_STR 형태를 가지는 경우에는 node->data.str.ptr(을)를 돌려준다. 그 이외의 경우, 반환값은 부정이다.

ReadStringByName

파일 노드를 탐색하고, 그 값을 돌려준다

```
const char* cvReadStringByName( const CvFileStorage* fs, const CvFileNode* map,
                                const char* name, const char* default_value=NULL );
```

fs

파일 스토리지.

map

친맵.NULL의 경우, 이 함수는 톱 레벨 노드를 탐색한다.

name

노드명.

default_value

파일 노드가 발견되지 않는 경우의 반환값.

함수 cvReadStringByName(은)는,cvGetFileNodeByName(와)과 cvReadString 의 단순한 합성이다.

Read

오브젝트를 디코드해, 그 포인터를 돌려준다

```
void* cvRead( CvFileStorage* fs, CvFileNode* node,
              CvAttrList* attributes=NULL );
```

fs

파일 스토리지.

node

루트 오브젝트 노드.

attributes

사용되지 않는 파라미터.

함수 cvRead(은)는, 유저 오브젝트를 디코드해 (오브젝트를 파일 스토리지의 서브 트리로부터 네이티브인 표현으로 작성), 그것을 돌려준다. 디코드되는 오브젝트는,read 메소드를 서포트하는 등록된 형태의 인스턴스로안 된다 (CvTypeInfo(을)를 참조). 오브젝트의 형태는 파일내에서 encode 된 형명으로 결정된다. 만약 오브젝트가 동적 구조체의 경우, 메모리스트레이지내에도 생성되어 cvOpenFileStorage 에게 건네진다. 여기서 NULL 포인터가 건네받았을 경우, 일시적인 메모리스트레이지 중(안)에서는 cvReleaseFileStorage 하지만 불렸을 때에 해방된다. 만약 오브젝트가 동적 구조체가 아닌 경우, heap 내에 생성되어 특별한 함수나 표준의 cvRelease (을)를 사용해 해방할 필요가 있다.

ReadByName

오브젝트를 탐색해, 디코드한다

```
void* cvReadByName( CvFileStorage* fs, const CvFileNode* map,
                    const char* name, CvAttrList* attributes=NULL );
```

fs

파일 스토리지.

map

친맵.NULL의 경우, 이 함수는 톱 레벨 노드를 탐색한다.

name

노드명.

attributes

사용되지 않는 파라미터.

함수 cvReadByName(은)는,cvGetFileNodeByName (와)과 cvRead 의 단순한 합성이다.

ReadRawData

복수의 수치를 읽어들인다

```
void cvReadRawData( const CvFileStorage* fs, const CvFileNode* src,
                    void* dst, const char* dt );
```

fs

파일 스토리지.

src

수치를 읽어들이는 파일 노드(순서).

dst

기입처의 배열에의 포인터.

dt

배열의 개개의 요소의 사양.사양은cvWriteRawData(와)과 같다.

함수 cvReadRawData(은)는, 스칼라의 순서로 나타내진 파일 노드로부터 요소를 읽어들인다.

StartReadRawData

파일 노드의 순서 리더의 초기화

```
void cvStartReadRawData( const CvFileStorage* fs, const CvFileNode* src,
                         CvSeqReader* reader );
```

fs

파일 스토리지.

src

읽어들이는 파일 노드(순서).

reader

순서 리더에게의 포인터.

함수 cvStartReadRawData(은)는, 파일 노드로부터 데이터를 읽어들이기 위한 순서 리더를 초기화한다. 초기화된 리더는 cvReadRawDataSlice 에 건네줄 수 있다.

ReadRawDataSlice

복수의 수치의 순서를 읽어들인다

```
void cvReadRawDataSlice( const CvFileStorage* fs, CvSeqReader* reader,
                         int count, void* dst, const char* dt );
```

fs

파일 스토리지.

reader

순서 리더.cvStartReadRawData그리고 초기화한다.

count

읽어들이는 요소수.

dst

출력 배열에의 포인터.

dt

각 배열 요소의 사양.사양은cvWriteRawData(와)과 같다.

함수 cvReadRawDataSlice(은)는, 파일로부터, 순서로 표현되는 하나 또는 복수의 요소를 유저 정의 배열에 읽어들인다. 읽어들이는 순서 요소의 총수는 count(와)과 각 배열 요소의 요소수의 적이다. 예를 들면 dt='2if'의 경우, 이 함수는 count*3 의 순서 요소를 읽어들인다.
리더가 cvSetSeqReaderPos 그리고 재배치되면, 파일 노드 순서의 몇개의 부분은 읽어 날아가는지, 중복 해 읽힐 가능성이 있다.

5-4 실행시 형 정보와 범용 함수(RTTI and Gene

ric Functions)

CvTypeInfo

형태 정보

```
typedef int (CV_CDECL *CvIsInstanceFunc)( const void* struct_ptr );
typedef void (CV_CDECL *CvReleaseFunc)( void** struct_dblptr );
typedef void* (CV_CDECL *CvReadFunc)( CvFileStorage* storage, CvFileNode* node );
typedef void (CV_CDECL *CvWriteFunc)( CvFileStorage* storage,
                                      const char* name,
                                      const void* struct_ptr,
                                      CvAttrList attributes );
typedef void* (CV_CDECL *CvCloneFunc)( const void* struct_ptr );

typedef struct CvTypeInfo
{
    int flags; /* 사용되지 않는다 */
    int header_size; /* sizeof(CvTypeInfo) */
    struct CvTypeInfo* prev; /* 리스트내에서 하나전에 등록된 형태 */
    struct CvTypeInfo* next; /* 리스트내에서 하나 후에 등록된 형태 */
    const char* type_name; /* 파일 스토리지에 써진 형태의 이름 */

    /* 메소드 */
    CvIsInstanceFunc is_instance; /* 건네받은 오브젝트가 그 형태에 속하는지를 체크한다 */
    CvReleaseFunc release; /* 오브젝트를 해방한다(메모리 등) */
    CvReadFunc read; /* 파일 스토리지로부터 오브젝트를 읽어들인다 */
    CvWriteFunc write; /* 파일 스토리지에 오브젝트를 쓴다 */
    CvCloneFunc clone; /* 오브젝트의 카피를 생성한다 */
}
CvTypeInfo;
```

구조체 CvTypeInfo(은)는, 표준형 혹은 유저 정의형의 어느 쪽인가에 관한 정보를 포함한다. 그 형태의 인스턴스는, 대응한다 CvTypeInfo 구조체에의 포인터를 가질 가능성이 있다. 어떠한 경우에서도, 함수 cvTypeOf(을)를 사용하는 것으로, 주어진 오브젝트의 형태 정보를 찾아낼 수 있다. 또 다른 방법으로서 오브젝트를 파일 스토리지로부터 읽어들일 때에 사용한다 cvFindType(을)를 이용해 형태의 이름으로부터 찾아낼 수도 있다. 유저는 새로운 형태를 cvRegisterType(을)를 사용하고, 형태 정보 구조체를 형태 리스트의 선두에 등록할 수 있다. 이와 같이, 일반적인 표준의 형태로부터 특별한 형태를 작성해, 기본적인 메소드를 오버라이드(override) 하는 것이 가능하다.

RegisterType

새로운 형태를 등록한다

```
void cvRegisterType( const CvTypeInfo* info );
```
info

형태 정보 구조체.

함수 cvRegisterType(은)는,info 에 기술된 새로운 형태를 등록한다. 이 함수는 구조체의 카피를 작성하므로, 유저는 함수 호출 후에 그 구조체를 삭제해야 한다.

UnregisterType

형태의 등록을 취소한다

```
void cvUnregisterType( const char* type_name );
```
type_name

등록을 취소하는 형태의 이름.

함수 cvUnregisterType(은)는, 지정한 이름의 형태의 등록을 취소한다. 만약 그 이름이 불명하면,cvTypeOf(을)를 사용해 형태의 인스턴스로부터 찾는지, 혹은 cvFirstType(으)로부터 형태 리스트를 순서에 보고 형태 정보를 찾아, cvUnregisterType(info->type_name)(을)를 실행하는 것으로 취소할 수 있다.

FirstType

형태 리스트의 선두를 돌려준다

```
CvTypeInfo* cvFirstType( void );
```
함수 cvFirstType(은)는, 등록되어 있는 형태의 리스트의 선두의 것을 돌려준다.
CvTypeInfo 구조체의 prev(와)과 next 필드를 사용하는 것으로, 리스트를 모두 주사 할 수 있다.

FindType

이름으로부터 형태를 찾아낸다

```
CvTypeInfo* cvFindType( const char* type_name );
```
type_name

형태의 이름.

함수 cvFindType(은)는, 등록된 형태를 이름으로 찾는다. NULL(을)를 돌려주었을 경우, 지정한 이름의 형태는 존재하지 않는다.

TypeOf

오브젝트의 형태를 돌려준다

```
CvTypeInfo* cvTypeOf( const void* struct_ptr );
```
struct_ptr

오브젝트에의 포인터.

함수 cvTypeOf(은)는, 주어진 오브젝트의 형태를 찾아낸다. 등록된 형태의 리스트를 주사 해, 각각의 형태 정보 구조체의 함수/메소드 is_instance(을)를 불러, 그러한 하나가 비 0(을)를 돌려주는지, 모든 리스트를 주사 끝마칠 때까지 반복한다.후자의 경우, 이 함수는 NULL(을)를 돌려준다.

Release

오브젝트를 해방한다

```
void cvRelease( void** struct_ptr );
```

struct_ptr

오브젝트의 포인터의 포인터.

함수 cvRelease(은)는, 주어진 오브젝트의 형태를 찾아내 주어진 포인터의 포인터를 인수에 release(을)를 부른다.

Clone

오브젝트의 카피를 작성한다

```
void* cvClone( const void* struct_ptr );
```

struct_ptr

카피하는 오브젝트.

함수 cvClone(은)는, 준 오브젝트의 형태를 찾아내 건네받은 오브젝트를 인수에 clone(을)를 부른다.

Save

오브젝트를 파일에 보존한다

```
void cvSave( const char* filename, const void* struct_ptr,
             const char* name=NULL,
             const char* comment=NULL,
             CvAttrList attributes=cvAttrList());
```

filename

파일명.

struct_ptr

보존하는 오브젝트.

name

오브젝트명(옵션).NULL의 경우, 이름은filename(으)로부터 생성된다.

comment

파일의 최초로 놓여지는 코멘트(옵션).

attributes

cvWrite에게 건네지는 속성(옵션).

함수 cvSave(은)는, 오브젝트를 파일에 보존한다. 이것은,cvWrite 에의 간단한 인터페이스를 제공한다.

Load

오브젝트를 파일로부터 읽어들인다

```
void* cvLoad( const char* filename, CvMemStorage* memstorage=NULL,
              const char* name=NULL, const char** real_name=NULL );
```

filename

파일명.

memstorage

CvSeq(이)나CvGraph 등의 동적 구조체를 위한 메모리스트레이지.행렬이나 이미지에는 이용되지 않는다.

name

오브젝트명(옵션).NULL의 경우, 파일 스토리지에 있는 최초의 톱 레벨 오브젝트가 읽힌다.

real_name

읽힌 오브젝트의 이름이 대입되는 출력 파라미터(옵션)name=NULL의 경우에 도움이 된다.

함수 cvLoad(은)는, 파일로부터 오브젝트를 읽어들인다. 이것은 cvRead 에의 간단한 인터페이스를 제공한다. 오브젝트가 읽힌 후, 파일 스토리지는 닫혀져 일시적인 버퍼는 모두 소거된다. 따라서, 순서나 윤곽, 그래프등의 동적 구조체를 읽어들이기 위해서는, 이 함수에 올바른 출력처 메모리스트레이지를 줄 필요가 있다.

6.그 외의 함수(Miscellaneous Functions)

CheckArr

입력 배열의 모든 요소에 대해서, 무효인 값이 존재하지 않을까를 체크한다

```
int cvCheckArr( const CvArr* arr, int flags=0,
                double min_val=0, double max_val=0);
#define cvCheckArray cvCheckArr
```

arr

체크 대상의 배열.

flags

처리 플래그.0 혹은 이하의 값의 편성.

CV_CHECK_RANGE - 세트 되고 있는 경우, 배열의 모든 요소에 대해 [minVal,maxVal) 의 범위내일지를 체크한다. 그 이외의 경우, 모든 요소가 NaN 인가 ±(Infinity) (이)가 아닌가만을 체크한다.

CV_CHECK_QUIET - 세트 되고 있는 경우, 요소에 무효인 값이나 범위외의 것이 있어도, 에러를 발생시키지 않는다.

min_val

유효한 치역의 하한치(이 값이상).CV_CHECK_RANGE 하지만 세트 되고 있을 때 마셔 유효.

max_val

유효한 치역의 상한치(이 값미만).CV_CHECK_RANGE 하지만 세트 되고 있을 때 마셔 유효.

함수 cvCheckArr (은)는, 모든 배열 요소가 NaN 혹은 ±(Infinity)(이)가 아닌가를 체크한다. CV_CHECK_RANGE 하지만 세트 되고 있는 경우는, 모든 배열 요소가 minVal 보다 큰가 동일하고, 한편 maxVal 보다 작은가를 체크한다. 체크가 올바르게 끝났을 경우(모든 요소가 유효하고 지정 범위내일 때) 함수는 0 이외를 돌려주어, 그 이외의 경우는 0(을)를 돌려준다. 에러가 있었을 경우,CV_CHECK_QUIET 플래그가 세트 되어 있지 않으면, 함수는 런타임 에러로서 채택한다.

KMeans2

벡터 집합을, 주어진 클러스터수로 분할한다

```
void cvKMeans2( const CvArr* samples, int cluster_count,
                CvArr* labels, CvTermCriteria termcrit );
```

samples

부동 소수점형의 입력 샘플 행렬.1행 맞아 하나의 샘플.

cluster_count

집합을 분할하는 클러스터수.

labels

출력의 정수 벡터.모든 샘플에 대해서, 각각이 어느 클러스터에 속하고 있을지가 보존되고 있다.

termcrit

최대 반복수와(또는), 정도(1루프로의 각 클러스터 중심 위치 이동거리)의 지정.

함수 cvKMeans2 (은)는, 입력 샘플을 각 클러스터로 분류하기 위해서 cluster_count 개의 클러스터의 중심을 요구한다 k-means 법을 실장한다.출력 labels(i) (은)는, 배열 samples 의 i 번째의 행의 샘플이 속하는 클러스터의 인덱스를 나타낸다.

(예)k-변량 Gauss 분포에 따르는 랜덤 샘플의 클러스터링을 실시한다

```
#include "cxcore.h"
#include "highgui.h"

void main( int argc, char** argv )
{
    #define MAX_CLUSTERS 5
    CvScalar color_tab[MAX_CLUSTERS];
    IplImage* img = cvCreateImage( cvSize( 500, 500 ), 8, 3 );
    CvRNG rng = cvRNG(0xffffffff);

    color_tab[0] = CV_RGB(255,0,0);
    color_tab[1] = CV_RGB(0,255,0);
    color_tab[2] = CV_RGB(100,100,255);
    color_tab[3] = CV_RGB(255,0,255);
```

```
    color_tab[4] = CV_RGB(255,255,0);

    cvNamedWindow( "clusters", 1 );

    for(;;)
    {
        int k, cluster_count = cvRandInt(&rng)%MAX_CLUSTERS + 1;
        int i, sample_count = cvRandInt(&rng)%1000 + 1;
        CvMat* points = cvCreateMat( sample_count, 1, CV_32FC2 );
        CvMat* clusters = cvCreateMat( sample_count, 1, CV_32SC1 );

        /* 다변량 Gauss 분포로부터 랜덤 샘플을 생성한다 */
        for( k = 0; k < cluster_count; k++ )
        {
            CvPoint center;
            CvMat point_chunk;
            center.x = cvRandInt(&rng)%img->width;
            center.y = cvRandInt(&rng)%img->height;
            cvGetRows( points, &point_chunk, k*sample_count/cluster_count,
                       k == cluster_count - 1 ? sample_count :
(k+1)*sample_count/cluster_count );
            cvRandArr( &rng, &point_chunk, CV_RAND_NORMAL,
                       cvScalar(center.x,center.y,0,0),
                       cvScalar(img->width/6, img->height/6,0,0) );
        }

        /* 샘플을 상출 한다 */
        for( i = 0; i < sample_count/2; i++ )
        {
            CvPoint2D32f* pt1 = (CvPoint2D32f*)points->data.fl +
cvRandInt(&rng)%sample_count;
            CvPoint2D32f* pt2 = (CvPoint2D32f*)points->data.fl +
cvRandInt(&rng)%sample_count;
            CvPoint2D32f temp;
            CV_SWAP( *pt1, *pt2, temp );
        }

        cvKMeans2( points, cluster_count, clusters,
                   cvTermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 10, 1.0 ));

        cvZero( img );

        for( i = 0; i < sample_count; i++ )
        {
            CvPoint2D32f pt = ((CvPoint2D32f*)points->data.fl)[i];
            int cluster_idx = clusters->data.i[i];
```

```
            cvCircle( img, cvPointFrom32f(pt), 2, color_tab[cluster_idx], CV_FILLED );
        }

        cvReleaseMat( &points );
        cvReleaseMat( &clusters );

        cvShowImage( "clusters", img );

        int key = cvWaitKey(0);
        if( key == 27 ) // 'ESC'
            break;
    }
}
```

SeqPartition

데이터 순서를 동치류(같은 클래스에 속한다고 정의된 데이터군)에 분할한다

```
typedef int (CV_CDECL* CvCmpFunc)(const void* a, const void* b, void* userdata);
int cvSeqPartition( const CvSeq* seq, CvMemStorage* storage, CvSeq** labels,
                    CvCmpFunc is_equal, void* userdata );
```

seq

분할 대상의 순서.

storage

동치류로서 분할된 순서의 보존 영역.NULL의 경우는,seq->storage (을)를 사용한다.

labels

출력 파라미터.입력 순서의 각 요소에 할당해졌다(분할 결과를 나타낸다)0(으)로부터 시작되는 라벨 순서에의 포인터의 포인터.

is_equal

2개의 순서 요소가 같은 클래스인 경우, 관계 함수는 0이외를 돌려준다. 그렇지 않으면0(을)를 돌려준다.분할 알고리즘은, 동치 기준으로서 관계 함수의 추이폐포를 이용한다.

userdata

함수is_equal 의 인수로서 건네주는 데이터에의 포인터.

함수 cvSeqPartition (은)는, 집합을 하나 이상의 동치류에 분할한다 2 차적(계산량이 $O(n^2)$) 알고리즘을 실장한다. 이 함수는 동치류의 개수를 돌려준다.

(예)2차원의 점의 분할

```
#include "cxcore.h"
#include "highgui.h"
#include <stdio.h>

CvSeq* point_seq = 0;
IplImage* canvas = 0;
CvScalar* colors = 0;
int pos = 10;
```

```
int is_equal( const void* _a, const void* _b, void* userdata )
{
    CvPoint a = *(const CvPoint*)_a;
    CvPoint b = *(const CvPoint*)_b;
    double threshold = *(double*)userdata;
    return (double)(a.x - b.x)*(a.x - b.x) + (double)(a.y - b.y)*(a.y - b.y) <=
threshold;
}

void on_track( int pos )
{
    CvSeq* labels = 0;
    double threshold = pos*pos;
    int i, class_count = cvSeqPartition( point_seq, 0, &labels, is_equal, &threshold );
    printf("%4d classes₩n", class_count );
    cvZero( canvas );

    for( i = 0; i < labels->total; i++ )
    {
        CvPoint pt = *(CvPoint*)cvGetSeqElem( point_seq, i );
        CvScalar color = colors[*(int*)cvGetSeqElem( labels, i )];
        cvCircle( canvas, pt, 1, color, -1 );
    }

    cvShowImage( "points", canvas );
}

int main( int argc, char** argv )
{
    CvMemStorage* storage = cvCreateMemStorage(0);
    point_seq = cvCreateSeq( CV_32SC2, sizeof(CvSeq), sizeof(CvPoint), storage );
    CvRNG rng = cvRNG(0xffffffff);

    int width = 500, height = 500;
    int i, count = 1000;
    canvas = cvCreateImage( cvSize(width,height), 8, 3 );

    colors = (CvScalar*)cvAlloc( count*sizeof(colors[0]) );
    for( i = 0; i < count; i++ )
    {
        CvPoint pt;
        int icolor;
        pt.x = cvRandInt( &rng ) % width;
        pt.y = cvRandInt( &rng ) % height;
        cvSeqPush( point_seq, &pt );
```

```
            icolor = cvRandInt( &rng ) | 0x00404040;
            colors[i] = CV_RGB(icolor & 255, (icolor >> 8)&255, (icolor >> 16)&255);
        }

        cvNamedWindow( "points", 1 );
        cvCreateTrackbar( "threshold", "points", &pos, 50, on_track );
        on_track(pos);
        cvWaitKey(0);
        return 0;
}
```

7.에러 처리와 시스템 함수(Error Handling and System Functions)

7-1 에러 핸들링(Error Handling)

OpenCV 에 있어서의 에러 핸들링은,IPL(Image Processing Library)(와)과 닮아 있다. 에러가 발생했을 경우에서도, 함수는 에러 코드를 돌려주지 않는다. 그 대신해, 에러 스테이터스를 cvSetErrStatus(을)를 이용해 세트 해, 표준 또는 유저 정의의 에러 핸들러(메시지 박스의 표시나, 로그에 쓰는, 등. cvRedirectError, cvNulDevReport, cvStdErrReport, cvGuiBoxReport(을)를 참조) (을)를 호출하는 함수 cvError(을)를 부르는 매크로 CV_ERROR (을)를 이용해 에러의 통지를 실시한다. 각 프로그람스렛드에 대해서, 각각 한개씩, 현재의 에러 스테이터스(하나의 정수치)를 가지는 글로벌 변수가 존재한다. 이 에러 스테이터스는, 함수 cvGetErrStatus (을)를 이용해 꺼내는 것이 가능하다.

에러 핸들링에는, 이하의 세 개의 모드가 존재한다(cvSetErrMode (와)과 cvGetErrMode (을)를 참조).

Leaf

프로그램은 에러 핸들러를 호출한 후, 도중 종료한다.*디폴트*.에러 발생 후, 즉시 통지되므로, 디버그시에 유효. 그러나, 제품(완성판의) 시스템에서는, 다른 두 개의 모드의 사용이 제어하기 쉽기 때문에 바람직하다.

Parent

프로그램은 도중 종료하지 않지만, 에러 핸들러가 불려 간다.스택은 해방되지 않는다(C++의 예외 처리를 이용하지 않기 때문에). 유저는CxCore의 함수cvGetErrStatus(을)를 호출해, 에러 코드를 체크해, 대처를 실시하지 않으면 안 된다.

Silent

*Parent*모드와 거의 같지만, 에러 핸들러는 불려 가지 않는다.

실제는, 모드 *Leaf* (와)과 *Parent* 의 동작은 에러 핸들러에 의해서 실장된다. 상기의 설명은 cvNulDevReport, cvStdErrReport 에 관해서는 올바르다. cvGuiBoxReport (은)는 다소 다른 동작을 해, 게다가 커스터마이즈 된 에러 핸들러는, 완전히 다른 동작의 실장이 될 가능성이 있다.

ERROR Handling Macros

에러 표시나 체크등의 기능을 가지는 매크로군

```
/* 함수내의 처리 스테이트먼트를 사이에 두어, 그것들을 프롤로그(자원의 초기화부)와
에필로그
   (확보된 자원의 해방부) 로 분리하는 특별한 매크로 */
#define __BEGIN__       {
#define __END__         goto exit; exit: ; }
/* 「자원 해방 스테이지」로 나아간다 */
#define EXIT            goto exit

/* CV_ERROR() 그리고 사용하는 함수명을 로컬에 정의한다 */
#define CV_FUNCNAME( Name )  ₩
    static char cvFuncName[] = Name

/* 현상의 에러를 보고한다 */
#define CV_ERROR( Code, Msg )                                          ₩
{                                                                      ₩
     cvError( (Code), cvFuncName, Msg, __FILE__, __LINE__ );           ₩
     EXIT;                                                             ₩
}

/* CXCORE 의 함수 호출의 뒤 상태를 체크한다 */
#define CV_CHECK()                                                     ₩
{                                                                      ₩
    if( cvGetErrStatus() < 0 )                                    ₩
        CV_ERROR( CV_StsBackTrace, "Inner function failed." );         ₩
}

/* CXCORE 의 함수 호출과 CV_CHECK()호출의 간략 표현 */
#define CV_CALL( Statement )                                           ₩
{                                                                      ₩
    Statement;                                                         ₩
    CV_CHECK();                                                        ₩
}

/* 디버그 모드와 릴리스 모드 양쪽 모두에 대응한 상태 체크 */
#define CV_ASSERT( Condition )                                             ₩
{                                                                          ₩
    if( !(Condition) )                                                     ₩
        CV_ERROR( CV_StsInternal, "Assertion: " #Condition " failed" ); ₩
}

/* 이러한 매크로는, 각각 대응하는 매크로 CV_... (와)과 닮아 있지만,
```

```
   종료 라벨도 정의용의 cvFuncName 도 필요로 하지 않는다 */
#define OPENCV_ERROR(status,func_name,err_msg) ...
#define OPENCV_ERRCHK(func_name,err_msg) ...
#define OPENCV_ASSERT(condition,func_name,err_msg) ...
#define OPENCV_CALL(statement) ...
```

여기에서는, 상세한 설명 대신에, 대표적인CXCORE의 함수와 그 사용 방법을 나타낸다.

에러 핸들링 매크로의 사용 방법

```
#include "cxcore.h"
#include <stdio.h>

void cvResizeDCT( CvMat* input_array, CvMat* output_array )
{
    CvMat* temp_array = 0; // 머지않아 해방되어야 할 포인터의 선언.

    CV_FUNCNAME( "cvResizeDCT" ); // cvFuncName 의 선언

    __BEGIN__; // 처리의 개시.이 매크로의 직후에 어떠한 선언이 있을 지도 모르지만,
               // 그것들은, 에필로그부에서는 액세스 할 수 없다.

    if( !CV_IS_MAT(input_array) || !CV_IS_MAT(output_array) )
        // 에러 표시를 위해 CV_ERROR() (을)를 이용한다
        CV_ERROR( CV_StsBadArg, "input_array or output_array are not valid matrices" );

    // 후의 버젼으로 삭제되는 몇개의 제한 사항,CV_ASSERT() 그리고 체크될지도 모른다
    CV_ASSERT( input_array->rows == 1 && output_array->rows == 1 );

    // 안전한 함수 호출을 위해서 CV_CALL (을)를 이용한다
    CV_CALL( temp_array = cvCreateMat( input_array->rows, MAX(input_array-
>cols,output_array->cols),
                                       input_array->type ));

    if( output_array->cols > input_array->cols )
        CV_CALL( cvZero( temp_array ));

    temp_array->cols = input_array->cols;
    CV_CALL( cvDCT( input_array, temp_array, CV_DXT_FORWARD ));
    temp_array->cols = output_array->cols;
    CV_CALL( cvDCT( temp_array, output_array, CV_DXT_INVERSE ));
    CV_CALL( cvScale( output_array, output_array, ₩
            1./sqrt((double)input_array->cols*output_array->cols), 0 ));

    __END__; // 처리의 종료.매크로의 뒤에 쓴다.

    // temp_array (을)를 해방한다.에러가 발생하기 전에 temp_array 의 파티션이 끝나지 않은
경우,
    // cvRleaseMat 하지만 처리를 실시한다.이번 같은 경우는, 아무것도 하지 않는다.
```

```
    cvReleaseMat( &temp_array );
}


int main( int argc, char** argv )
{
    CvMat* src = cvCreateMat( 1, 512, CV_32F );
#if 1 /* 에러 없음 */
    CvMat* dst = cvCreateMat( 1, 256, CV_32F );
#else
    CvMat* dst = 0; /* 에러 처리의 메카니즘을 테스트한다 */
#endif
    cvSet( src, cvRealScalar(1.), 0 );
#if 0 /* 에러 핸들러의 기동을 하지 않게 하기 위해서는 0 (으)로부터 1 에 변경한다 */
    cvSetErrMode( CV_ErrModeSilent );
#endif
    cvResizeDCT( src, dst ); // 에러가 발생했을 경우, 메시지 박스가 팝업 하는지,
                             // 메세지가 로그에 써지든가, 혹은 유저 정의의 처리를 한다
    if( cvGetErrStatus() < 0 )
        printf("Some error occured" );
    else
        printf("Everything is OK" );
    return 0;
}
```

GetErrStatus

현재의 에러 스테이터스를 돌려준다

```
int cvGetErrStatus( void );
```

함수 cvGetErrStatus (은)는, 현재의 에러 스테이터스를 돌려준다. 스테이터스치는, 함수 cvSetErrStatus 에 의해서 세트 된다. *Leaf* 모드의 경우는, 에러 발생 후, 즉시 프로그램이 도중 종료하므로, 함수 소환 후도 항상 제어 가능하게 해 두기 위해서는, cvSetErrMode 그리고 에러 모드를 *Parent* 인가 *Silent* 에 세트 해 두어야 하는 것인 것에 주의한다.

SetErrStatus

에러 스테이터스를 세트 한다

```
void cvSetErrStatus( int status );
```

status

에러 스테이터스.

함수 cvSetErrStatus (은)는, 에러 스테이터스를 지정된 값에 세트 한다. 대부분의 경우, 이 함수는 에러 처리를 실시한 후에 에러 스테이터스를 리셋트 한다(CV_StsOk(을)를 세트 한다)

위해(때문에) 이용된다. 그 외의 경우는,cvError 혹은 CV_ERROR (을)를 호출하는 것이 일반적이다.

GetErrMode

현재의 에러 모드를 돌려준다

```
int cvGetErrMode( void );
```

함수 cvGetErrMode (은)는, 현재의 에러 모드를 돌려준다. 모드치는, 직전의 함수 cvSetErrMode 호출에 의해서 세트 된다.

SetErrMode

에러 모드를 세트 한다

```
#define CV_ErrModeLeaf    0
#define CV_ErrModeParent  1
#define CV_ErrModeSilent  2
int cvSetErrMode( int mode );
```

mode

에러 모드.

함수 cvSetErrMode (은)는, 지정된 에러 모드를 세트 한다. 에러 모드의 차이에 관해서는, 이섹션의 처음의 부분을 참조.

Error

에러를 발생시킨다

```
int cvError( int status, const char* func_name,
             const char* err_msg, const char* file_name, int line );
```

status

에러 스테이터스.

func_name

에러가 발생한 함수명.

err_msg

에러에 대한 추가 정보/진단 결과.

file_name

에러가 발생한 파일명.

line

에러가 발생한 행 번호.

함수 cvError (은)는, (cvSetErrStatus(을)를 이용해) 에러 스테이터스를 지정의 값에 세트 한다.한층 더 에러 모드가 *Silent* 이외의 경우는, 에러 핸들러를 호출한다.

ErrorStr

에러 스테이터스의 코드의 텍스트 정보를 돌려준다

```
const char* cvErrorStr( int status );
```

status

에러 스테이터스.

함수 cvErrorStr (은)는, 지정한 에러 스테이터스 코드의 텍스트 기술을 돌려준다. 불명한 스테이터스의 경우는 NULL 포인터를 돌려준다.

RedirectError

새로운 에러 핸들러를 세트 한다

```
typedef int (CV_CDECL *CvErrorCallback)( int status, const char* func_name,
                    const char* err_msg, const char* file_name, int line );

CvErrorCallback cvRedirectError( CvErrorCallback error_handler,
                                 void* userdata=NULL, void** prev_userdata=NULL );
```

error_handler

새로운 에러 핸들러.

userdata

에러 핸들러에의 인수로서 건네받는 임의의 포인터.

prev_userdata

미리 할당할 수 있고 있는 유저 데이터에의 포인터의 포인터.

함수 cvRedirectError (은)는,standard handlers 안의 하나인가, 특정의 인터페이스를 가지는 독자적인 핸들러를 새로운 에러 핸들러에 세트 한다. 에러 핸들러는, 함수 cvError(와)과 같은 파라미터를 가진다. 핸들러가 0 이외의 값을 돌려주었을 경우, 프로그램은 도중 종료해, 그 이외의 경우는 계속 실행한다. 에러 핸들러는, 이러한 동작을 결정하기 위해서 cvGetErrMode 그리고 현재의 에러 모드를 확인하고 있다.

cvNulDevReport cvStdErrReport cvGuiBoxReport

표준의 에러 핸들링을 제공한다

```
int cvNulDevReport( int status, const char* func_name,
                    const char* err_msg, const char* file_name,
                    int line, void* userdata );

int cvStdErrReport( int status, const char* func_name,
                    const char* err_msg, const char* file_name,
                    int line, void* userdata );
```

```
int cvGuiBoxReport( int status, const char* func_name,
                    const char* err_msg, const char* file_name,
                    int line, void* userdata );
```

status

에러 스테이터스.

func_name

에러가 발생한 함수명.

err_msg

에러에 대한 추가 정보/진단 결과.

file_name

에러가 발생한 파일명.

line

에러가 발생한 행 번호.

userdata

유저 데이터에의 포인터.표준 핸들러에서는 무시된다.

함수 cvNullDevReport, cvStdErrReport (와)과 cvGuiBoxReport(은)는 표준의 에러 핸들링을 제공한다. Win32(으)로의 디폴트 에러 핸들러는 cvGuiBoxReport(이어)여, 다른 시스템에서는 cvStdErrReport 된다. cvGuiBoxReport(은)는 에러 메세지를 표시하는 메시지 박스를 팝업시켜, 한층 더 몇개의 옵션을 제공한다. 전술의 sample code(으)로의 메시지 박스를 이용한 에러 처리의 예를 이하에 나타낸다.

에러 메시지 박스

에러 핸들러가cvStdErrReport에 세트 되고 있는 경우, 전술의 메세지가 표준 에러 출력에 출력되어 그 후 프로그램은 에러 모드에 따라서 도중 종료할까 실행을 계속한다.

표준 에러 출력에의 에러 메세지 출력(*Leaf* 모드)

```
OpenCV ERROR: Bad argument (input_array or output_array are not valid matrices)
        in function cvResizeDCT, D:₩User₩VP₩Projects₩avl_proba₩a.cpp(75)
Terminating the application...
```

7-2 시스템 함수(System Functions)

Alloc

메모리밧파의 영역을 확보한다

```
void* cvAlloc( size_t size );
```

size

밧파사이즈(아르바이트 단위).

함수 cvAlloc 하 size 아르바이트의 영역을 확보해, 그 영역에의 포인터를 돌려준다. 에러가 생겼을 경우는, 에러를 통지해 NULL 포인터를 돌려준다. 디폴트로 cvAlloc (은)는,malloc(을)를 부른다 icvAlloc 의 호출을 실시하지만, 함수 cvSetMemoryManager (을)를 이용하고, 유저 정의의 메모리 파티션/영역 해방을 위한 함수를 정의하는 것도 가능하다.

Free

메모리밧파의 영역을 해방한다

```
void cvFree( void** ptr );
```

ptr

해방하는 영역에의 포인터의 포인터.

함수 cvFree (은)는,cvAlloc 에 의해서 확보된 메모리밧파의 영역 해방을 실시한다. 함수로부터 나올 때에 버퍼에의 포인터를 클리어 하기 위한(해), 포인터의 포인터를 이용하고 있다. *ptr 하지만 이미 NULL 의 경우, 이 함수는 아무것도 하지 않는다.

GetTickCount

tick 수를 돌려준다

```
int64 cvGetTickCount( void );
```

함수 cvGetTickCount (은)는, 플랫폼 의존의 개시시점으로부터의 tick 수(스타트업으로부터의 CPU tick 수,1970 해부터의 밀리 세컨드등)을 돌려준다. 이 함수는, 어느 함수나 유저 코드의 실행 시간을 정확하게 계측하는데 편리하다. tick 수로부터 시간 단위로 변환하기 위해서는,cvGetTickFrequency(을)를 이용한다.

GetTickFrequency

1 마이크로 세컨드 근처의 tick 수를 돌려준다

```
double cvGetTickFrequency( void );
```

함수 cvGetTickFrequency (은)는,1 마이크로 세컨드 근처의 tick 수를 돌려준다.
즉,cvGetTickCount() (을)를 cvGetTickFrequency() 그리고 나눈 값이, 플랫폼 의존의 개시시각으로부터의 마이크로 세컨드 단위의 시각이 된다.

RegisterModule

다른 모듈을 등록한다

```
typedef struct CvPluginFuncInfo
{
    void** func_addr;
    void* default_func_addr;
    const char* func_names;
    int search_modules;
    int loaded_from;
}
CvPluginFuncInfo;

typedef struct CvModuleInfo
{
    struct CvModuleInfo* next;
    const char* name;
    const char* version;
    CvPluginFuncInfo* func_tab;
}
CvModuleInfo;

int cvRegisterModule( const CvModuleInfo* module_info );
```

module_info

모듈에 관한 정보.

함수 cvRegisterModule (은)는, 모듈이 등록되어 있는 리스트에 새로운 모듈을 추가한다. 모듈이 등록되면, 그 모듈의 정보는, 함수 cvGetModuleInfo (을)를 이용해 꺼낼 수 있게 된다. 등록된 모듈도,CXCORE 그리고 서포트되고 있는 최적화 플러그 인(IPP,MKL,...)(을)를 활용한다. CXCORE 및 CV (computer vision),CVAUX (auxilary computer vision) (이)나 HIGHGUI (visualization & image/video acquisition) 등은 모듈의 일례이다. 통상은, 등록이 끝나고 나서 공유 라이브러리(shared library)(이)가 로드 된다. 어떻게 등록될까의 자세한 것은,cxcore/src/cxswitcher.cpp (와)과 cv/src/cvswitcher.cpp (을)를 참조.
또,IPP(와)과 MKL 하지만 어떻게 모듈과 링크 되는지에 대해서는,cxcore/src/cxswitcher.cpp, cxcore/src/_cxipp.h (을)를 참조.

GetModuleInfo

등록된 모듈과 플러그 인의 정보를 꺼낸다

```
void cvGetModuleInfo( const char* module_name,
                      const char** version,
                      const char** loaded_addon_plugins );
```

module_name

대상의 모듈명,NULL의 경우는 모든 모듈.

version

출력 파라미터.모듈에 대한 정보(버젼을 포함한다).

loaded_addon_plugins

CXCORE하지만 로드 가능한 최적화 플러그 인의 이름과 버젼의 리스트.

함수 cvGetModuleInfo (은)는, 등록 모듈의 하나, 또는 모든 정보를 돌려준다. 돌려주어진 정보는 라이브러리 내부에 보존된다.그 때문에 유저가 영역 해방이나 돌려주어진 문자열의 변경을 실시할 필요는 없다.

UseOptimized

최적화 모드/비최적화 모드를 바꾼다

```
int cvUseOptimized( int on_off );
```

on_off

0이외 때 최적화,0때 비최적화.

함수 cvUseOptimized (은)는,cxcore(이)나,OpenCV 외의 라이브러리등을, 순수한 C 언어만으로 실장한 모드와 가능한 부분은 IPP (와)과 MKL(을)를 이용해 최적화한 모드를 바꾼다. cvUseOptimized(0) 하지만 불렸을 때는, 최적화 라이브러리는 로드 되지 않는다. 이 함수는 디버그시,IPP&MKL 의 업그레이드시, 처리 속도를 동작중과 비교하고 싶을 때 등에 유효하다. 반환값은, 로드 된 최적화 함수의 수. 디폴트로 최적화 플러그 인이 로드 되므로, 프로그램의 최초로 cvUseOptimized(1)(을)를 부를 필요는 없다 (실제, 스타트 업시에 처리 시간이 증가할 뿐) 일로 주의.

SetMemoryManager

커스텀 혹은 디폴트의 메모리 메니지먼트 함수를 지정한다

```
typedef void* (CV_CDECL *CvAllocFunc)(size_t size, void* userdata);
typedef int (CV_CDECL *CvFreeFunc)(void* pptr, void* userdata);

void cvSetMemoryManager( CvAllocFunc alloc_func=NULL,
                         CvFreeFunc free_func=NULL,
                         void* userdata=NULL );
```

alloc_func

파티션 함수(인터페이스는, 콘텍스트를 결정하기 위해서 사용되는 일이 있다userdata이외는, malloc(와)과 같다).

free_func

영역 해방 함수(인터페이스는,free(와)과 같다).

userdata

커스텀 함수에 인수로서 건네주는 유저 데이터.

함수 cvSetMemoryManager (은)는,cvAlloc, cvFree (이)나 상위 레벨의 함수 (예를 들면,cvCreateImage)(으)로부터 불리는 유저 정의의 메모리 메니지먼트 함수(malloc(와)과 free 에 취해 대신하는 것)의 설정을 실시한다. cvAlloc 에 의해서 데이터 영역을 확보할 경우에, 이 함수가 불리는 것에 주의. 재귀 호출의 엔들레스 루프를 피하기 위해, 유저 정의의 파티션/해방 함수에서는 cvAlloc(와)과 cvFree 의 호출을 실시해선 안 된다.

alloc_func (와)과 free_func 에의 포인터가 NULL 의 경우는, 디폴트의 메모리 메니지먼트 함수가 이용된다.

SetIPLAllocators

이미지 영역의 확보와 해방을 위한 IPL 함수로 전환한다

```
typedef IplImage* (CV_STDCALL* Cv_iplCreateImageHeader)
                            (int,int,int,char*,char*,int,int,int,int,int,
                            IplROI*,IplImage*,void*,IplTileInfo*);
typedef void (CV_STDCALL* Cv_iplAllocateImageData)(IplImage*,int,int);
typedef void (CV_STDCALL* Cv_iplDeallocate)(IplImage*,int);
typedef IplROI* (CV_STDCALL* Cv_iplCreateROI)(int,int,int,int,int);
typedef IplImage* (CV_STDCALL* Cv_iplCloneImage)(const IplImage*);

void cvSetIPLAllocators( Cv_iplCreateImageHeader create_header,
                         Cv_iplAllocateImageData allocate_data,
                         Cv_iplDeallocate deallocate,
                         Cv_iplCreateROI create_roi,
                         Cv_iplCloneImage clone_image );

#define CV_TURN_ON_IPL_COMPATIBILITY()                                  ₩
    cvSetIPLAllocators( iplCreateImageHeader, iplAllocateImage,         ₩
                        iplDeallocate, iplCreateROI, iplCloneImage )
```

create_header

iplCreateImageHeader에의 포인터.

allocate_data

iplAllocateImage에의 포인터.

deallocate

iplDeallocate에의 포인터.

create_roi

iplCreateROI에의 포인터.

clone_image

iplCloneImage에의 포인터.

함수 cvSetIPLAllocators(은)는,CXCORE 하지만, 이미지 영역의 확보와 해방을 실시하기 위해서 IPL 의 함수를 사용하도록(듯이) 변경한다. 편리성을 위해서, 랩핑 매크로

CV_TURN_ON_IPL_COMPATIBILITY 하지만 존재한다. 이 함수는,IPL (와)과 CXCORE/OpenCV(을)를 동시에 사용해 있거나,iplCreateImageHeader 등을 불러 있거나 하는 경우에는 유효하다. IPL 하지만 단지 데이터 처리만으로, 모든 파티션이나 해방은 CXCORE 그리고 가고 있는 경우나, 반대로 모든 파티션이나 해방을 IPL 그리고, 그리고 데이터 처리는 OpenCV 의 함수로 가고 있는 경우는, 이 함수는 불필요하다.

GetNumThreads

현재 사용되고 있는 스렛드수를 돌려준다

```
int cvGetNumThreads( void);
```

함수 cvGetNumThreads(은)는, (OpenMP(을)를 이용해) 병렬화 되었다 OpenCV 함수에 의해서 사용되는 현재의 스렛드수를 돌려준다.

SetNumThreads

스렛드수를 세트 한다

```
void cvSetNumThreads( int threads=0 );
```

threads

스렛드수.

함수 cvSetNumThreads (은)는, 병렬화 되었다 OpenCV 함수에 의해서 사용되는 스렛드수를 세트 한다. 인수가 0 인가 부의 경우, 또 프로그램 개시시는, 스렛드수에 OpenMP 런타임의 함수 omp_get_num_procs() 의 반환값인 시스템의 프로세서수가 세트 된다.

GetThreadNum

현재의 스렛드의 인덱스를 돌려준다

```
int cvGetThreadNum( void );
```

함수 cvGetThreadNum (은)는, 이 함수를 부른 스렛드의 인덱스 (0(으)로부터 cvGetNumThreads()-1 의 범위의 값)을 돌려준다. 이 함수는,OpenMP 런타임의 함수 omp_get_thread_num()의 래퍼이다. 꺼내진 인덱스는, 병렬화 된 각 코드내의 로카르스렛드의 데이터에 액세스 하기 위해서 이용된다.

Sample Code

IplImage

OpenCV그럼,IPL(Intel Image Processing Library)(으)로 사용되고 있던 구조체 IplImage 포맷의 일부를 서포트하고 있다. OpenCV(이)가 많은 함수가, 구조체 IplImage (을)를 포함한다 CvArr (을)를, 그 인수에 취한다. 구조체 IplImage의 멤버,imageData 에, 실제의 픽셀의 값이 격납되고 있어 그 이미지의 폭과 높이는 각각,width,height그리고 나타난다. 또, 멤버 변수 widthStep (은)는, 이미지의 수평 방향1라인분을 아르바이트 단위로 나타내는 값이며, 이것은,imageData 의 데이터에 직접 액세스 할 때에 이용된다. widthStep (은)는 메모리의 얼라이먼트(alignment)에 의해,width (와)과 같은가, 그것보다 큰 값이 된다.

픽셀데이타의 직접 액세스 IplImage

8 비트 3 채널 칼라 이미지를 읽어들여, 픽셀 데이타를 변경한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
  int x, y;
  uchar p[3];
  IplImage *img;

  if (argc != 2 || (img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0)
    return -1;

  // (1)픽셀 데이타(R,G,B)(을)를 차례차례 취득해, 변경한다
  for (y = 0; y < img->height; y++) {
    for (x = 0; x < img->width; x++) {
      p[0] = img->imageData[img->widthStep * y + x * 3];         // B
      p[1] = img->imageData[img->widthStep * y + x * 3 + 1];     // G
      p[2] = img->imageData[img->widthStep * y + x * 3 + 2];     // R
      img->imageData[img->widthStep * y + x * 3] = cvRound (p[0] * 0.7 + 10);
      img->imageData[img->widthStep * y + x * 3 + 1] = cvRound (p[1] * 1.0);
      img->imageData[img->widthStep * y + x * 3 + 2] = cvRound (p[2] * 0.0);
    }
  }

  cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
```

```
  cvShowImage ("Image", img);
  cvWaitKey (0);

  cvDestroyWindow ("Image");
  cvReleaseImage (&img);

  return 0;
}
```

// (1)픽셀 데이타(R,G,B)(을)를 차례차례 취득해, 변경한다
이미지의 각 픽셀 데이타(칼라 이미지이라면, 각 픽셀의 RGB 값인 것이 많다)를 취득하는(즉, 색을 꺼낸다), 혹은 값을 격납한다고 하는 처리는, 이미지 처리의 기본 조작이다. OpenCV 그럼, 구조체 IplImage 의 멤버이다 imageData 배열에 픽셀 데이타가 격납되고 있다. 이 배열의 값을 꺼내고, 적당한 연산에 의해 값을 변경하고 있다. 이번은,R(빨강) 채널의 값을 0 에,B(파랑) 채널치를 약 0.7 배로 하고 있다.
또, 픽셀 데이타를 취득하기 위한 코드 쓰는 법에는, 이하와 같은 것이 있다.

(1) 픽셀 데이타에의 포인터를 최초로 계산한다

```
 char *pt1, *pt2;
  for(pt1 = img->imageData; pt1 < img->imageData + img->widthStep*img->height; pt1 +=
img->widthStep) {
    for(pt2 = pt1; pt2 < pt1 + img->width*3; pt2 += 3) {
      p[0] = pt2[0];
      p[1] = pt2[1];
      p[2] = pt2[2];
    }
  }
```

(2)CV_IMAGE_ELEM 매크로를 이용한다

```
for(y=0; yheight; y++) {
    for(x=0; xwidth; x++) {
      p[0] = CV_IMAGE_ELEM(img, uchar, y, x*3+0);
      p[1] = CV_IMAGE_ELEM(img, uchar, y, x*3+1);
      p[2] = CV_IMAGE_ELEM(img, uchar, y, x*3+2);
    }
}
```

(3)OpenCV 의 배열 액세스 함수를 이용한다

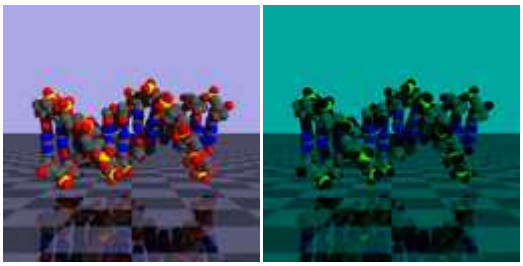
```
 CvScalar s;
  for(y = 0; y < img->height; y++) {
    for(x = 0; x < img->width; x++) {
      s = cvGet2D(img, x, y);
    }
  }
```

마지막, 함수cvGet*D(을)를 이용한 액세스는, 함수 호출의 오바헷드도 있어 매우 저속이다. 어느 특정 픽셀의 값 에

대해서만 액세스 하고 싶은 경우를 제외하고, 함수cvGet*D(을)를 이용한 방법은 추천 되지 않는다.

실행 결과예

[좌]입력 이미지 [우]처리 결과



부분 이미지의 치환 cvSetImageROI, cvResetImageROI

ROI(을)를 이용하고, 부분 이미지끼리를 바꿔 넣는다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <time.h>

/* 분할수의 정의 */
#define DIVX  (4)
#define DIVY  (4)
#define DIVXY (DIVX*DIVY)

int
main (int argc, char **argv)
{
  int w, h, i, j;
  IplImage *src_img, *dst_img, *tmp_img[2];
  CvRect roi[DIVXY];
  CvRNG rng = cvRNG (time (NULL));

  // (1)이미지를 읽어들인다
  if (argc != 2 || (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_ANYDEPTH |
CV_LOAD_IMAGE_ANYCOLOR)) == 0)
    return -1;

  w = src_img->width - src_img->width % DIVX + DIVX;
  h = src_img->height - src_img->height % DIVY + DIVY;
  dst_img = cvCreateImage (cvSize (w, h), src_img->depth, src_img->nChannels);
  tmp_img[0] = cvCreateImage (cvSize (w / DIVX, h / DIVY), src_img->depth, src_img-
```

```
>nChannels);
  tmp_img[1] = cvCreateImage (cvSize (w / DIVX, h / DIVY), src_img->depth, src_img-
>nChannels);
  cvResize (src_img, dst_img);

  // (2)이미지를 분할하기 위한 구형을 설정
  for (i = 0; i < DIVX; i++) {
    for (j = 0; j < DIVY; j++) {
      roi[DIVX * j + i].x = w / DIVX * i;
      roi[DIVX * j + i].y = h / DIVY * j;
      roi[DIVX * j + i].width = w / DIVX;
      roi[DIVX * j + i].height = h / DIVY;
    }
  }

  // (3)ROI(을)를 이용해 부분 이미지를 바꿔 넣는다
  for (i = 0; i < DIVXY * 2; i++) {
    int p1 = cvRandInt (&rng) % DIVXY;
    int p2 = cvRandInt (&rng) % DIVXY;
    cvSetImageROI (dst_img, roi[p1]);
    cvCopy (dst_img, tmp_img[0]);
    cvSetImageROI (dst_img, roi[p2]);
    cvCopy (dst_img, tmp_img[1]);
    cvCopy (tmp_img[0], dst_img);
    cvSetImageROI (dst_img, roi[p1]);
    cvCopy (tmp_img[1], dst_img);
  }
  cvResetImageROI (dst_img);
  cvResize (dst_img, src_img);

  // (4)이미지의 표시
  cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Image", src_img);
  cvWaitKey (0);

  cvDestroyWindow ("Image");
  cvReleaseImage (&src_img);
  cvReleaseImage (&dst_img);
  cvReleaseImage (&tmp_img[0]);
  cvReleaseImage (&tmp_img[1]);

  return 0;
}
```

// (1)이미지를 읽어들인다
커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고

읽어들인다.2 번째의 인수에 CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR(을)를 지정하는 것으로, 원이미지의 데프스, 채널을 변 갱 하지 않고 읽어들인다. 읽힌 이미지는, 지정된 만큼 할수로 등분 가능한 것 같게, 함수 cvResize()에 의해 사이즈를 조정한다. 이 때에, 보간 방법을 지정하지 않는 경우는, 디폴트치의 CV_INTER_LINEAR 하지만 이용된다.

// (2)이미지를 분할하기 위한 구형을 설정
이미지를 분할하기 위해서, 구조체 CvRect(을)를 설정한다.

// (3)ROI(을)를 이용해 부분 이미지를 바꿔 넣는다
설정된 구형 CvRect (을)를 이용해 이미지에 ROI (을)를 설정해, 그 ROI 전체를 카피하고, 랜덤에 바꿔 넣는다. 이러한 처리는 ROI (을)를 이용하지 않아도,MASK 의 이용이나 1 픽셀씩 카피한다, 실제로 이미지를 분할하는 등의 수법에 의해 실현될 수 있지만, ROI (을)를 이용하는 것으로 비교적 단순한 처리로 할 수 있다. 또, 함수 cvRandShuffle() (을)를 이용하면, 반복 계산 없이 이미지를 상홀 하는 것이 가능하다. 교체 후는,ROI(을)를 해방해, 사이즈를 원래의 크기(입력 이미지 사이즈)에 되돌린다. 사이즈를 되돌릴 때도 같이 함수 cvResize()(을)를 이용해, 여기서 보간 방법을 지정하지 않는 경우는, 디폴트치의 CV_INTER_LINEAR 하지만 이용된다.

// (4)이미지의 표시
부분 이미지가 상홀 된 이미지 포함한 윈도우를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예

[좌]입력 이미지 [우]처리 결과

이미지의 연결 cvSetImageROI, cvResetImageROI

복수의 이미지를 횡방향에 연결해 1 매의 이미지로 한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <stdlib.h>

/* 프로토 타입 선언 */
IplImage *combine_image (int num, IplImage ** tmp);

/* 메인 함수 */
int
```

```
main (int argc, char **argv)
{
  int i, img_num;
  IplImage **img;
  IplImage *combined_img;

  // (1)커멘드 인수로 지정된 이미지를 모두 읽어들인다
  if (argc < 2) {
    return 1;
  }
  else {
    img_num = argc - 1;
    img = (IplImage **) cvAlloc (sizeof (IplImage *) * img_num);
    for (i = 0; i < img_num; i++) {
      img[i] = cvLoadImage (argv[i + 1], CV_LOAD_IMAGE_COLOR);
      if (img[i] == 0)
        return -1;
    }
  }

  // (2)이미지를 연결한다
  combined_img = combine_image (img_num, img);

  cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Image", combined_img);
  cvWaitKey (0);

  cvDestroyWindow ("Image");
  cvReleaseImage (&combined_img);
  for (i = 0; i < img_num; i++) {
    cvReleaseImage (&img[i]);
  }
  cvFree (&img);

  return 0;
}

/* 이미지를 연결하는 함수 */
IplImage *
combine_image (int num, IplImage ** tmp)
{
  int i;
  int width = 0, height = 0;
  IplImage *cimg;
  CvRect roi = cvRect (0, 0, 0, 0);
```

```
  // (3)주어진 각 이미지로부터, 연결 후의 폭과 높이를 요구한다
  for (i = 0; i < num; i++) {
    width += tmp[i]->width;
    height = height < tmp[i]->height ? tmp[i]->height : height;
  }
  cimg = cvCreateImage (cvSize (width, height), IPL_DEPTH_8U, 3);
  cvZero (cimg);

  // (4)ROI(을)를 이용해 각 이미지를 카피한다
  for (i = 0; i < num; i++) {
    roi.width = tmp[i]->width;
    roi.height = tmp[i]->height;
    cvSetImageROI (cimg, roi);
    cvCopy (tmp[i], cimg);
    roi.x += roi.width;
  }
  cvResetImageROI (cimg);

  return cimg;
}
```

// (1)커멘드 인수로 지정된 이미지를 모두 읽어들인다
커멘드 인수로 지정된 이미지를 모두, 칼라 이미지로서 읽어들인다.그 안에 1 개에서도 읽어들일 수 없는 이미지가 있는 경우에는 종료한다.

// (2)이미지를 연결한다
어머나 장독 정의한 함수 combined_img()(을)를 호출하고, 이미지를 횡방향에 연결한다. 이 예에서는 단순하게 이미지의 외관을 가지런히 해 횡방향에 연결을 실시하는 것만으로 있지만, 이미지를 되풀이해 연결하거나 한층 더 이른바 「구형 패킹 문제」를의 해를 요구하는 것으로 쓸데 없는 영역이 작아지도록(듯이) 연결할 수도 있다. 또, 이 샘플에서는 연결하는 함수 combined_image()에 이미지의 포인터의 배열을 건네주고 있지만, 가변 인수를 이용(stdarg.h) 하고,combine_image()(을)를 다음과 같이 정의하는 방법도 있다.

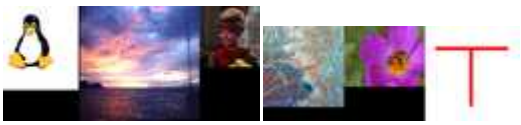
```
IplImage*
combine_image(int num, ...)
{
  va_list list;
  int i;
  int width = 0, height = 0;
  IplImage *tmp[num];
  IplImage *cimg;
  CvRect roi = cvRect(0, 0, 0, 0);

  va_start(list, num);
  for(i=0; iwidth;
    height = height < tmp[i]->height ? tmp[i]->height : height;
```

```
  }
  va_end(list);
  cimg = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);
  cvZero(cimg);

  for(i=0; iwidth;
    roi.height = tmp[i]->height;
    cvSetImageROI(cimg, roi);
    cvCopy(tmp[i], cimg);
    roi.x += roi.width;
  }
  cvResetImageROI(cimg);

  return cimg;
}
```

// (3)주어진 각 이미지로부터, 연결 후의 폭과 높이를 요구한다
주어진 각 이미지의 폭의 합계치를 연결 후의 이미지폭(횡방향에 연결하므로)으로 해, 또, 주어진 각 이미지의 최대의 높이를 연결 후의 이미지 높이로서 이미지를 작성한다. 작성된 이미지를, 함수 cvZero()에 의해 초기화한다.

// (4)ROI(을)를 이용해 각 이미지를 카피한다
각 이미지의 사이즈(폭과 높이)를 만나게 해 ROI(을)를 설정해, 각 이미지를 cvCopy()에 의해 카피한다. 마스크는 지정하지 않기 때문에, 디폴트치 NULL 하지만 이용된다. 또, 마지막에 함수 cvResetImageROI()(을)를 이용하고, 설정했다 ROI(을)를 리셋트 하는 일을을 잊지 않게 주의.

실행 결과예



이미지의 카피 cvCopy

카피처의 이미지를 2 치화해 마스크 영역을 작성한 다음, 카피원이미지를 카피처 이미지에 카피한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
  IplImage *src_img = 0, *dst_img = 0;
```

```
  IplImage *cpy_img, *dst_img_gray, *msk_img;

  // (1)이미지를 읽어들인다
  if (argc != 3 ||
      (src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR)) == 0 ||
      (dst_img = cvLoadImage (argv[2], CV_LOAD_IMAGE_COLOR)) == 0)
    return -1;

  cpy_img = cvCloneImage (dst_img);
  if (src_img->width != dst_img->width)
    return -1;
  if (src_img->height != dst_img->height)
    return -1;


  // (2)dst_img 의 이미지의 Red 의 프레인만을 2 값화해 마스크를 작성한다
  dst_img_gray = cvCreateImage (cvGetSize (dst_img), IPL_DEPTH_8U, 1);
  cvSplit (dst_img, NULL, NULL, dst_img_gray, NULL);

  msk_img = cvCloneImage (dst_img_gray);
  cvSmooth (dst_img_gray, dst_img_gray, CV_GAUSSIAN, 5);
  cvThreshold (dst_img_gray, msk_img, 0, 255, CV_THRESH_BINARY_INV | CV_THRESH_OTSU);

  // (3)src_img(을)를 dst_img 에 카피한다.마스킹되고 있는 부분은 원래의 값이 보관
유지된다.
  cvCopy (src_img, cpy_img, msk_img);


  // (4)결과를 표시한다
  cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("mask", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("copy", CV_WINDOW_AUTOSIZE);
  cvShowImage ("src", src_img);
  cvShowImage ("dst", dst_img);
  cvShowImage ("mask", msk_img);
  cvShowImage ("copy", cpy_img);
  cvWaitKey (0);

  cvDestroyWindow ("src");
  cvDestroyWindow ("dst");
  cvDestroyWindow ("mask");
  cvDestroyWindow ("copy");
  cvReleaseImage (&src_img);
  cvReleaseImage (&dst_img);
  cvReleaseImage (&msk_img);
```

```
  cvReleaseImage (&cpy_img);
  cvReleaseImage (&dst_img_gray);

  return 1;

}
```

// (1)이미지를 읽어들인다
커멘드 인수로 지정된 파일명의 이미지(카피원이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.2 번째의 인수에 CV_LOAD_IMAGE_COLOR(을)를 지정하는 것으로,
데프스 8 비트,3 채널의 칼라 이미지로서 읽어들인다.
이와 같이 2 번째의 인수로 지정된 파일명의 이미지(카피처 이미지)도 읽어들인다. 이미지의 사이즈가 같은가를 확인해 둔다.

// (2)dst_img 의 이미지의 Red 의 프레인만을 2 치화해 마스크를 작성한다
카피처 이미지다 dst_img(을)를 2 값화해, 그 결과를 마스킹 이미지로 한다. 실행 예의 같게 인간의 피부색을 분리할 때 , 칼라플레인에 주목와 예쁘게 분리할 수가 있다. 이 예에서는 단순하게,Red 의 프레인만을 추출해 그 프레인에 대해서, 2 치화를 행하고 있다. 2 치화 방법의 상세한 것에 대하여는이미지의 2 치화 cvThreshold, cvAdaptiveThreshold(을)를 참조의 일.

// (3)src_img(을)를 dst_img 에 카피한다.마스킹되고 있는 부분은 원래의 값이 보관 유지된다.
함수 cvCopy()(을)를 이용하고, 마스크 영역을 고려한 이미지의 카피를 행한다. 구체적으로는, 이하와 같이 마스크 이미지의 화소치가 0(이)가 아닌 영역만 카피원이미지가 카피된다.

mask(I)!=0 의 경우,dst(I)=src(I)
// (4)결과를 표시한다
형상을 변환한 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예

카피원이미지	카피처 이미지	마스크 이미지	카피 결과
			

이미지 형상의 변형 cvReshape

데이터를 카피하지 않고 ,3 채널 이미지를 각 채널 데이터를 전개했다 1 채널 이미지에 변형한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>
```

```
int
main (int argc, char **argv)
{
  IplImage *src_img = 0;
  IplImage img_hdr, *dst_img;
  CvMat mat_hdr;

  // (1)이미지를 읽어들인다
  if (argc >= 2)
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
  if (src_img == 0)
    return -1;

  // (2)이미지 형상의 변경을 실시한다
  cvReshape (src_img, &mat_hdr, 1, 0);
  dst_img = cvGetImage (&mat_hdr, &img_hdr);

  printf ("src_img size:[ %d, %d ]  data-pointer:[%p]¥n", src_img->width, src_img-
>height, src_img->imageData);
  printf ("mat_hdr size:[ %d, %d ]  data-pointer:[%p]¥n", mat_hdr.width, mat_hdr.height,
mat_hdr.data.ptr);

  // (3)결과를 표시한다
  cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
  cvShowImage ("src", src_img);
  cvShowImage ("dst", dst_img);
  cvWaitKey (0);

  cvDestroyWindow ("src");
  cvDestroyWindow ("dst");
  cvReleaseImage (&src_img);

  return 1;
}
```

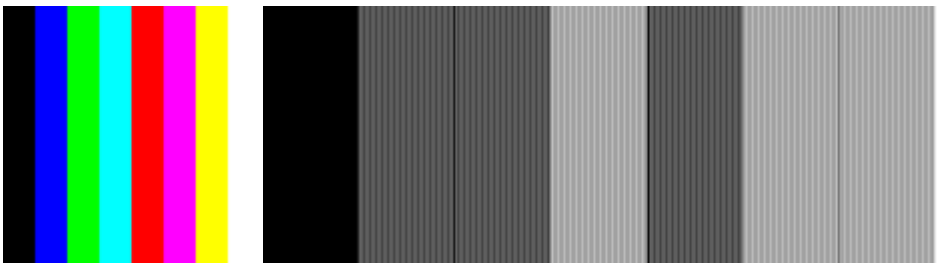
// (1)이미지를 읽어들인다
커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.2 번째의 인수에 CV_LOAD_IMAGE_COLOR(을)를 지정하는 것으로, 데프스 8 비트,3 채널의 칼라 이미지로서 읽어들인다.

// (2)이미지 형상의 변경을 실시한다
함수 cvReshape()의 3 번째의 인수로, 채널수를 1 으로 설정.4 번 뿌리의 인수에 0(을)를 지정하는 것으로, 배열의 행수(이미지의 높이에 상당)를 바꾸지 않고 배열의 사이즈를 바꾼다. 이것은, 배열의 렬수(이미지의 폭)를 3 배가 되는 일을 의미하고 있다.
cvReshape()하 CvMat 형태의 헤더를 돌려주기 위해, 함수 cvGetImage()그리고,IplImage 구조체의

헤더 (으)로 변환하고 있다.
이 후의 printf 문장으로 출력되는 정보를 확인하면, 데이터에의 포인터는 같은 일을 알 수 있다.

// (3)결과를 표시한다
형상을 변환한 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예



■ 치환

OpenCV의CXCORE레퍼런스 메뉴얼의Transforms and Permutations에 분류되는 함수 가운데, Permutations에는, 데이터의 치환에 의한 배열(이미지)의 매핑이나 반전등을 행하는 기본적인 함수가 준비되어 있다.
샘플

타일링 cvRepeat

이미지의 타일링을 행한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <stdio.h>

int
main (int argc, char **argv)
{
  IplImage *src_img = 0;
  IplImage *dst_img;

  // (1)이미지를 읽어들인다
  if (argc >= 2)
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
  if (src_img == 0)
    return -1;

  // (2)출력 이미지 영역을 작성해, 입력 이미지를 타일링 한다
  dst_img = cvCreateImage (cvSize (384, 512), IPL_DEPTH_8U, 3);
```

```
  cvRepeat (src_img, dst_img);

  // (3)결과를 표시한다
  cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
  cvShowImage ("src", src_img);
  cvShowImage ("dst", dst_img);
  cvWaitKey (0);

  cvDestroyWindow ("src");
  cvDestroyWindow ("dst");
  cvReleaseImage (&src_img);
  cvReleaseImage (&dst_img);

  return 1;
}
```

// (1)이미지를 읽어들인다
커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다.2 번째의 인수에 CV_LOAD_IMAGE_COLOR(을)를 지정하는 것으로, 데프스 8 비트,3 채널의 칼라 이미지로서 읽어들인다.

// (2)출력 이미지 영역을 작성해, 입력 이미지를 타일링 한다
함수 cvCreateImage()에 의해서, 폭 384, 높이 512 의 이미지 영역을 생성한다. 생성한 이미지에 대해서, 함수 cvRepeat()(을)를 이용해 입력 이미지를 타일링 한다.

// (3)결과를 표시한다
처리된 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예

입력 이미지(256x192)하지만 출력 이미지(384x512)보다 작은 경우

입력 이미지(512x384)하지만 출력 이미지(384x512)보다 폭이 큰 경우

이미지의 반전 cvFlip(1)

이미지의 수직·수평· 양축반전을 실시한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
  IplImage *src_img = 0;
  IplImage *dst_img_1, *dst_img_2, *dst_img_3;

  // (1)이미지를 읽어들여, 같은 사이즈의 출력 이미지를 준비한다
  if (argc >= 2)
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
  if (src_img == 0)
    return -1;
  dst_img_1 = cvCloneImage (src_img);
  dst_img_2 = cvCloneImage (src_img);
  dst_img_3 = cvCloneImage (src_img);

  // (2)이미지의 수평축반전 · 수직축반전 · 양축반전을 실시한다
  cvFlip (src_img, dst_img_1, 0);
  cvFlip (src_img, dst_img_2, 1);
  cvFlip (src_img, dst_img_3, -1);

  // (3)결과를 표시한다
  cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst_1", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst_2", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst_3", CV_WINDOW_AUTOSIZE);
  cvShowImage ("src", src_img);
  cvShowImage ("dst_1", dst_img_1);
```

```
  cvShowImage ("dst_2", dst_img_2);
  cvShowImage ("dst_3", dst_img_3);
  cvWaitKey (0);

  cvDestroyWindow ("src");
  cvDestroyWindow ("dst_1");
  cvDestroyWindow ("dst_2");
  cvDestroyWindow ("dst_3");
  cvReleaseImage (&src_img);
  cvReleaseImage (&dst_img_1);
  cvReleaseImage (&dst_img_2);
  cvReleaseImage (&dst_img_3);

  return 1;
}
```

// (1)이미지를 읽어들여, 같은 사이즈의 출력 이미지를 준비한다
커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다. 또, 반전 후의 이미지를 보존하기 위해서 3 개의 이미지 영역을 생성한다.

// (2)이미지의 수평축반전·수직축반전· 양축반전을 실시한다
함수 cvFlip()에 의해서, 이미지의 반전을 행한다. 3 번째의 인수에 0(을)를 지정해 수평축반전을,1(>0)(을)를 지정해 수직축반전을, -1(<0)(을)를 지정해 양축반전을 실시한다.

// (3)결과를 표시한다
처리된 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예

입력 이미지	수평축반전	수직축반전	양축반전
			

ROI(을)를 이용한 이미지의 부분 반전 cvFlip(2)

ROI(을)를 이용해 이미지의 일부분에 대해서 수직·수평· 양축반전을 실시한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
```

```
int
main (int argc, char **argv)
{
  int w, h;
  IplImage *src_img = 0;
  IplImage *dst_img;

  // (1)이미지를 읽어들여, 입력 이미지를 카피한 출력 이미지를 준비한다
  if (argc >= 2)
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
  if (src_img == 0)
    return -1;
  dst_img = cvCloneImage (src_img);

  // (2)출력 이미지에 ROI(을)를 세트 해 각부에 대해서 수평축반전・수직축반전・ 양축반전을
인프레이스모드로 실시한다
  w = dst_img->width / 2;
  h = dst_img->height / 2;

  cvSetImageROI (dst_img, cvRect (w, 0, w, h));
  cvFlip (dst_img, NULL, 0);

  cvSetImageROI (dst_img, cvRect (0, h, w, h));
  cvFlip (dst_img, NULL, 1);

  cvSetImageROI (dst_img, cvRect (w, h, w, h));
  cvFlip (dst_img, NULL, -1);

  cvResetImageROI (dst_img);

  // (3)결과를 표시한다
  cvLine (src_img, cvPoint (0, h), cvPoint (src_img->width - 1, h), CV_RGB (255, 0, 255),
2, 8, 0);
  cvLine (src_img, cvPoint (w, 0), cvPoint (w, src_img->height - 1), CV_RGB (255, 0,
255), 2, 8, 0);

  cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
  cvShowImage ("src", src_img);
  cvShowImage ("dst", dst_img);

  cvWaitKey (0);
  return 1;

}
```

// (1)(1)이미지를 읽어들여, 입력 이미지를 카피한 출력 이미지를 준비한다
커멘드 인수로 지정된 파일명의 이미지(입력 이미지)을 오픈해, 함수 cvLoadImage()그리고 읽어들인다. 또, 처리용의 이미지로서 입력 이미지의 카피를 생성한다.

// (2)출력 이미지에 ROI(을)를 세트 해 각부에 대해서 수평축반전·수직축반전· 양축반전을 인프레이스모드로 실시한다
이미지를 4 분할해, 최초로 우상의 영역을 ROI(으)로서 세트 한다. 계속 되어 함수 cvFlip()의 2 번째의 인수에 NULL(을)를 지정해,3 번째의 인수에 0(을)를 지정하는 것으로, 세트 했다 ROI 영역에 대해서 인프레이스모드로 부분적인 수평축반전을 실시한다.
계속 되어 같은 방법으로에, 좌하 영역, 우하 영역에 대해서 각각 수직축반전(제 3 인수에 1(을)를 지정), 양축반전(제 3 인수에-1(을)를 지정)을 실시한다.
마지막에 cvResetImageROI()그리고,ROI(을)를 해방한다.

// (3)결과를 표시한다
입력 이미지의 4 분할의 영역을 표시하기 위해서, 입력 이미지 src_img 위에 cvLine()그리고 수평·수직선분을 그리기 한다. 처리된 이미지를 실제로 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예

입력 이미지(분할 영역을 표시)과 부분 반전을 행한 출력 이미지

역행열의 계산 cvInvert

역행열의 계산을 실시한다

샘플 코드

표시의 변환

```
#include <stdio.h>
#include <highgui.h>
#include <time.h>

int
main (int argc, char **argv)
{
  int i, j;
  int nrow = 3;
  int ncol = 3;
  CvMat *src, *dst, *mul;
  double det;
```

```
CvRNG rng = cvRNG (time (NULL));      /* 난수의 초기화 */

// (1) 행렬의 메모리 확보
src = cvCreateMat (nrow, ncol, CV_32FC1);
dst = cvCreateMat (ncol, nrow, CV_32FC1);
mul = cvCreateMat (nrow, nrow, CV_32FC1);

// (2) 행렬 src 에 난수를 대입
printf ("src₩n");
cvmSet (src, 0, 0, 1);
for (i = 0; i < src->rows; i++) {
  for (j = 0; j < src->cols; j++) {
    cvmSet (src, i, j, cvRandReal (&rng));
    printf ("% lf₩t", cvmGet (src, i, j));
  }
  printf ("₩n");
}

// (3) 행렬 src 의 역행열을 요구하고, 행렬 dst 에 대입
det = cvInvert (src, dst, CV_SVD);

// (4) 행렬 src 의 행렬식을 표시
printf ("det(src)=%lf₩n", det);

// (5) 행렬 dst 의 표시
printf ("dst₩n");
for (i = 0; i < dst->rows; i++) {
  for (j = 0; j < dst->cols; j++) {
    printf ("% lf₩t", cvmGet (dst, i, j));
  }
  printf ("₩n");
}

// (6) 행렬 src(와)과 dst 의 적을 계산해 확인
cvMatMul (src, dst, mul);
printf ("mul₩n");
for (i = 0; i < mul->rows; i++) {
  for (j = 0; j < mul->cols; j++) {
    printf ("% lf₩t", cvmGet (mul, i, j));
  }
  printf ("₩n");
}

// (7) 행렬의 메모리를 개방
cvReleaseMat (&src);
cvReleaseMat (&dst);
```

```
  cvReleaseMat (&mul);

  return 0;
}
```

// (1) 행렬의 메모리 확보
함수 cvCreateMat()에 의해 32 비트 부동 소수점수(실수)형 1 채널의 nrow 행 ncol 열의 행렬 src,ncol 행 nrow 열의 행렬 dst,nrow 행 nrow 열의 행렬 mut(을)를 각각 준비한다.

// (2) 행렬 src 에 난수를 대입
함수 cvRandReal()에 의해 행렬 src 에 0 이상 1 미만의 부동 소수점수(실수)의 적당한 난수를 대입해, 행렬의 요소를 표시한다.

// (3) 행렬 src 의 역행열을 요구하고, 행렬 dst 에 대입
함수 cvInvert()에 의해 행렬 src 의 역행열을 요구해 행렬 dst 에 대입한다. 여기에서는 제 3 인수에 CV_SVD(을)를 지정해, 특이치 분해에 의해 역행열 요구하고 있다.

// (4) 행렬 src 의 행렬식을 표시
함수 cvInvert()하지만 돌려주는 행렬 src 의 행렬식을 표시한다. 이 값이 너무 작은 경우, 역행열이 구해지지 않는 경우가 있으므로 주의한다.

// (5) 행렬 dst 의 표시
구할 수 있던 행렬 dst(을)를 표시한다.

// (6) 행렬 src(와)과 dst 의 적을 계산해 확인
함수 cvMatMul()에 의해 행렬 src(와)과 dst 의 적을 요구해 단위행렬이 되는 것을 확인한다.

실행 결과예

■ 사상
샘플

색공간(color space)의 사상 cvTransform

행렬과 벡터에 의해 색공간(color space)를 한 번에 사상한다

샘플 코드

표시의 변환

```
#include <cv.h>
```

```
#include <highgui.h>
#include <stdio.h>
#include <ctype.h>

int
main (int argc, char **argv)
{
  IplImage *src_img = 0, *dst_img = 0;
  CvMat *mat;
  CvMat *sft;

  // (1)이미지 읽기
  src_img = cvLoadImage ("colorbar_src.png", CV_LOAD_IMAGE_ANYDEPTH |
CV_LOAD_IMAGE_ANYCOLOR);
  if (src_img == 0)
    return -1;
  dst_img = cvCloneImage (src_img);
  if (dst_img == 0)
    return -1;

  // (2)변환 행렬과 시프트 벡터의 메모리 확보
  mat = cvCreateMat (src_img->nChannels, src_img->nChannels, CV_32FC1);
  sft = cvCreateMat (src_img->nChannels, 1, CV_32FC1);

  // (3)변환 행렬의 요소를 세트
  // B 채널을 파기
  cvmSet (mat, 0, 0, 0.0);      // src_img 의 B 채널→dst_img 의 B 채널
  cvmSet (mat, 1, 0, 0.0);      // src_img 의 B 채널→dst_img 의 G 채널
  cvmSet (mat, 2, 0, 0.0);      // src_img 의 B 채널→dst_img 의 R 채널

  // G 채널을 R 채널에 변환
  cvmSet (mat, 0, 1, 0.0);      // src_img 의 G 채널→dst_img 의 B 채널
  cvmSet (mat, 1, 1, 0.0);      // src_img 의 G 채널→dst_img 의 G 채널
  cvmSet (mat, 2, 1, 1.0);      // src_img 의 G 채널→dst_img 의 R 채널

  // R 채널을 B 채널에 변환
  cvmSet (mat, 0, 2, 1.0);      // src_img 의 R 채널→dst_img 의 B 채널
  cvmSet (mat, 1, 2, 0.0);      // src_img 의 R 채널→dst_img 의 G 채널
  cvmSet (mat, 2, 2, 0.0);      // src_img 의 R 채널→dst_img 의 R 채널

  // (4)시프트 벡터의 요소를 세트
  cvmSet (sft, 0, 0, -128.0);   // B 채널을-128
  cvmSet (sft, 1, 0, 0.0);      // G 채널은 그대로
  cvmSet (sft, 2, 0, 0.0);      // R 채널은 그대로

  // (5)mat, sft 에 따라서 src_img(을)를 dst_img 에 변환
```

```
  cvTransform (src_img, dst_img, mat, sft);

  // (6)결과를 파일 출력
  cvSaveImage ("colorbar_dst.png", dst_img);

  // (7)결과를 화면에 표시
  cvNamedWindow ("src", CV_WINDOW_AUTOSIZE);
  cvNamedWindow ("dst", CV_WINDOW_AUTOSIZE);
  cvShowImage ("src", src_img);
  cvShowImage ("dst", dst_img);

  // (8)키 입력 대기
  cvWaitKey (0);

  // (9)후처리
  cvDestroyWindow ("src");
  cvDestroyWindow ("dst");
  cvReleaseImage (&src_img);

  return 0;
}
```

// (1) 이미지 읽기
함수 cvLoadImage()에 의해 원이미지 colorbar_src.png(을)를 읽어들여,src_img 에 격납한다. 또
함수 cvCloneImage()에 의해 src_img 의 카피를 작성해,dst_img 에 격납한다.

// (2) 변환 행렬과 시프트 벡터의 메모리 확보
함수 cvCreateMat()에 의해, src_img 의 채널수와 같은 행 수열수를 가진다 3x3 부동
소수점형 32 비트 1 채널의 변환 행렬 mat(을)를 작성한다. 이와 같이 3x1 요소를 가지는 시프트
벡터 sft(을)를 작성한다.

// (3) 변환 행렬의 요소를 세트 // (4) 시프트 벡터의 요소를 세트
함수 cvmSet()에 의해,mat(와)과 sft 의 요소를 이하와 같이 세트 한다.

// (5) mat, sft 에 따라서 src_img(을)를 dst_img 에 변환
함수 cvTransform()에 의해, 색공간(color space)의 사상을 이하의 식에 따라서 요구해
결과를 dst_img 에 격납한다.

// (6) 결과를 파일 출력
함수 cvSaveImage()에 의해,dst_img(을)를 colorbar_dst.png(으)로서 보존한다.

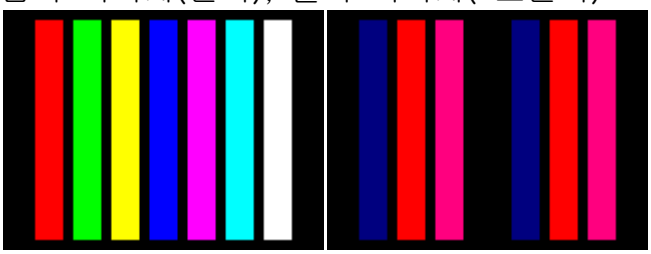
// (7) 결과를 화면에 표시

// (8) 키 입력 대기

// (9) 후처리

실행 결과예

입력 이미지(왼쪽), 출력 이미지(오른쪽)



이산 푸리에 변환 cvDFT

이산 푸리에 변환을 이용하고, 진폭 이미지를 생성한다.이것은, OpenCV 부속의 샘플 코드(와 거의 동일)이다.

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

/* 프로토 타입 선언 */
void cvShiftDFT (CvArr * src_arr, CvArr * dst_arr);

int
main (int argc, char **argv)
{
  IplImage *src_img;
  IplImage *realInput;
  IplImage *imaginaryInput;
  IplImage *complexInput;
  IplImage *image_Re;
  IplImage *image_Im;
  int dft_M, dft_N;
  CvMat *dft_A, tmp;
```

```
  double m, M;

  src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_GRAYSCALE);
  if (!src_img)
    return -1;

  realInput = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_64F, 1);
  imaginaryInput = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_64F, 1);
  complexInput = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_64F, 2);

  // (1)입력 이미지를 실수 배열에 카피해, 허수 배열과 머지 해 복소수 평면을 구성
  cvScale (src_img, realInput, 1.0, 0.0);
  cvZero (imaginaryInput);
  cvMerge (realInput, imaginaryInput, NULL, NULL, complexInput);

  // (2)DFT 용무의 최적 사이즈를 계산해, 그 사이즈로 행렬을 확보한다
  dft_M = cvGetOptimalDFTSize (src_img->height - 1);
  dft_N = cvGetOptimalDFTSize (src_img->width - 1);
  dft_A = cvCreateMat (dft_M, dft_N, CV_64FC2);
  image_Re = cvCreateImage (cvSize (dft_N, dft_M), IPL_DEPTH_64F, 1);
  image_Im = cvCreateImage (cvSize (dft_N, dft_M), IPL_DEPTH_64F, 1);

  // (3)복소수 평면을 dft_A 에 카피해, 나머지의 행렬 우측 부분을 0 그리고 묻는다
  cvGetSubRect (dft_A, &tmp, cvRect (0, 0, src_img->width, src_img->height));
  cvCopy (complexInput, &tmp, NULL);
  if (dft_A->cols > src_img->width) {
    cvGetSubRect (dft_A, &tmp, cvRect (src_img->width, 0, dft_A->cols - src_img->width,
src_img->height));
    cvZero (&tmp);
  }

  // (4)이산 푸리에 변환을 실시해, 그 결과를 실수 부분과 허수 부분에 분해
  cvDFT (dft_A, dft_A, CV_DXT_FORWARD, complexInput->height);
  cvSplit (dft_A, image_Re, image_Im, 0, 0);

  // (5)스펙트럼의 진폭을 계산 Mag = sqrt(Re^2 + Im^2)
  cvPow (image_Re, image_Re, 2.0);
  cvPow (image_Im, image_Im, 2.0);
  cvAdd (image_Re, image_Im, image_Re, NULL);
  cvPow (image_Re, image_Re, 0.5);

  // (6)진폭의 대수를 취한다 log(1 + Mag)
  cvAddS (image_Re, cvScalarAll (1.0), image_Re, NULL);
  cvLog (image_Re, image_Re);

  // (7)원점(직류 성분)이 이미지의 중심에 오도록(듯이), 이미지의 상한을 바꿔 넣는다
```

```
  cvShiftDFT (image_Re, image_Re);

  // (8)진폭 이미지의 픽셀치가 0.0-1.0 에 분포하도록(듯이) 슬캘링
  cvMinMaxLoc (image_Re, &m, &M, NULL, NULL, NULL);
  cvScale (image_Re, image_Re, 1.0 / (M - m), 1.0 * (-m) / (M - m));

  cvNamedWindow ("Image", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Image", src_img);
  cvNamedWindow ("Magnitude", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Magnitude", image_Re);
  cvWaitKey (0);

  cvDestroyWindow ("Image");
  cvDestroyWindow ("Magnitude");
  cvReleaseImage (&src_img);
  cvReleaseImage (&realInput);
  cvReleaseImage (&imaginaryInput);
  cvReleaseImage (&complexInput);
  cvReleaseImage (&image_Re);
  cvReleaseImage (&image_Im);
  cvReleaseMat (&dft_A);

  return 0;
}

/* 원점(직류 성분)이 이미지의 중심에 오도록(듯이), 이미지의 상한을 바꿔 넣는 함수.
   src_arr, dst_arr (은)는 같은 사이즈, 타입의 배열 */
void
cvShiftDFT (CvArr * src_arr, CvArr * dst_arr)
{
  CvMat *tmp = 0;
  CvMat q1stub, q2stub;
  CvMat q3stub, q4stub;
  CvMat d1stub, d2stub;
  CvMat d3stub, d4stub;
  CvMat *q1, *q2, *q3, *q4;
  CvMat *d1, *d2, *d3, *d4;

  CvSize size = cvGetSize (src_arr);
  CvSize dst_size = cvGetSize (dst_arr);
  int cx, cy;

  if (dst_size.width != size.width || dst_size.height != size.height) {
    cvError (CV_StsUnmatchedSizes, "cvShiftDFT", "Source and Destination arrays must have
equal sizes", __FILE__,
             __LINE__);
```

```
  }
  // (9)인프레이스모드용의 텐포라리밧파
  if (src_arr == dst_arr) {
    tmp = cvCreateMat (size.height / 2, size.width / 2, cvGetElemType (src_arr));
  }
  cx = size.width / 2;          /* 이미지 중심 */
  cy = size.height / 2;

  // (10)1~4 상한을 나타내는 배열과 그 카피처
  q1 = cvGetSubRect (src_arr, &q1stub, cvRect (0, 0, cx, cy));
  q2 = cvGetSubRect (src_arr, &q2stub, cvRect (cx, 0, cx, cy));
  q3 = cvGetSubRect (src_arr, &q3stub, cvRect (cx, cy, cx, cy));
  q4 = cvGetSubRect (src_arr, &q4stub, cvRect (0, cy, cx, cy));
  d1 = cvGetSubRect (src_arr, &d1stub, cvRect (0, 0, cx, cy));
  d2 = cvGetSubRect (src_arr, &d2stub, cvRect (cx, 0, cx, cy));
  d3 = cvGetSubRect (src_arr, &d3stub, cvRect (cx, cy, cx, cy));
  d4 = cvGetSubRect (src_arr, &d4stub, cvRect (0, cy, cx, cy));

  // (11)실제의 상한의 교체
  if (src_arr != dst_arr) {
    if (!CV_ARE_TYPES_EQ (q1, d1)) {
      cvError (CV_StsUnmatchedFormats, "cvShiftDFT", "Source and Destination arrays must
have the same format",
               __FILE__, __LINE__);
    }
    cvCopy (q3, d1, 0);
    cvCopy (q4, d2, 0);
    cvCopy (q1, d3, 0);
    cvCopy (q2, d4, 0);
  }
  else {                        /* 인프레이스모드 */
    cvCopy (q3, tmp, 0);
    cvCopy (q1, q3, 0);
    cvCopy (tmp, q1, 0);
    cvCopy (q4, tmp, 0);
    cvCopy (q2, q4, 0);
    cvCopy (tmp, q2, 0);
  }
}
```

// (1)입력 이미지를 실수 배열에 카피해, 허수 배열과 머지 해 복소수 평면을 구성
입력 이미지(그레이 스케일)을 실수 배열에 카피해,0 그리고 클리어 된 허수 배열과 머지 하고, 복소수 평면 데이터를 격납하는 이미지 구조체에 카피한다. 입력 이미지로부터 실수 배열에의 카피는, 입력과 출력으로 배열이 다르기 위해, 함수 cvCopy()(은)는 아니고 함수 cvScale()(을)를 이용한다.

// (2)DFT 용무의 최적 사이즈를 계산해, 그 사이즈로 행렬을 확보한다
함수 cvGetOptimalDFTSize()에 의해, 고속으로 계산 가능하고 한편, 입력 이미지가 다스려지는 최소 사이즈를 요구해 그 사이즈로 행렬을 확보한다. 또, 이산 푸리에 변환 후의 데이터를 분리하기 위한 이미지 구조체,image_Re,image_Im 도 동시에 확보한다.

// (3)복소수 평면을 dft_A 에 카피해, 나머지의 행렬 우측 부분을 0 그리고 묻는다
앞이 요구한 행렬 dft_A 에, 복소수 평면의 데이터를 카피해, 여백 부분인 행렬의 우측을 0 그리고 묻는다. 후에 호출하는 함수 cvDFT()그럼,nonzero_rows 파라미터를 이용하므로, 아래 쪽의 여백을 0 그리고 묻을 필요는 없다.

// (4)이산 푸리에 변환을 실시해, 그 결과를 실수 부분과 허수 부분에 분해
실제로,dft_A 에 대해서 이산 푸리에 변환을 실시해, 그 결과를 dft_A 에 돌려준다. 그리고, 그 결과를 실수 부분과 허수 부분 으로 분해한다.

// (5)스펙트럼의 진폭을 계산 Mag = sqrt(Re^2 + Im^2)
실수 부분과 허수 부분의 제곱화의 히라카타를 잡아, 진폭(Magnitude)(을)를 계산한다.

// (6)진폭의 대수를 취한다 log(1 + Mag)
일반적으로, 이미지에서는 진폭을 가지는 주파수 성분과 비교해서 직류 성분이 매우 많아, 실제의 진폭을 그대로 표시하려고 하면, 직류 성분 이외의 데이터가 0(이)가 되어 버린다. 거기서, (1 이상의 값이 되도록(듯이) 해) 진폭의 대수를 계산하는 것으로, 성분끼리의 격차를 완화한다.

// (7)원점(직류 성분)이 이미지의 중심에 오도록(듯이), 이미지의 상한을 바꿔 넣는다
이대로, 이미지를 표시하면 원점이 되는 직류 성분이 화면단이 되므로, 그것이 중심에 오도록(듯이), 이미지를 상한 단위로 나란해져 바꾼다.

// (8)진폭 이미지의 픽셀치가 0.0~1.0 에 분포하도록(듯이) 슬캘링
작성된 진폭 이미지가,0.0~1.0 의 범위에 분포하도록(듯이) 슬캘링을 실시해, 진폭 이미지와 그 원이미지를 표시한다. 무엇인가 키가 밀릴 때까지 기다린다. 덧붙여서, 이 진폭 이미지는,1 픽셀이 IPL_DEPTH_64F 그리고 정의되는 데이터형이므로, 이대로는 보존할 수 없다. 이미지로서 보존하고 싶은 경우는,

```
cvConvertScale(image_Re, tmp_image, 255);
```

등으로 해서,0.0-1.0의 범위에 정규화되었다IPL_DEPTH_64F의 데이터를,IPL_DEPTH_8U의 데이터로 변환한 후에,cvSaveImage()그리고 보존한다.

// (9)인프레이스모드용의 텐포라리밧파
인프레이스모드, 즉, 입력 배열 포인터와 출력 배열 포인터가 같은 것을 별로 있는 경우, 단순하게 이미지의 카피를 실시할 수 없다. 그 때문에, 데이터 카피용으로 텐포라리밧파를 확보한다.

// (10)1-4 상한을 나타내는 배열과 그 카피처
제 1-제 4 상한까지를 나타내는 행렬과 그 카피처인 동사이즈의 행렬을, 부분 행렬에의 포인터로서 지정한다.

// (11)실제의 상한의 교체
실제로, 이미지를 정렬한다.제 1 상한과 제 3 상한의 교체 , 제 2 상한과 제 4 상한의 교체를 실시한다. 또, 인프레이스모드의 경우는, 먼저 확보한 텐포라리밧파를 이용한다.

실행 결과예

윤곽 좌표의 취득 cvInitTreeNodeIterator, cvNextTreeNode

윤곽의 검출을 실시해, 나무 구조를 가지는 윤곽 데이터로부터 좌표를 꺼낸다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char *argv[])
{
  int i;
  IplImage *src_img = 0;
  IplImage *src_img_gray = 0;
  IplImage *tmp_img;
  CvMemStorage *storage = cvCreateMemStorage (0);
  CvSeq *contours = 0;
  CvPoint *point, *tmp;
  CvSeq *contour;
  CvTreeNodeIterator it;
  CvFileStorage *fs;

  if (argc >= 2)
    src_img = cvLoadImage (argv[1], CV_LOAD_IMAGE_COLOR);
  if (src_img == 0)
    return -1;

  src_img_gray = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);
  cvCvtColor (src_img, src_img_gray, CV_BGR2GRAY);
  tmp_img = cvCreateImage (cvGetSize (src_img), IPL_DEPTH_8U, 1);

  // (1)이미지의 2 치화와 윤곽의 검출
  cvThreshold (src_img_gray, tmp_img, 120, 255, CV_THRESH_BINARY);
  cvFindContours (tmp_img, storage, &contours, sizeof (CvContour), CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE);
```

```
  /* 윤곽 순서로부터 좌표를 취득 */
  fs = cvOpenFileStorage ("contours.yaml", 0, CV_STORAGE_WRITE);
  // (2)트리노드이테레이타의 초기화
  cvInitTreeNodeIterator (&it, contours, 1);
  // (3)각 노드(윤곽)를 주사
  while ((contour = (CvSeq *) cvNextTreeNode (&it)) != NULL) {
    cvStartWriteStruct (fs, "contour", CV_NODE_SEQ);
    // (4)윤곽을 구성하는 정점 좌표를 취득
    tmp = CV_GET_SEQ_ELEM (CvPoint, contour, -1);
    for (i = 0; i < contour->total; i++) {
      point = CV_GET_SEQ_ELEM (CvPoint, contour, i);
      cvLine (src_img, *tmp, *point, CV_RGB (0, 0, 255), 2);
      cvStartWriteStruct (fs, NULL, CV_NODE_MAP | CV_NODE_FLOW);
      cvWriteInt (fs, "x", point->x);
      cvWriteInt (fs, "y", point->y);
      cvEndWriteStruct (fs);
      tmp = point;
    }
    cvEndWriteStruct (fs);
  }
  cvReleaseFileStorage (&fs);

  cvNamedWindow ("Contours", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Contours", src_img);
  cvWaitKey (0);

  cvDestroyWindow ("Contours");
  cvReleaseImage (&src_img);
  cvReleaseImage (&src_img_gray);
  cvReleaseImage (&tmp_img);
  cvReleaseMemStorage (&storage);

  return 0;
}
```

// (1)이미지의 2 치화와 윤곽의 검출
여기까지의 처리는, 윤곽의 검출과 그리기(으)로의 샘플과 동일해서, 그 쪽을 참조.

// (2)트리노드이테레이타의 초기화
검출된 윤곽의 순서로부터 좌표를 취득한다. 우선, 함수 cvInitTreeNodeIterator()에 의해, 트리노드이테레이타를 초기화한다 (반드시 하나의 윤곽 밖에 가지지 않는 듯한 단순한 경우는, 트리노드이테레이타를 이용하지 않고와도 상관없다). 여기서, 마지막 인수는, 검출하는 윤곽의 최대 레벨을 나타내고 있다(여기에서는 레벨 2(을)를 지정해 있다). 즉 여기에서는, 함수 cvFindContours()그리고 윤곽을 검출할 때에, 제 5 인수에 CV_RETR_TREE 하지만 지정되는 경우를 상정하고 있다. 함수 cvFindContours()의 인수가,CV_RETR_EXTERNAL 의 경우(가장 외측의 윤곽 밖에 검출되지 않는다)와 CV_RETR_LIST 의 경우(리스트 스트럭션)는, 검출된 윤곽

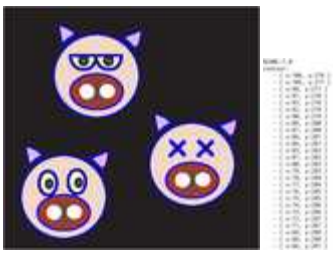
순서가 계층 구조를 가지지 않기 때문에, 레벨 지정은 의미를 만들어내지 않는다.
CV_RETR_CCOMP 의 경우는, 윤곽 순서가 2 계층의 구조 밖에 갖지 않기 때문에,2 레벨까지로 모든 윤곽을 취득할 수 있다. 또, 함수 cvFindContours()의 제 6 인수로,CV_CHAIN_CODE 이외가 지정되어 있지 않으면 안 된다. 왜냐하면,CV_CHAIN_CODE(을)를 지정했을 경우는, 윤곽 데이터가 정점의 순서는 아니고 체인 코드로 표현되어 버리기 때문이다(물론, 거기로부터 좌표를 재계산하는 것은 가능하다).

// (3)각 노드(윤곽)를 주사
다음에, 함수 cvNextTreeNode()에 의해, 차례차례 트리 노드(각 바퀴유곽 순서)에의 포인터를 얻는다. 돌려주는 것 트리 노드가 존재하지 않게 된다, 즉 마지막 노드까지 가까스로 도착하면, 이 함수는 NULL(을)를 돌려준다.

// (4)윤곽을 구성하는 정점 좌표를 취득
얻을 수 있던 노드의 요소인 좌표 데이터(CvPoint)에의 포인터를, 매크로 CV_GET_SEQ_ELEM 에 의해 취득한다. 그리고 취득된 좌표끼리를 직선으로 묶는다(cvLine) 일로 윤곽을 그리기 해, 게다가 각 좌표를 파일에 YAML 형식에서 보존한다.
또, 이 방법으로 보존된 파일은, 나무 구조를 보관 유지하지 않고, 모든 가지(윤곽)를 병렬에 늘어놓을 수 있다.

실행 결과예

[좌]각 좌표를 직선(파랑)으로 묶어 표시된 윤곽.[우]보존된 윤곽 데이터(contours.yml).



도형의 그리기 cvLine, cvRectangle, cvCircle, cvEllipse

선분, 구형, 엔, 타원(선형을 포함한다)을 랜덤에 그리기 한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <time.h>

int
main (int argc, char **argv)
{
  int i;
  IplImage *img = 0;
  CvRNG rng = cvRNG (time (NULL));
  CvPoint pt1, pt2;
```

```
CvScalar rcolor;
int irandom;

// (1)이미지 영역을 확보해 초기화한다
img = cvCreateImage (cvSize (800, 800), IPL_DEPTH_8U, 3);
cvZero (img);

// (2)랜덤인 선분을 그리기 한다
for (i = 0; i < 5; i++) {
  pt1.x = cvRandInt (&rng) % (img->width);
  pt1.y = cvRandInt (&rng) % (img->height);
  pt2.x = cvRandInt (&rng) % (img->width);
  pt2.y = cvRandInt (&rng) % (img->height);
  irandom = cvRandInt (&rng);
  rcolor = CV_RGB (irandom & 255, (irandom >> 8) & 255, (irandom >> 16) & 255);
  cvLine (img, pt1, pt2, rcolor, cvRandInt (&rng) % 4, CV_AA, 0);
}

// (3)랜덤인 구형을 그리기 한다
for (i = 0; i < 5; i++) {
  pt1.x = cvRandInt (&rng) % (img->width);
  pt1.y = cvRandInt (&rng) % (img->height);
  pt2.x = cvRandInt (&rng) % (img->width);
  pt2.y = cvRandInt (&rng) % (img->height);
  irandom = cvRandInt (&rng);
  rcolor = CV_RGB (irandom & 255, (irandom >> 8) & 255, (irandom >> 16) & 255);
  cvRectangle (img, pt1, pt2, rcolor, cvRandInt (&rng) % 5 - 1, CV_AA, 0);
}

// (4)랜덤인 엔을 그리기 한다
for (i = 0; i < 5; i++) {
  pt1.x = cvRandInt (&rng) % (img->width);
  pt1.y = cvRandInt (&rng) % (img->height);
  irandom = cvRandInt (&rng);
  rcolor = CV_RGB (irandom & 255, (irandom >> 8) & 255, (irandom >> 16) & 255);
  cvCircle (img, pt1, cvRandInt (&rng) % (img->width / 4) + img->width / 4, rcolor,
            cvRandInt (&rng) % 5 - 1, CV_AA, 0);
}

// (5)랜덤인 타원을 그리기 한다
for (i = 0; i < 5; i++) {
  pt1.x = cvRandInt (&rng) % (img->width);
  pt1.y = cvRandInt (&rng) % (img->height);
  irandom = cvRandInt (&rng);
  rcolor = CV_RGB (irandom & 255, (irandom >> 8) & 255, (irandom >> 16) & 255);
  cvEllipse (img, pt1, cvSize (img->width / 2, img->height / 2), irandom % 180,
```

```
irandom % 90, irandom % 90 + 90,
                rcolor, cvRandInt (&rng) % 5 - 1, 0);
  }

  // (6)이미지의 표시, 키가 밀렸을 때에 종료
  cvNamedWindow ("Drawing", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Drawing", img);
  cvWaitKey (0);

  cvDestroyWindow ("Drawing");
  cvReleaseImage (&img);

  return 0;
}
```

// (1)이미지 영역을 확보해 초기화한다
이 샘플에서는, 외부 이미지를 읽어들이지 않고 , 프로그램 내부에서 이미지를 준비한다. 또, 작성된 칼라 이미지를, 흑(0)(으)로 전부 칠해 초기화한다.

// (2)랜덤인 선분을 그리기 한다
이미지내에 들어가도록(듯이), 랜덤인 사이즈와 색을 가지는 선분을 5 본작성해, 그리기 한다.

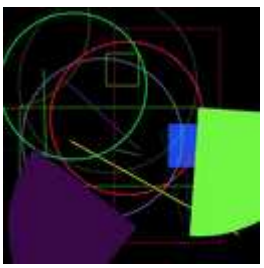
// (3)랜덤인 구형을 그리기 한다
이미지내에 들어가도록(듯이), 랜덤인 사이즈와 색을 가지는 구형을 5 개작성해, 그리기 한다.

// (4)랜덤인 엔을 그리기 한다
이미지내에 들어가도록(듯이), 랜덤인 사이즈와 색을 가지는 엔을 5 개작성해, 그리기 한다.

// (5)랜덤인 타원을 그리기 한다
이미지내에 들어가도록(듯이), 랜덤인 사이즈와 색을 가지는 타원을 5 개작성해, 그리기 한다.

// (6)이미지의 표시, 키가 밀렸을 때에 종료
결과의 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예



다각형의 그리기 cvFillpoly

1 개가 있는 있어는 복수의 다각형에 의해서 단락지어진 영역을 전부 칠한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
  IplImage *img1, *img2, *img3;
  int npts[3] = { 3, 3, 4 };
  CvPoint **pts;

  // (1)이미지를 확보해 초기화한다
  img1 = cvCreateImage (cvSize (500, 500), IPL_DEPTH_8U, 3);
  img2 = cvCloneImage (img1);
  img3 = cvCloneImage (img1);
  cvZero (img1);

  // (2)다각형의 정점을 배열에 대입
  pts = (CvPoint **) cvAlloc (sizeof (CvPoint *) * 3);
  pts[0] = (CvPoint *) cvAlloc (sizeof (CvPoint) * 3);
  pts[1] = (CvPoint *) cvAlloc (sizeof (CvPoint) * 3);
  pts[2] = (CvPoint *) cvAlloc (sizeof (CvPoint) * 4);
  pts[0][0].x = 0 + 50;
  pts[0][0].y = int (200 * 1.73) + 50;
  pts[0][1].x = 200 + 50;
  pts[0][1].y = 0 + 50;
  pts[0][2].x = 400 + 50;
  pts[0][2].y = int (200 * 1.73) + 50;
  pts[1][0].x = 0 + 50;
  pts[1][0].y = 0 + 50;
  pts[1][1].x = 200 + 50;
  pts[1][1].y = int (200 * 1.73) + 50;
  pts[1][2].x = 400 + 50;
  pts[1][2].y = 0 + 50;
  pts[2][0].x = 200;
  pts[2][0].y = int (200 * 1.73 / 2.0) + 50;
  pts[2][1].x = 200 + 50;
  pts[2][1].y = int (200 * 1.73 * 1 / 4.0) + 50;
  pts[2][2].x = 200 + 100;
  pts[2][2].y = int (200 * 1.73 / 2.0) + 50;
  pts[2][3].x = 200 + 50;
  pts[2][3].y = int (200 * 1.73 * 3 / 4.0) + 50;
```

```
  // (3)다각형을 그리기 한다
  cvFillPoly (img1, pts, npts, 1, CV_RGB (255, 0, 0));
  cvFillPoly (img2, pts, npts, 2, CV_RGB (255, 0, 0));
  cvFillPoly (img3, pts, npts, 3, CV_RGB (255, 0, 0));

  // (4)이미지의 표시, 키가 밀렸을 때에 종료
  cvNamedWindow ("Fillpoly1", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Fillpoly1", img1);
  cvNamedWindow ("Fillpoly2", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Fillpoly2", img2);
  cvNamedWindow ("Fillpoly3", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Fillpoly3", img3);
  cvWaitKey (0);

  cvDestroyWindow ("Fillpoly1");
  cvDestroyWindow ("Fillpoly2");
  cvDestroyWindow ("Fillpoly3");
  cvReleaseImage (&img1);
  cvReleaseImage (&img2);
  cvReleaseImage (&img3);
  cvFree (&pts[0]);
  cvFree (&pts[1]);
  cvFree (&pts[2]);
  cvFree (pts);

  return 0;
}
```

// (1)이미지를 확보해 초기화한다
이 샘플에서는, 외부 이미지를 읽어들이지 않고 , 프로그램 내부에서 이미지를 준비한다. 또, 작성된 칼라 이미지를, 흑(0)(으)로 전부 칠해 초기화한다.

// (2)다각형의 정점을 배열에 대입
그리기하기 위한 다각형을 정의한다. 3 정점의 다각형(삼각형)을 두 개,4 정점의 다각형(사각형)을 하나, 분의 정점을, CvPoint 의 이차원 배열 pts 에 대입한다.

// (3)다각형을 그리기 한다
함수 cvFillPoly()의 4 번째의 인수는, 주어진 배열 가운데, 실제로 그리기 되는 다각형의 개수를 나타내고 있어 이번은, 세 개의 다각형(를 나타내는 정점 좌표)이 여라고 있으므로, 이 인수가 취할 수 있는 범위는 1~3(이)가 된다. 이것은, 내부적으로 배열의 인덱스로서 이용되므로,0(이)나, 준비된 다각형수를 넘는 값을 지정할 수 없다.

// (4)이미지의 표시, 키가 밀렸을 때에 종료
결과의 이미지를 표시해, 무엇인가 키가 밀릴 때까지 기다린다. 1 번째의 이미지는, 단지 삼각형의 다각형을 표시하는 것만으로 있지만,2 번째의 이미지에서는, 두 개의 삼각형 다각형(1 번째의 이미지의 다각형과 그것을 상하 반전한 다각형)의 거듭해 맞댐이 표시되고 있다.겹친 장소는, 다각형의 칠부수기를 하지 않는다. 3 번째의 이미지에서는, 한층 더

사각형(능형)의 다각형이 추가되고 있어 세 개의 다각형이 겹친 장소에서는, 통상대로 발라 부수기를 하고 있는 것이 안다.

실행 결과예

철다각형의 그리기 cvFillConvexpoly

1 개의 철다각형을 전부 칠한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
  int i;
  IplImage *img1, *img2;
  CvPoint *pts;

  // (1)이미지 영역을 확보해 초기화한다
  img1 = cvCreateImage (cvSize (400, 400), IPL_DEPTH_8U, 3);
  cvZero (img1);
  img2 = cvCloneImage (img1);

  // (2)다각형의 정점을 배열에 대입
  pts = (CvPoint *) cvAlloc (sizeof (CvPoint) * 4);
  pts[0].x = 100;
  pts[0].y = 100;
  pts[1].x = 200;
  pts[1].y = 150;
  pts[2].x = 140;
  pts[2].y = 300;
  pts[3].x = 30;
  pts[3].y = 200;
```

```
  // (3)철다각형을 그리기 한다
  cvFillConvexPoly (img1, pts, 4, CV_RGB (255, 0, 0), CV_AA, 0);
  cvFillConvexPoly (img1, pts, 4, CV_RGB (0, 255, 0), CV_AA, 1);

  // (4)다각형의 좌표를 1 비트 오른쪽 시프트 시키고, 그리기 한다
  for (i = 0; i < 4; i++) {
    pts[i].x >>= 1;
    pts[i].y >>= 1;
  }
  cvFillConvexPoly (img2, pts, 4, CV_RGB (0, 0, 255), CV_AA, 0);

  // (5)이미지의 표시, 키가 밀렸을 때에 종료
  cvNamedWindow ("Fillconvexpoly1", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Fillconvexpoly1", img1);
  cvNamedWindow ("Fillconvexpoly2", CV_WINDOW_AUTOSIZE);
  cvShowImage ("Fillconvexpoly2", img2);
  cvWaitKey (0);

  cvDestroyWindow ("Fillconvexpoly1");
  cvDestroyWindow ("Fillconvexpoly2");
  cvReleaseImage (&img1);
  cvReleaseImage (&img2);
  cvFree (&pts);

  return 0;
}
```

// (1)이미지 영역을 확보해 초기화한다
이 샘플에서는, 외부 이미지를 읽어들이지 않고 , 프로그램 내부에서 이미지를 준비한다. 또, 작성된 칼라 이미지를, 흑(0)(으)로 전부 칠해 초기화한다.

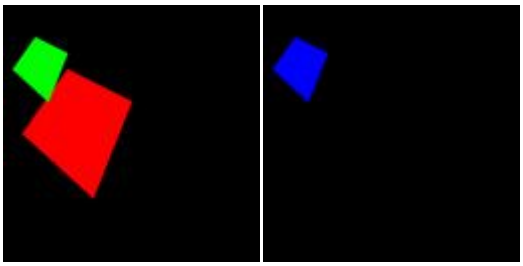
// (2)다각형의 정점을 배열에 대입
4 정점의 철다각형(사각형)을 하나 그리기 하기 위해서, 그 다각형의 정점 좌표를 정의한다.

// (3)철다각형을 그리기 한다
위에서 정의한 철다각형을 그리기 한다. 함수 cvFillConvexPoly()의 마지막 인수는, 정점 좌표의 Shift 양을 나타내고 있어 예를 들면, 여기에"1"(을)를 지정하면, 좌표치의 하위 1 비트는 소수점 이하를 나타내는 비트이다고 보여진다. 즉, 각 정점 좌표가 1 비트 오른쪽 시프트 되었다(좌표치가 1/2(이)가 된) 다각형과 거의 같은 것이 표시된다. 여기에서는, 적색으로 지정 좌표 그대로의 다각형을, 녹색으로 마지막에 인수에"1"(을)를 지정했을 경우의 다각형을 그리기 한다. 또, 그 윤곽이 수평인 스캔 라인과 2 회이하 밖에 교차하지 않는 듯한 단순한 다각형을 가정하고 있기 때문에, 이 함수 cvFillConvexPoly()(은)는, 함수 cvFillPoly()보다 고속으로 동작한다.

// (4)다각형의 좌표를 1 비트 오른쪽 시프트 시키고, 그리기 한다
비교를 위해서, 각 좌표치를 오른쪽으로 1 비트 시프트 시킨 다각형을, 별이미지에 청색으로 그리기 한다.

// (5)이미지의 표시, 키가 밀렸을 때에 종료
결과의 이미지 각각을 표시해, 무엇인가 키가 밀릴 때까지 기다린다.

실행 결과예



텍스트의 그리기 cvInitFont, cvPutText

폰트를 초기화하고, 텍스트를 그리기 한다

샘플 코드

표시의 변환

```
#include <cv.h>
#include <highgui.h>
#include <time.h>

int
main (int argc, char **argv)
{
  int i;
  IplImage *img;
  CvRNG rng = cvRNG (time (NULL));
  CvScalar rcolor;
  int irandom;
  int font_face[] = {
    CV_FONT_HERSHEY_SIMPLEX,
    CV_FONT_HERSHEY_PLAIN,
    CV_FONT_HERSHEY_DUPLEX,
    CV_FONT_HERSHEY_COMPLEX,
    CV_FONT_HERSHEY_TRIPLEX,
    CV_FONT_HERSHEY_COMPLEX_SMALL,
    CV_FONT_HERSHEY_SCRIPT_SIMPLEX,
    CV_FONT_HERSHEY_SCRIPT_COMPLEX
  };
  CvFont font[16];

  // (1)이미지를 확보해 초기화한다
  img = cvCreateImage (cvSize (400, 500), IPL_DEPTH_8U, 3);
  cvZero (img);
```