INTRODUCTION

AVW is a comprehensive set of imaging functions which facilitates full exploration and analysis of multidimensional, multimodality biomedical image data sets. The software also facilitates the development and implementation of advanced imaging algorithms and techniques and easy integration of these into specific applications solutions by imaging software developers. The extensive functionality and interactive speed of this software are unequivocally the key features which distinguish it from other image processing and visualization packages.

LIBRARY ORGANIZATION AND FUNCTIONALITY DESCRIPTIONS

The AVW imaging functions are packaged into a library that provides developers with the power of its predecessor, the ANALYZE system, at a callable C language function level. It is extensible, allowing developers to write their own specialized code while still making full use of the imaging capabilities of the package. The functions are completely user-interface independent, allowing for implementation at the applications level in different interface and operating environments. Each library function has a well defined calling sequence utilizing standardized AVW parameters and data structures, described in the next section. While only one linkable library is provided, the functionality in AVW can be separated into six distinct functional groups: ImageIO, Transform, Process, Segment, Analysis and Visualize. The contents of these groups are briefly outlined next.

Resource Functions

The Resource functions include the basic routines to create and destroy many of the *AVW* structures. Also included are list management, verification, and byte swapping routines.

ImageIO Functions

The ImageIO functions provide a user extendable interface which allows the reading and writing of images in many common formats.

Transform Functions

The Transform functions include 2-D and 3-D spatial transformations, densitometric transformations and mathematical transformations. The functions necessary for oblique and curved sectioning, wavelet enhancement and compression transformations, and image registration / fusion are also included in this group.

Process Functions

The Process functions provide a wide variety of histogram operations, spatial / convolution filtering, 3-D FFT's, 3-D Fourier domain filtering and 3-D deconvolution, including fast nonlinear iterative methods.

Segment Functions

The Segment functions include thresholding, 2-D and 3-D region growing, automated boundary detection, creation of object maps, multiresolution decomposition, morphological processing and multispectral classification.

Analysis Functions

The Analysis functions include extraction of line profiles, sampling of ROI's which includes the computation of size, density, shape and texture parameters (e.g., area, mean, circularity, fractal signature).

Visualize Functions

The Visualize functions provide all operations necessary to support 3-D display using volume and surface rendering. Supported algorithms for volume rendering include depth-only shading, depth gradient shading, voxel gradient shading, integrated surface projection, maximum intensity projection and summation projection. Multiple object creation and rendering, color transparency (24-bit), image extraction tools, and image measurement tools are supported in conjunction with the volume rendering functions.

IMAGE DATA TYPES AND DATA STRUCTURES

The AVW library of functions was designed to handle image and volume data in a fast and efficient manner while providing the functions useful for image display, processing, and measurement. An image is a 2-D array of values called pixels which are ordered in lines, and a volume is a 3-D array of values called voxels ordered in images. In this design, multiple types of volumes, referred to as bands, spectra or timepoints, are represented by an array of volumes. Since in multi-volume data sets each volume has its own structure, multiple bands are not required to have the same dimensions or data type. Image and volume structures have been designed to make function calls simple by requiring fewer parameters. These structures all assume that an image or volume is stored in contiguous memory.

AVW_Image

Images are represented by the following structure:

```
typedef struct
   {
   void *Mem;                        /* Pointer to the first pixel of the image */
   int DataType;                     /* Identifies the data type, unsigned char, int, etc... */
   unsigned int Width, Height;       /* Number of columns and rows in an image */
   unsigned int BytesPerPixel;       /* Number of bytes per pixel */
   unsigned int BytesPerLine;        /* BytesPerPixel * Width */
   unsigned int BytesPerImage;       /* BytesPerPixel * Width * Height */
   unsigned int PixelsPerImage;      /* Width * Height */
   AVW_Colormap *Colormap;           /* Pointer to colormap structure */
   char *Info;          /* Pointer to information character string */
   unsigned int *YTable; /* Array of offsets to each row of an image */
   } AVW_Image;
```

AVW_Volume

The volume structure is as follows:

```
typedef struct
   {
   void *Mem;                        /* Pointer to the first voxel of the volume*/
   int DataType;                     /* Identifies the data type, unsigned char, int, etc... */
   int Width, Height, Depth;         /* Number of columns, rows, and images in a volume*/
   unsigned int BytesPerPixel        /* Number of bytes per pixel */
   unsigned int BytesPerLine;        /* BytesPerPixel * Width */
   unsigned int BytesPerImage;       /* BytesPerPixel * Width * Height */
   unsigned int PixelsPerImage;      /* Width * Height */
   unsigned int BytesPerVolume;      /* BytesPerImage * Depth */
   unsigned int VoxelsPerVolume;     /* Width * Height * Depth */
   AVW_Colormap *Colormap;           /* Pointer to colormap structure */
   char *Info;          /* Pointer to information character string */
   unsigned int *ZTable; /* Array of offsets to each image in the volume */
   unsigned int *YTable; /* Array of offsets to each row of an image */
   } AVW_Volume;
```

AVW_Colormap

The AVW_Colormap structure, which is a part of the AVW_Image and AVW_Volume structures, contains a color lookup table, which is NULL for grey scale data. This structure is defined as:

```
typedef struct
   {
   int Size;                   /* Size of Red, Green, and Blue Arrays */
   unsigned char *Red;
   unsigned char *Green;
   unsigned char *Blue;
```

```
} AVW_Colormap;
```

Info Strings

The character Info string is used to store any relevant information about the volume or image which is not contained in the AVW_Volume or AVW_Image structures themselves. Character strings and numeric values may be stored in this string with the following functions:

```
int AVW_PutStringInfo(MatchString, String, InfoString)
char *MatchString;
char *string;
char *InfoString;

int AVW_PutNumericInfo(MatchString, Value, InfoString)
char *MatchString;
double Value;
char *InfoString;
```

The MatchString is used to identify the information so that it can be retrieved upon request. Examples of values which would be stored in this string include voxel dimensions, units of measurement, maximum and minimum intensity values, processing history, and patient information. Functions to extract values or strings from the Info string are also provided:

```
char *AVW_GetStringInfo(MatchString, InfoString)
char *MatchString, *InfoString;

double AVW_GetNumericInfo(MatchString, InfoString)
char *MatchString, *InfoString;
```

Create

Functions also exist for creating an AVW_Image or AVW_Volume structure from parameters supplied by the user:

```
AVW_Image *AVW_CreateImage(mem, width, height, type)
void *mem;
int width, height, type;

AVW_Volume *AVW_CreateVolume(mem, width, height, depth, type)
void *mem;
int width, height, depth, type;
```

Mem is a pointer to the raw image or volume. If mem is equal to NULL the memory is allocated by the function and returned. The type parameter refers to the supported data types. This value is the same as the DataType element of the AVW_Image and AVW_Volumes structures and can have the following values.

```
#define AVW_UNSIGNED_CHAR      (1<<0)  /* 1   */
#define AVW_SIGNED_CHAR        (1<<1)  /* 2   */
#define AVW_UNSIGNED_SHORT     (1<<2)  /* 4   */
#define AVW_SIGNED_SHORT       (1<<3)  /* 8   */
#define AVW_UNSIGNED_INT       (1<<4)  /* 16  */
#define AVW_SIGNED_INT         (1<<5)  /* 32  */
#define AVW_FLOAT              (1<<6)  /* 64  */
#define AVW_COMPLEX            (1<<7)  /* 128 */
#define AVW_COLOR              (1<<8)  /* 256 */
```

Most of the data types are self explanatory.

Complex pixels or voxels are represented by two floating point values, the real and imaginary components.

Color Images

Color images are represented by three consecutive bands which correspond to the red, green, and blue components of the color image. Each value is an unsigned char. The *Mem* element of the image structure points to the first value of the red band. This is followed by the rest of the red band values, which are followed by the green and blue band values.

Destroy

Images and volumes are released by the following functions:

```
void AVW_DestroyImage(image)
AVW_Image *image;

void AVW_DestroyVolume(volume)
AVW_Volume *volume;
```

MEMORY USAGE

AVW provides a mechanism for reusing and allocating structures. Though this document will only discuss *AVW_Images,* the same rules apply to *AVW_Volumes* and other *AVW* structures.

Many *AVW* functions return an *AVW_Image* and also have an *out_image* as part of the function's parameter list.

*AVW_Image *AVW_Function(in_image, out_image)*

Out_image is provided as a method of reusing an existing *AVW_Image.*

Reuse is possible if, and only if, the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable and not *NULL, out_image* will be released and reallocated.

A *NULL out_image* passed in as a parameter, guarantees creation of a satisfactory *AVW_Image* which is returned by the function.

Two typical ways of using this capability through the function calls are:

out_image = AVW_Function(in_image, out_image);

and

out_image = AVW_Function(in_image, NULL);

In the first call *out_image* may be reused and in the second call it is created. In any case the returned pointer should always be used as the results of the function.

Out_image will be freed if an error occurs in the function and *NULL* will be returned.

In the following example *image* appears as both the returned pointer and as an input parameter. The first time *AVW_GetOrthogonal()* is called, *image* will be *NULL,* and *AVW_GetOrthogonal()* will call *AVW_CreateImage()* to allocate the required *AVW_Image.* On successive calls to *AVW_GetOrthogonal(), image* will contain a valid and reuseable *AVW_Image*.

```
#include "AVW.h"

test(volume)
AVW_Volume *volume;
   {
   AVW_Image *image = NULL;
   int i;

   for(i=0;i<volume->Depth;++i)
      {
      image = AVW_GetOrthogonal(volume, AVW_TRANSVERSE, i, image);

      if(!image)
         {
         AVW_Error("test");
         return;
         }
      your_display_function(image);
      }
   if(image) AVW_DestroyImage(image);
   }
```

FUNCTION SPECIFICATION and CALLING CONVENTIONS

The AVW library function names start with AVW_ followed by the name of the function which has the first letter of each word capitalized. The parameters for the functions follow the pattern:

AVW_FunctionName(input parameters, ..., output parameter(s))

A pointer to the output image or volume is also returned by most functions. If the output image or volume had not been allocated prior to the subroutine call, it would be created, copied into and returned to the user. If the output image or volume already existed but was the wrong size or type it would be freed, reallocated and returned to the user. In the case where the output image or volume could not be allocated, a NULL would be returned and the appropriate error number and error message would be set.

ERROR HANDLING

All of the AVW functions support error checking in a consistent manner, providing the user with an error number and an error message if a function fails. See the AVW_Error.h include file for a detailed list of each error code which may be returned.

AVW IMAGE I/O

The AVW Image IO functions are designed to provide a single programmer interface to images of all file types. For supported file formats the files are opened without regard to the data format and images are read and returned in the AVW_Image or AVW_Volume structure. In addition the developer can add support for other formats to the Image IO functions.

A listing and short description of all of the AVW Image IO functions is provided.
AVW_OpenImageFile - opens a named image file
AVW_SeekImageFile - seeks to a specified image
AVW_ReadImageFile - reads an image from the file
AVW_ReadVolume - reads a volume from the file
AVW_WriteImageFile - writes an image to a file
AVW_WriteVolume - writes a volume to a file
AVW_CreateImageFile - creates an image file of specified format

AVW_ExtendImageFile - extends AVW ImageIO functions
to support additional image file formats.

Image File Formats

The following file formats are directly supported by AVW Image File IO routines:

AnalyzeImage
AnalyzeScreen
SunRaster
AVW_VolumeFile
GE9800 (Read Only)
GESigna (Read Only)
GE Advantage (Read Only)
Imatron (Read Only)
SiemensCT (Read Only)
INTERFILE (Read Only)
ACRNEMA (Read Only)
PAPYRUS (Read Only)
SGIrgb
SGIbw
PPM
PGM
YUV
BMP
PIC
PICKERMRI (Read Only)
SMIS (Read Only)
CTI (Read Only)
AVW_ImageFile

Support for some of the formats is limited to Read Only. Files cannot be created or written in these formats.

AVW_VolumeFile

Most medical image formats do not support 3-D directly, since each slice in a study is written to a separate file. The AVW ImageIO routines provide a psuedo format for treating such groups of files as a single 3-D entity.

A text file which lists the files to be used for each slice in the file can be created. The first line of the file must be "AVW_VolumeFile", and each subsequent line the name of a file. For example:

```
AVW_VolumeFile
ST001SER002.001
ST001SER002.002
ST001SER002.003
ST001SER002.004
```

The name of this textfile is passed to AVW_OpenImageFile(), and it checks the rest of the files for consistancy. Alternatively, all the image files may be placed in a subdirectory (with no other files) and the name of the directory can be passed to the same routine. Wildcard strings can also be used.

```
sdb = AVW_OpenImageFile("ST001SER001.???", "r");
```

When a directory name or wild card is used as a file name the file list is sorted alphabetically.

Image File IO Data Structures

The following structures are defined in AVW_ImageFile.h:

```
typedef struct
{
        char *FileName;
        char *FileModes;
        int DataFormat;
        int DataType;
        int Width;
        int Height;
        int Depth;
        int NumVols;
        int BitsPerPixel;
        int BytesPerPixel;
        int BytesPerLine;
        int BytesPerImage;
        int BytesPerVolume;
        int BytesPerFile;
        int PixelsPerImage;
        int VoxelsPerVolume;
        int VoxelsPerFile;
        int CurrentSlice;
        int CurrentVolume;
        AVW_Colormap *Colormap;

        void *NativeData;   /* a pointer to the native header,
              or other format specific data */
        char *Info;
} AVW_ImageFile;
```

A pointer to this structure is returned by AVW_OpenImageFile() in the same way that fopen() returns a pointer to a FILE. The pointer is passed to the other AVW Image IO functions, in the same way that a FILE pointer is passed to fread(), fwrite(), fseek(), and fclose().

Supporting other formats

The *AVW_ExtendImageFile()* function is used to add support for additional formats. The elements of an *AVW_ExtendIO* structure must be initialized. This structure contains a list of functions for interacting with the file format.

```
typedef struct
{
        char        *Extension;
        char        *Description;
        int         MagicNumber;
        int         Properties;
        AVW_ImageFile  (*Open)();
        int         (*Seek)();
        AVW_Image      (*Read)();
        int         (*Write)();
        int         (*Close)();
        AVW_ImageFile  (*Create)();
        int         (*Query)();
} AVW_ExtendIO;
```

For example,

AVW_ReadImageFile()
calls the "(*Read)()" for for the appropriate format.

int AVW_ExtendImageFile(AVW_ExtendIO * fl)

```
char   *Extension;     /* text extension, if any by which
                 the file is known  */


char   *Description;   /* the text name by which the format
                 will be known. */

int    MagicNumber;  /*  magic number (if any) by which
                 the file type is identified */
int    Properties;

               /*  an | mask of properties that are
                 supported by the format or this
                 implementation */

AVW_ImageFile  (*Open)();

               /* the Open() function, should return
                 a pointer to an AVW_ImageFile
                 structure, AVW_OpenImageFile()
                 invokes and returns the pointer
                 returned by this function */

int          (*Seek)();

               /* positions the file stream so that
                 the next call to AVW_ReadImageFile()
                 returns  the specified image. */

AVW_Image     (*Read)();

              /* reads and returns an AVW_Image
                from the file */

int          (*Write)();

             /* writes an AW_Image to the file */


int          (*Close)();

             /* closes the file, and releases
               format specific data structures */


AVW_ImageFile  (*Create)();

             /* creates a file given format and image */


int          (*Query)();
```

```
/* given a file name returns TRUE or
  FALSE to indicate if file is of
  this Format */
```

Extending for the TIFF format.

In the $BIR/AVW/extras/io/expand_io/TIFF directory the file avw_tif.c contains source and instructions for expanding the ImageIO support to include the Tiff format.

Extending AVW Image IO to support the TIFF format is done by initializing the elements of an AVW_ExtendIO structure and passing a pointer to that structure to AVW_ExtendImageFile().

For example, the following shows how AVW can be extended to support TIFF images.

```
AVW_ExtendIO *fl;
fl=(AVW_ExtendIO *)malloc(sizeof(AVW_ExtendIO));

        fl->Extension         = ".tif";
        fl->Description       = "TIFF";
        fl->MagicNumber              = AVW_NO_MAGIC_NUMBER;
        fl->Properties        = AVW_SUPPORT_UNSIGNED_CHAR|
              AVW_SUPPORT_COLOR|
              AVW_SUPPORT_READ|
              AVW_SUPPORT_WRITE ;
        fl->Open     = avw_open_tiff_image;
  fl->Seek     = avw_seek_tiff_image;
  fl->Read     = avw_read_tiff_image;
  fl->Write    = avw_write_tiff_image;
  fl->Close    = avw_close_tiff_image;
  fl->Create   = avw_create_tiff_image;
  fl->Query    = avw_query_tiff_image;

  AVW_ExtendImageFile(fl);
  free(fl);
```

Where the following user supplied functions call functions in TiffLib. The application must link to TiffLib as well.

```
int avw_query_tiff_image(filename)
char *filename;
```

is a user supplied function which returns true if the named file is a Tiff file, otherwise false.

```
AVW_ImageFile *avw_open_tiff_image(filename, modes)
char *filename;
char *modes;
```

is a user supplied function which opens the tiff file and returns an initialized AVW_ImageFile structure.

```
int avw_seek_tiff_image(sdb,vol,slc)
```

```
AVW_ImageFile *sdb;
int vol,slc;
```

is a user supplied function which seeks to the specified volume and slice in a tiff file. For file formats that do not support multiple slices per file, this routine does nothing.

```
AVW_Image *avw_read_tiff_image(sdb, img)
AVW_ImageFile *sdb;
AVW_Image *img;
```

is a user supplied function which returns the current image after being called by AVW_OpenImage(). This routine assumes that sufficient memory for the entire image is at img->Mem.

```
int avw_write_tiff_image(sdb, img)
AVW_ImageFile *sdb;
AVW_Image *img;
```

is a user supplied function which writes the image to the file indicated by AVW_ImageFile *sdb.

```
int avw_close_tiff_image(sdb)
AVW_ImageFile *sdb;
```

is a user supplied function which closes the file and frees any memory which was allocated for format specific data structures.

ImageIO and Info Strings

The *AVW_ImageFile* structure contains an information string which is used to hold information that is not contained in the structure. Info strings are used to hold additional information which may be present in some formats. *VoxelWidth, VoxelHeight, and VoxelDepth* are used to carry the voxel dimensions if available.

AVW++

The Biomedical Imaging Resource has developed a C++ version of *AVW* (currently referred to as *AVW++*) in the form of a class library. In general, classes are based on the *AVW* structures and the methods are based on the *AVW* functions. These functions have not been rewritten but rather are called directly from within the methods, making this library a C++ "wrapper" for *AVW*.

The private data for most of the classes consists only of the corresponding *AVW* structures. This construct provides a layer of protection against direct manipulation of the data, i.e. data encapsulation. Each of the *AVW* functions has been matched with the appropriate class and is called via public methods of the same name or through the constructors, destructors and operator overload methods (=, +, -, *, /, ==, etc.) It should be noted since many of the methods consist of several lines of source code that do little more than call their corresponding *AVW* routines, the possible performance degradation brought on by the extra C++ layer can be limited by "inlining" the methods.

AVW++ was developed to support internal C++ projects, and does not have the same documentation and support found in *AVW*. It is, however, recommended that any C++ developers requiring *AVW* functionality consider looking into *AVW++* first.

Classes

The following classes are found in *AVW++:*

A_Colormap A_ObjectMap

A_FPoint2	A_Point2
A_FPoint3	A_Point3
A_FPointList2	A_PointList2
A_FPointList3	A_PointList3
A_FilterCoeffs	A_PointValueList
A_Histogram	A_Rect2
A_Image	A_Rect3
A_ImageFile	A_RenderedImage
A_IntensityStats	A_Surface
A_Line2	A_Volume
A_Line3	A_VolumeRenderParms
A_Matrix	K_String

To inquire about acquiring *AVW++* contact the Biomedical Imaging Resource, Mayo Foundation.

PROGRAMMING EXAMPLES

Creating and deleting an AVW_Image.

```
#include "AVW.h"

main()
   {
   AVW_Image *image;
   int w = 64, h = 64;

   image = AVW_CreateImage(NULL, w, h, AVW_UNSIGNED_CHAR);
      .
      .
      .
   AVW_DestroyImage(image);
   }
```

Example #2

Wrapping an AVW_Volume structure around a block of memory.

```
#include "AVW.h"

sub(mem, xsize, ysize, zsize, datatype)
void *mem;
int xsize, ysize, zsize, datatype;
   {
   AVW_Volume *volume;

   volume = AVW_CreateVolume(mem, xsize, ysize, zsize, datatype);
      .
      .
      .
   volume->Mem = NULL;    /* prevents the freeing of memory */
   AVW_DestroyVolume(volume);
   }
```

Example #3

Error Checking.

```
#include "AVW.h"

main()
   {
   AVW_Image *image;
```

```
  int w = 64, h = 64;

  image = AVW_CreateImage(NULL, w, h, AVW_UNSIGNED_CHAR);

  if(image == NULL)
    {
    AVW_Error("program: AVW_CreateImage");
    exit(1);
    }
        .
        .
        .

  if(!AVW_SetImage(image, 0.0))
    {
    AVW_Error("program: AVW_SetImage");
    exit(1);
    }
        .
        .
        .
  }
```

Example #4 Info Strings.

```
#include "AVW.h"

main()
  {
  AVW_Volume *volume;
  double voxel_width;
        .
        .
        .

  voxel_width = AVW_GetNumericInfo("VoxelWidth", volume->Info);
  volume->Info = AVW_PutStringInfo("Processed By",
                    "Your Name Here", volume->Info);
  }
```

Example #5 Accessing a Volume Voxel. (Slow)

```
#include "AVW.h"

main()
  {
  AVW_Volume *volume;
  AVW_Point3 point;
  double voxel_value;
        .
        .
        .

  point.X = volume->Width / 2;
  point.Y = volume->Height / 2;
  point.Z = volume->Depth / 2;
```

```
   voxel_value = AVW_GetVoxel(volume, &point);

   }
```

Example #6

Accessing Volume Voxels. (Fast)

```
#include "AVW.h"

sub(volume)
AVW_Volume *volume;
   {
   register int i;

   i = volume->VoxelsPerVolume;

   switch(volume->DataType)
      {
   case AVW_UNSIGNED_CHAR:
         {
         register unsigned char *ptr;

         ptr = (unsigned char *) volume->Mem;

         while(i--) *ptr++ = 0;
         }
      break;

   case AVW_SIGNED_CHAR:
         {
         register char *ptr;

         ptr = (char *) volume->Mem;

         while(i--) *ptr++ = 0;
         }
      break;

   default:
      return(AVW_FAIL);
      break;
      }

   return(AVW_SUCCESS);
   }
```

Example #7

Orthogonal Slices.

```
#include "AVW.h"

main()
   {
   AVW_Image *image;
   AVW_Volume *volume;
   int slice = 0;
         .
         .
         .
```

```
   image = AVW_GetOrthogonal(volume, AVW_TRANSVERSE, slice, NULL);

   if(image)
      {
         .
         .
         .
      AVW_PutOrthogonal(image, volume, AVW_TRANSVERSE, slice);
      AVW_DestroyImage(image);
      }
   }
```

Example #8

Re-using Memory.

```
#include "AVW.h"

main()
   {
   AVW_Image *image = NULL;
   AVW_Volume *volume;
   int slice = 0;
         .
         .
         .

   for(slice = 0 ; slice < volume->Width ; ++slice)
      {
      image = AVW_GetOrthogonal(volume, AVW_SAGITTAL, slice, image);

      if(image)
         {
            .
            .
            .
         AVW_PutOrthogonal(image, volume, AVW_SAGITTAL, slice);
         }
      }

   AVW_DestroyImage(image);
   image = NULL;
   }
```

Example #9

Volume Rendering.

```
#include "AVW.h"
#include "AVW_Render.h"
#include "AVW_ObjectMap.h"

main()
   {
   AVW_Volume *vol = NULL;
   AVW_ObjectMap *omap = NULL;
   AVW_RenderParameters *r_param = NULL;

   float xangle = -90., yangle = 0., zangle = 0.;
```

```
      .
      .
      .
/*
 * AVW_InitializeRenderParameters() usually called when volume is loaded
 */

   r_param = AVW_InitializeRenderParameters(vol, omap, r_param);
      .
      .
      .

   while(something_to_do)
      {
         .
         .
         .
      r_param->ThresholdMinimum = 135;

      r_param->Matrix = AVW_RotateMatrix(r_param->Matrix, xangle, yangle, zangle, r_param->Matr

      if((rendered = AVW_RenderVolume(r_param, rendered)) == NULL)
         {
         AVW_Error("AVW_RenderVolume");
         break;
         }
         else
         {
         YourDisplayRoutine(rendered->Image);
         }
            .
            .
            .
      if(option_to_load_object_map_selected)
         {
         if((omap = AVW_LoadObjectMap("filename.obj")))
            {
            r_param = AVW_InitializeRenderParameters(vol, omap, r_param);
            }
         }
            .
            .
            .
      }
   }
```

Example #10

Converting specified volume from one file format to another.

convert_vol infile invol outfile outformat

```
#include <stdio.h>

#include          "AVW.h"
#include   "AVW_ImageFile.h"
```

```
main(ac,av)
int ac;
char **av;
{
AVW_ImageFile *sdbi= AVW_NULL, *sdbo= AVW_NULL;
AVW_Volume *rvol = AVW_NULL;
int i, invol;
char *infilename, *outfilename, *outformat, mess[80];;

if(ac != 5)
{
  printf("Usage: cv infile infile_number outfilename outformat \n");
  exit(0);
}
infilename  = av[1];

            /* Allow user to reference volumes from 1 to n */
invol       = atoi(av[2]) - 1;
if(invol <=0)
{
            printf("positive non-zero volume number 0);
            exit(0);
}
outfilename = av[3];

/* "AnalyzeImage", "SunRaster", "PPM", etc.*/
outformat   = av[4];

if((sdbi = AVW_OpenImageFile(infilename,"r"))==AVW_NULL)
{
            sprintf(mess, "Open of <%s> failed\n", infilename);
            AVW_Error(mess);
            exit(0);
}

if(invol >= sdbi->NumVols)
{
            sprintf(mess, "%s only has %d volumes \n",
               infilename,sdbi->NumVols);
            AVW_Error(mess);
            exit(0);
}

if((rvol = AVW_ReadVolume(sdbi,invol,rvol))==AVW_NULL)
{
            AVW_Error("ReadVolume failed\n");
            exit(0);
}

if((sdbo = AVW_CreateImageFile(outfilename,
         outformat, rvol->Width,
         rvol->Height,rvol->Depth,
         rvol->DataType)) == AVW_FAIL)
{

  sprintf(mess, "Create of %s fails. \n",outfilename );
  AVW_Error(mess);
```

```
  exit(0);
}

if((i = AVW_WriteVolume(sdbo, 0, rvol))!=AVW_SUCCESS)
{
  sprintf(mess, "WriteVolume to %s fails. \n",outfilename );
  AVW_Error(mess);
  exit(0);
}

AVW_CloseImageFile(sdbi);
AVW_CloseImageFile(sdbo);
}
```

Example #11

Converting the first image from any file to PPM format.

toppm infile outfile

```
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <memory.h>

#include "AVW.h"
#include "AVW_ImageFile.h"

main(ac,av)
int ac;
char **av;
{

AVW_ImageFile *sdb_in=AVW_NULL, *sdb_out=AVW_NULL;
AVW_Image *Img=AVW_NULL;
int  i,j,ret;
char *infile, outfile[128], mess[80];



if(ac != 3)
{
        printf("usage: toppm infile outfile");
        exit(0);
}

infile = av[1];
/* Open input file */
if((sdb_in = (AVW_ImageFile *)AVW_OpenImageFile(infile,"r"))==NULL)
{
        sprintf(mess, "Error opening:<%s> \n",infile);
        AVW_Error(mess);
        exit(0);
}

/* Seek to First Image */
if((ret = AVW_SeekImageFile(sdb_in, 0, 0))!=AVW_SUCCESS)
{
        sprintf(mess,"Error seeking v:%d s:%d\n",0,0);
```

```
        AVW_Error(mess);
        exit(0);
}

if((Img=AVW_ReadImageFile(sdb_in,Img))==AVW_NULL)
{
        sprintf(mess,"Error reading v:%d s:%d",i,j);
        AVW_Error(mess);
        exit(0);
}

sprintf(outfile,"%s.PPM", av[2]);

if((sdb_out = AVW_CreateDB(outfile, "PPM",
                        Img->Width, Img->Height,
                        1,Img->DataType))==AVW_NULL)
{
        sprintf(mess,"Error Creating:<%s> \n", outfile);
        AVW_Error(mess);
        exit(0);
}
AVW_WriteImageFile (sdb_out, Img);
AVW_CloseImageFile(sdb_out);
AVW_CloseImageFile(sdb_in);

}
```

Example #12

C++ code fragments that use AVW++.

```
{
  K_String fileName("/images/demos/sample.hdr");      // creates file name string
  A_ImageFile myImageFile(fileName);                  // opens specified image file
  A_Image currImage(myImageFile.getImage(4, 2));      // constructor: slice 4, vol 2
  A_Image oldImage = currImage;                       // makes a copy of currImage

// the following flips currImage vertically, resizes it 4x, then dithers it to 64 colors
  currImage.flip(AVW_FALSE, AVW_TRUE).resize(400,400).dither(64);

  .
  .      // 2 other A_Image's called otherImage and newImage are constructed.
  .

// the following adds two images if they are not the same
  if (currImage != otherImage)                        // != operator is overloaded
    newImage = currImage + otherImage;                // + operator is overloaded
  .
  .      // an AVW_Volume* called anAVWVolume is created
  .
  A_Volume someVolume(anAVWVolume);      // constructor: directly from an AVW_Volume*

// the following puts currImage into someVolume oriented coronally at slice 4
  someVolume.putOrthogonal(currImage, AVW_CORONAL, 4);
  .
  .
  .
}                    // Note: at this point the destructor for each object constructed
                     // within the scope of these brackets is called, freeing allocated
                     // memory automatically, thus simplifying memory management.
```

ADDITIONAL PROGRAM EXAMPLES

Additional AVW example programs and tcl scripts may be found in the directory:
$AVW/extras

NAME AVW_AHEImage – performs 2D Adaptive Histogram Equalization

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_AHEImage(in_image, num_x_regions, num_y_regions, clip_maximum,
            clip_minimum, clip_fraction, out_image)
AVW_Image *in_image;
unsigned int num_x_regions, num_y_regions;
double clip_maximum, clip_minimum, clip_fraction;
AVW_Image *out_image;
```

DESCRIPTION

AVW_AHEImage() performs Adaptive Histogram Equalization on *in_image.* Adaptive Histogram Equalization is a process of adjusting the gray scale values of an image based on the histogram or count of pixel values in a localized region of the image. The purpose of this equalization is to enhance the viewable contrast in all areas of the image without regard to maintaining any strict mathematical relationship between gray scale value. This method creates a histogram of gray scale values for each set of predefined regions within the image, and adjusts the gray scale values within each region to increase the contrast locally.

NOTE: Value-based measurements should be avoided on Histogram Equalized images. The resulting gray scale values are no longer related between regions.

In_image specifies the image to be enhanced.

Num_x_regions and *num_y_regions* specify the number of regions along each axis.

Clip_maximum and *clip_minimum* specify a range of voxel values. All voxels with values greater than *clip_maximum* are set equal to *clip_maximum* before the gathering of the histogram information. Likewise, all voxels with values less than *clip_minimum* are set to *clip_minimum.*

Clip_fraction limits the contribution of any given gray scale value, and thereby reduces the enhancement of noise in the resulting image. This method is particularly effective for images which have subtle detail in both very bright (high-valued) and very dim (low-valued) regions of the image. It may also be used to expand the dynamic range of gray scale values in a region which has subtle but significant changes.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_AHEImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_AnisotropicAffineImage(), AVW_AnisotropicDiffusionImage(), AVW_LowpassFilterImage(), AVW_MedianFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_Image

NAME

AVW_AHEVolume – performs 3D Adaptive Histogram Equalization

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_AHEVolume(in_volume, num_x_regions, num_y_regions,
                num_z_regions, clip_maximum, clip_minimum,
                clip_fraction, out_volume)
AVW_Volume *in_volume;
unsigned int num_x_regions, num_y_regions, num_z_regions;
double clip_maximum, clip_minimum, clip_fraction;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_AHEVolume() performs Adaptive Histogram Equalization on *in_volume.* Adaptive Histogram Equalization is a process of adjusting the gray scale values of a volume based on the histogram or count of voxel values in a localized region of the volume. The purpose of this equalization is to enhance the viewable contrast in all areas of the volume without regard to maintaining any strict mathematical relationship between gray scale values. This method creates a histogram of gray scale values for each set of predefined regions within the volume, and adjusts the gray scale values within each region to increase the contrast locally.

NOTE: Value-based measurements should be avoided on Histogram Equalized volumes. The resulting gray scale values are no longer related between regions.

In_volume specifies the volume to be enhanced.

Num_x_regions, Num_y_regions and *num_z_regions* specify the number of regions along each axis.

Clip_maximum and *clip_minimum* specify a range of voxel values. All voxels with values greater than *clip_maximum* are set equal to *clip_maximum* before the gathering of the histogram information. Likewise, all voxels with values less than *clip_minimum* are set to *clip_minimum.*

Clip_fraction limits the contribution of any given gray scale value, and thereby reduces the enhancement of noise in the resulting volume. This method is particularly effective for volumes which have subtle detail in both very bright (high-valued) and very dim (low-valued) regions of the volume. It may also be used to expand the dynamic range of gray scale values in a region which has subtle but significant changes.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_AHEVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_LowpassFilterVolume(), AVW_MedianFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SigmanFilterVolume(), AVW_SobelFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_AddFPoint2 – adds a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_AddFPoint2(trace, point)
AVW_FPointList2 *trace;
AVW_FPoint2 *point;
```

DESCRIPTION

AVW_AddFPoint2() adds an *AVW_FPoint2*, *point*, to an *AVW_FPointList2*, *trace.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_AddFPoint2()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddFPoint2() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddFPoint3(), AVW_AddIPoint2(), AVW_AddIPoint3(), AVW_AddPoint2(), AVW_AddPoint3(), AVW_AddPointValue(), AVW_CopyPointList2() AVW_CreateFPointList2(), AVW_DestroyFPointList2(), AVW_RemoveFPoint2(), AVW_FPointList2, AVW_FPoint2

NAME

AVW_AddFPoint3 – adds a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_AddFPoint3(trace, point)
AVW_FPointList3 *trace;
AVW_FPoint3 *point;
```

DESCRIPTION

AVW_AddFPoint3() adds an *AVW_FPoint3*, *point*, to an *AVW_FPointList3*, *trace*.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_AddFPoint3()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddFPoint3() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddFPoint2(), AVW_AddIPoint2(), AVW_AddIPoint3(), AVW_AddPoint2(), AVW_AddPoint3(), AVW_AddPointValue(), AVW_CopyFPointList3(), AVW_CreateFPointList3(), AVW_DestroyFPointList3(), AVW_GetFPoint3(), AVW_RemoveFPoint3(), AVW_FPointList3, AVW_FPoint3

NAME

AVW_AddIPoint2 – adds a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_AddIPoint2(trace, point)
AVW_IPointList2 *trace;
AVW_IPoint2 *point;
```

DESCRIPTION

AVW_AddIPoint2() adds an *AVW_IPoint2*, *point* to an *AVW_IPointList2*, *trace*.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_AddIPoint2()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddIPoint2() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddIPoint3(), AVW_AddFPoint2(), AVW_AddFPoint3(), AVW_AddPoint2(), AVW_AddPoint3(), AVW_AddPointValue(), AVW_CopyIPointList2(), AVW_CreateIPointList2(), AVW_DestroyIPointList2(), AVW_GetIPoint2(), AVW_RemoveIpoint2(), AVW_IPointList2, AVW_IPoint2

NAME

AVW_AddIPoint3 – adds a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_AddIPoint3(trace, point)
AVW_IPointList3 *trace;
AVW_IPoint3 *point;
```

DESCRIPTION

AVW_AddIPoint3() adds an *AVW_IPoint3*, *point*, to an *AVW_IPointList3*, *trace.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_AddIPoint3()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddIPoint3() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddIPoint2(), AVW_AddFPoint2(), AVW_AddFPoint3(), AVW_AddPoint2(), AVW_AddPoint3(), AVW_AddPointValue(), AVW_CopyIPointList3(), AVW_CreateIPointList3(), AVW_DestroyIPointList3(), AVW_GetIPoint3(), AVW_RemoveIPoint3(), AVW_IPointList3, AVW_IPoint3

NAME

AVW_AddObject – adds an object to an object map

SYNOPSIS

#include "AVW_ObjectMap.h"

```
int AVW_AddObject(object_map)
AVW_ObjectMap *object_map;
```

DESCRIPTION

AVW_AddObject() increases the *object_map->NumberOfObjects* by one and initializes *object_map->Object[object_map->NumberOfObjects - 1].*

RETURN VALUES

If successful *AVW_AddObject()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_AddObject() will fail if:

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_CopyObjectMap(), AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_GetObject(), AVW_LoadObjectMap(), AVW_PutObject(), AVW_SaveObjectMap(), AVW_RemoveUnusedObjects(), AVW_RenderVolume(), AVW_ObjectMap

NAME

AVW_AddPoint2 – adds a point to a list structure

SYNOPSIS

#include "AVW.h"

```
int AVW_AddPoint2(trace, point)
AVW_PointList2 *trace;
AVW_Point2 *point;
```

DESCRIPTION

AVW_AddPoint2() adds an *AVW_Point2, point*, to an *AVW_PointList2, trace.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_AddPoint2()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddPoint2() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddFPoint2(), AVW_AddFPoint3(), AVW_AddIPoint2(), AVW_AddIPoint3(), AVW_AddPoint3(), AVW_AddPointValue(), AVW_CopyPointList2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_EditPointList2(), AVW_FillPointList2(), AVW_GetPoint2(), AVW_RemovePoint2(), AVW_PointList2, AVW_Point2

NAME

AVW_AddPoint3 – adds a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_AddPoint3(trace, point)
AVW_PointList3 *trace;
AVW_Point3 *point;
```

DESCRIPTION

AVW_AddPoint3() adds an *AVW_Point3, point*, to an *AVW_PointList3, trace.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_AddPoint3()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddPoint3() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddFPoint2(), AVW_AddFPoint3(), AVW_AddIPoint2(), AVW_AddIPoint3(), AVW_AddPoint2(), AVW_AddPointValue(), AVW_CopyPointList3(), AVW_CreatePointList3(), AVW_DestroyPointList3(), AVW_FillPointList3(), AVW_GetPoint3(), AVW_RemovePoint3(), AVW_PointList3, AVW_Point3

NAME

AVW_AddPointValue – adds a point and a value to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_AddPointValue(trace, point, value)
AVW_PointValueList *trace;
AVW_Point2 *point;
double value;
```

DESCRIPTION

AVW_AddPointValue() adds an *AVW_Point2, point,* and a *value* to an *AVW_PointValueList, trace.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_AddPointValue()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddPointValue() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddFPoint2(), AVW_AddFPoint3(), AVW_AddIPoint2(), AVW_AddIPoint3(), AVW_AddPoint2(), AVW_AddPoint3(), AVW_CopyPointValueList(), AVW_CreatePointValueList(), AVW_DestroyPointValueList(), AVW_GetPointValue(), AVW_RemovePointValue(), AVW_PointValueList, AVW_Point2

NAME

AVW_AddTreeChild – adds a point to a tree structure

SYNOPSIS

```
#include "AVW_Tree.h"

int AVW_AddTreeChild(tree, parent, child)
AVW_Tree *tree;
AVW_Point3 *parent, *child;
```

DESCRIPTION

AVW_AddTreeChild() adds an *AVW_Point3*, *child*, to an *AVW_Tree*, *tree*. The *parent* immediately preceeds the *child* in the *tree*. The *parent* point is used to search the *tree* and determine where the *child* is inserted.

These structures are defined in *AVW_Tree.h* and are used for skeletal tree information.

RETURN VALUES

If successful *AVW_AddTreeChild()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_AddTreeChild() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_CreateTree(), AVW_DestroyTree(), AVW_FindTreeIndex(), AVW_FindTreeStart(), AVW_LoadTree(), AVW_MakeTree(), AVW_PruneVolume(), AVW_SaveTree(), AVW_TreeAnalysis(), AVW_Point3, AVW_Tree

NAME AVW_AnisotropicAffineImage - performs a 2D anisotropic affine transformation

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_AnisotropicAffineImage(in_image, dt, iterations, out_image)
AVW_Image *in_image;
double dt;
int iterations;
AVW_Image *out_image;
```

DESCRIPTION

AVW_AnisotropicAffineImage() performs 2D affine anisotropic diffusion on the *in_image*, as specified in reference (1) below, using the implementation scheme described therein. *Dt* is a time step parameter which specifies how long the diffusion is allowed to run per iteration. A value of 0.25 is recommended for general use and will be used as the default if the supplied value is zero or negative. However, a value of 0.10 or less is required to guarantee stable behavior in all circumstances. The lower value will require more iterations (and hence will yield lower performance) but should be used if absolutely correct results are essential. *Iterations* specifies how many iterations to run the filter.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

REFERENCES

(1) Sapiro and Tannenbaum, "Affine Invariant Scale Space", Intl. Journal of Comp. Vision 11:1, pp25-44, 1993.

RETURN VALUES

If successful *AVW_AnisotropicAffineImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_AnisotropicDiffusionImage(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterImage(), AVW_MedianFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_Image

NAME

AVW_AnisotropicDiffusionImage – performs a 2D anisotropic diffusion transformation

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_AnisotropicDiffusionImage(in_image, iterations, kappa,
                                          bias_flag, out_image)
AVW_Image *in_image;
int iterations;
double kappa;
int bias_flag;
AVW_Image *out_image;
```

DESCRIPTION

AVW_AnisotropicDiffusionImage() performs 2D Anisotropic Diffusion on the *in_image*, as specified in references (1) and (2) below, using the exponential diffusion function described therein. *Iterations* specifies how many iterations to run the filter. *Kappa* specifies the gray level value difference above which the diffusion between two adjacent pixels becomes significantly suppressed (corresponding to the *kappa* in the equations in the references). *Bias_flag* specifies whether or not to perform biased anisotropic diffusion, as described in (2) and (3). A value of 1 specifies that biased anisotropic diffusion will be performed and 0 specifies that the biased option will not be performed.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

REFERENCES

(1) Perona and Malik, "Scale-Space and Edge Detection Using Anisotropic Diffusion", IEEE PAMI 12, pp 629-639, 1990.

(2) Gerig, Kubler, Kikinis, and Jolesz, "Nonlinear Anisotropic Filtering of MRI Data", IEEE Trans Med Img 11, pp 221-232, 1992.

(3) Nordstrom, "Biased anisotropic diffusion - A unified regularization and diffusion approach to edge detection", Image Vision Comput., vol. 8, no. 4, pp 318-327, 1990.

RETURN VALUES

If successful *AVW_AnisotropicDiffusionImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_AnisotropicAffineImage(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterImage(), AVW_MedianFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_Image

NAME

AVW_AnisotropicDiffusionImages – performs a 2D anisotropic diffusion transformation

SYNOPSIS

```
#include "AVW_Filter.h"

int AVW_AnisotropicDiffusionImages(in_images, num_images, iterations, kappa,
                                   bias_flag)
AVW_Image **in_image;
int num_images;
int iterations;
double kappa;
int bias_flag;
```

DESCRIPTION

AVW_AnisotropicDiffusionImages() performs 2D Anisotropic Diffusion on the a group of spatially correlated *in_images*, as specified in references (1) and (2) below, using the exponential diffusion function described therein. *num_images* is number of spatially correlated images.

Iterations specifies how many iterations to run the filter. *Kappa* specifies the gray level value difference above which the diffusion between two adjacent pixels becomes significantly suppressed (corresponding to the *kappa* in the equations in the references). *Bias_flag* specifies whether or not to perform biased anisotropic diffusion, as described in (2) and (3). A value of 1 specifies that biased anisotropic diffusion will be performed and 0 specifies that the biased option will not be performed.

REFERENCES

(1) Perona and Malik, "Scale-Space and Edge Detection Using Anisotropic Diffusion", IEEE PAMI 12, pp 629-639, 1990.

(2) Gerig, Kubler, Kikinis, and Jolesz, "Nonlinear Anisotropic Filtering of MRI Data", IEEE Trans Med Img 11, pp 221-232, 1992.

(3) Nordstrom, "Biased anisotropic diffusion - A unified regularization and diffusion approach to edge detection", Image Vision Comput., vol. 8, no. 4, pp 318-327, 1990.

RETURN VALUES

If successful *AVW_AnisotropicDiffusionImages()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AnnisotropicDiffusionImage(), AVW_Image

NAME

AVW_AnisotropicDiffusionVolumes – performs a 2D anisotropic diffusion transformation

SYNOPSIS

```
#include "AVW_Filter.h"

int AVW_AnisotropicDiffusionVolumes(in_volumes, num_volumes, iterations, kappa,
                    bias_flag)
AVW_Volume **in_volumes;
int num_volumes;
int iterations;
double kappa;
int bias_flag;
```

DESCRIPTION

AVW_AnisotropicDiffusionVolumes() performs 2D (slice by slice) Anisotropic Diffusion on the a group of spatially correlated *in_volumes*, as specified in references (1) and (2) below, using the exponential diffusion function described therein. *num_volumes* is number of spatially correlated volumes.

Iterations specifies how many iterations to run the filter. *Kappa* specifies the gray level value difference above which the diffusion between two adjacent pixels becomes significantly suppressed (corresponding to the *kappa* in the equations in the references). *Bias_flag* specifies whether or not to perform biased anisotropic diffusion, as described in (2) and (3). A value of 1 specifies that biased anisotropic diffusion will be performed and 0 specifies that the biased option will not be performed.

REFERENCES

(1) Perona and Malik, "Scale-Space and Edge Detection Using Anisotropic Diffusion", IEEE PAMI 12, pp 629-639, 1990.

(2) Gerig, Kubler, Kikinis, and Jolesz, "Nonlinear Anisotropic Filtering of MRI Data", IEEE Trans Med Img 11, pp 221-232, 1992.

(3) Nordstrom, "Biased anisotropic diffusion - A unified regularization and diffusion approach to edge detection", Image Vision Comput., vol. 8, no. 4, pp 318-327, 1990.

RETURN VALUES

If successful *AVW_AnisotropicDiffusionVolumes()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AnnisotropicDiffusionImages(), AVW_AnnisotropicDiffusionImage(), AVW_Image

NAME

AVW_AutoTrace – returns a mask for a connected thresholded region

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_AutoTrace(image, thresh_max, thresh_min, seed,
            type, exterior_only, gap_size, mask)
AVW_Image *image;
double thresh_max, thresh_min;
AVW_Point2 *seed;
int type, exterior_only, gap_size;
AVW_Image *mask;
```

DESCRIPTION

AVW_AutoTrace() finds and returns a mask in the *AVW_Image, mask.* The mask identifies the region defined by the *seed, thresh_max,* and *thresh_min*. The region is grown from the *seed* and includes all pixels which are connected via four neighbors to it and within the threshold values.

Type specifies the type of borders to return in the mask.

> *AVW_AUTO_OFF_EDGE* **-** sets the pixels which are just off the actual edge of the connected object.
>
> *AVW_AUTO_ON_EDGE* **-** indicates that the pixels exactly on the edge should be set.
>
> *AVW_AUTO_FILLED* **-** returns the entire connected object, and not just the edge pixels.

Exterior_only causes interior holes to be filled and not shown.

Gap_size specified the minimum allowable gap. A value of *zero (0)* specifies an unrestricted gap.

Mask is provided as a method of reusing an existing *AVW_Image.*

RETURN VALUES

If successful *AVW_AutoTrace()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_GetThresholdedBoundary(), AVW_GetBoundaryAndDelete(), AVW_GetMaskBoundary(), AVW_ThresholdImage(), AVW_Point2, AVW_Image

NAME

AVW_BestOpDataType – returns best output datatype for an operation

SYNOPSIS

```
#include       "AVW.h"

int AVW_BestOpDataType(dt1, max1, min1, op, dt2, max2, min2)
int dt1;
double max1, min1;
int op,
int dt2,
double max2, min2;
```

DESCRIPTION

AVW_BestOpDataType() returns the best output datatype.

NAME

AVW_BlendImage – merges two images

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_BlendImages(image1, factor, image2, blend_type, out_image)
AVW_Image *image1;
register double factor;
AVW_Image *image2;
int blend_type;
AVW_Image *out_image;
```

DESCRIPTION

AVW_BlendImages() returns a blended image.

Image1 and *image2* specify the two images to merge. They must have a data type of *AVW_UNSIGNED_CHAR* or *AVW_COLOR.*

Factor is a number from 0.0 to 1.0, specifying a *image1* to *image2* weighting factor. *Factor is only used for the*

Blend_type is one of the folling.

> *AVW_BLEND_AVERAGE* **-** use weighted average of th two input images. *AVW_BLEND_AVG_IGNORE_ZEROS* **-** zero in one image, causes 100% of the otehr image to be used *AVW_BLEND_CHECKBOARD* **-** everyother pixel is used from everyother image.
>
> *AVW_BLEND_DIFF* **-** returns a normaized difference *AVW_BLEND_REDGREEN* **-** Red channel is determined by image1, green channel from image2.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_BlendImages()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_BlendImages() will fail if:

BADMAL
: Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_Image()

NAME

AVW_BoundedStepSearchExtreme – Searches for best registration of two volume images.

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_Matrix *AVW_BoundedStepSearchExtreme(dirflag,base,match,points,steps,func,interpolation,m
int dirflag,interpolation;
AVW_Volume *base,*match;
AVW_FPointList3 *points;
AVW_Matrix *matrix;
AVW_StepSearchSpec *steps;
int func;
```

DESCRIPTION

AVW_BoundedStepSearchExtreme() Performs a bounded stepwise search of 6-DOF physical registration space to find the nearest extreme of a voxel statistic function relating two *AVW_Volumes. AVW_BoundedStepSearchExtreme* returns the *AVW_Matrix* which transforms the match volume into the space of the base volume at the extreme.

dirflag determines whether maxima or minima are searched for. Defined values are *AVW_MAXIMUM* and *AVW_MINIMUM.*

base and *match* are the *AVW_Volume* s to be registered.

points is an *AVW_FPointList3* containing a list of coordinate points in the match image to be used as the sample voxels for the registration. *points* is usually created by a call to *AVW_SetupVolumeSample*

steps contains the specification of the step search, primarily specific step sizes for each of the 6 degrees of freedom and the maximum number of steps which may be taken in searching for the extreme.

func specifies the statistical measure to be used. Defined values are *AVW_NMI.*

interpolation specifies the intepolation type to be used in the search, and may be any of the defined AVW interpolation types.

matrix is taken as the starting orientation for the search. If *Matrix* is NULL, a new identity matrix is created, and used as the starting orientation.

RETURN VALUES

If successful returns an *AVW_Matrix* which transforms the match volume into the space of the base volume at the extreme.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_BoundedStepSearchExtreme() will fail if the following is true:

BADMAL
: Could not allocate memory for internal structures.

ILLPAR
Sample or interpolation type is not recognized.

SEE ALSO *AVW_SetupVolumeSample() AVW_StepSearchExtreme() AVW_SampleSpec AVW_StepSearchSpec*

NAME

AVW_BoundedStepSearchExtreme2D – Searches for best registration of two volume images.

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_Matrix *AVW_BoundedStepSearchExtreme2D(dirflag,base,match,points,steps,func,interpolatio
int dirflag,interpolation;
AVW_Image *base,*match;
AVW_FPointList2 *points;
AVW_Matrix *matrix;
AVW_StepSearchSpec *steps;
int func;
```

DESCRIPTION

AVW_BoundedStepSearchExtreme2D() Performs a bounded stepwise search of 3-DOF physical registration space to find the nearest extreme of a voxel statistic function relating two *AVW_Images*. *AVW_BoundedStepSearchExtreme2D* returns the *AVW_Matrix* which transforms the match image into the space of the base image at the extreme.

dirflag determines whether maxima or minima are searched for. Defined values are *AVW_MAXIMUM* and *AVW_MINIMUM*.

base and *match* are the *AVW_Image* s to be registered.

points is an *AVW_FPointList2* containing a list of coordinate points in the match image to be used as the sample voxels for the registration. *points* is usually created by a call to *AVW_SetupImageSample*

steps contains the specification of the step search, primarily specific step sizes for each of the 3 degrees of freedom and the maximum number of steps which may be taken in searching for the extreme. Elements of *steps* relating to 6-DOF volume registration are ignored by *AVW_BoundedStepSearchExtreme2D*.

func specifies the statistical measure to be used. Defined values are *AVW_NMI*.

interpolation specifies the intepolation type to be used in the search, and may be any of the defined AVW interpolation types.

matrix is taken as the starting orientation for the search. If *Matrix* is NULL, a new identity matrix is created, and used as the starting orientation.

RETURN VALUES

If successful returns an *AVW_Matrix* which transforms the match image into the space of the base image at the extreme.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_BoundedStepSearchExtreme2D() will fail if the following is true:

BADMAL
Could not allocate memory for internal structures.

ILLPAR
Sample or interpolation type is not recognized.

SEE ALSO *AVW_SetupImageSample() AVW_StepSearchExtreme2D() AVW_SampleSpec AVW_StepSearchSpec*

NAME

AVW_CalculateObjectRegions – calculated minimum enclosing region for objects

SYNOPSIS

```
#include "AVW_ObjectMap.h"

int AVW_CalculateObjectRegions(object_map, flag)
AVW_ObjectMap *object_map;
int flag;
```

DESCRIPTION

AVW_CalculateObjectRegions() calculates the minimum enclosing regions for *AVW_Objects* within a *AVW_ObjectMap.*

If *flag* is set to *AVW_TRUE (1),* then the regions are calculated for all objects. A value of *AVW_FALSE (2),* indicates that only regions for enabled objects (DisplayFlag = AVW_TRUE) will be calculated.

RETURN VALUES

If successful *AVW_CalculateObjectRegions()* returns AVW_SUCCESS.. On failure it returns AVW_FAIL and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CalculateObjectRegions() will fail if the following is true:

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_CreateObjectMap(), AVW_GetObject(), AVW_PutObject() AVW_LoadObjectMap(), AVW_SaveObjectMap(), AVW_RenderVolume(), AVW_ObjectMap, AVW_Object

NAME

AVW_Malloc – allocates system memory

SYNOPSIS

```
#include "AVW.h"

void *AVW_Malloc(size)
unsigned int size;

void *AVW_Calloc(num, size)
unsigned int num;
unsigned int size;

void *AVW_Realloc(size, ptr)
unsigned int size;
void *ptr;

void AVW_Free(ptr)
void *ptr;
```

DESCRIPTION

These procedures provide a platform and compiler independent interface for memory allocation. Programs that need to transfer ownership of memory blocks between *AVW* and other modules should use these routines rather than the native *malloc()* and *free()* routines provided by the *C* run-time library.

AVW_Malloc returns a pointer to a size bytes suitably aligned for any use.

AVW_Calloc allocates space for an array nelem elements of *size* elsize. The space is initialized to zeros.

Tcl_Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the new block. The contents will be unchanged up to the lesser of the new and old sizes. The returned location may be different from *ptr.*

Tcl_Free makes the space referred to by *ptr* available for further allocation.

SEE ALSO

malloc(), free()

NAME

AVW_ChamferDistanceImage – calculates chamfer distances

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ChamferDistanceImage(in_image, out_image)
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ChamferDistanceImage() calculates the chamfer distance between all pixels in *in_image* to the nearest nonzero pixel within the image. The integer distances, which are scaled by a factor of 3, are returned in the pixels of *out_image.*

In_image must be of data type *AVW_UNSIGNED_CHAR. Out_image* may be of data type *AVW_UNSIGNED_CHAR* or *AVW_UNSIGNED_SHORT* depending on the maximum distance computed within the image.

The input image may be preprocessed using *AVW_ThresholdImage()* and / or *AVW_FindImageEdges().*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

REFERENCE

Borgefors, G. "Distance Transformations in Arbitrary Dimensions." Computer Vision, Graphics, and Image Processing, vol27, pp.321-345, 1984.

RETURN VALUES

If successful *AVW_ChamferDistanceImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ChamferDistanceImage() will fail if:

ILLDT
  Data type of input image must be AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_ThresholdImage(), AVW_FindImageEdges()

NAME

AVW_ChamferDistanceVolume – calculates chamfer distances

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ChamferDistanceVolume(in_image, out_image)
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ChamferDistanceVolume() calculates the chamfer distance between all voxels in *in_volume* to the nearest nonzero pixel within the volume. The integer distances, which are scaled by a factor of 3, are returned in the voxels of *out_volume.*

In_volume must be of data type *AVW_UNSIGNED_CHAR. Out_volume* may be of data type *AVW_UNSIGNED_CHAR* or *AVW_UNSIGNED_SHORT* depending on the maximum distance computed within the volume.

The input volume may be preprocessed using *AVW_ThresholdVolume()* and / or *AVW_FindVolumeEdges().*

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

REFERENCE

Borgefors, G. "Distance Transformations in Arbitrary Dimensions." Computer Vision, Graphics, and Volume Processing, vol27, pp.321-345, 1984.

RETURN VALUES

If successful *AVW_ChamferDistanceVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ChamferDistanceVolume() will fail if:

ILLDT
    Data type of input volume must be AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_ThresholdVolume(), AVW_FindVolumeEdges()

NAME

AVW_ChangeIsolatedPixels – changes the values of isolated pixels to that of their neighbors

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ChangeIsolatedPixels(in_image, changedPixels, out_image)
AVW_Image *in_image;
int *changedPixels;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ChangeIsolatedPixels() uses a mode filter to change the values of isolated pixels to that of their surrounding neighbors. This function can be used as a post processing step for the images returned from multi-spectral classification.

in_image must be of DataType *AVW_UNSIGNED_CHAR.*

changedPixels is a pointer to an integer which returns the number of pixels changed.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ChangeIsolatedPixels()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ChangeIsolatedPixels() will fail if one or more of the following are true:

ILLDT
Input image is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_ClassifyImage, AVW_Image

NAME

AVW_ClassifiedImageToCentroidFile – creates a multi-spectral sample file from a mask

SYNOPSIS

```
#include "AVW.h"

int AVW_ClassifiedImageToCentroidFile(imgs, numimgs, classImage, CentroidFile)
AVW_Image **imgs;
int numimgs;
AVW_Image *classImage;
char *CentroidFile;
```

DESCRIPTION

AVW_ClassifiedImageToCentroidFile() generates an AVW multi-spectral centroid file from a mask image and a list of corresponding spatially correlated images. This file can be used to perform image classification with *AVW_UnsuperClassifyImage()* and *AVW_UnsuperClassifyVolume().*

An arithmetic average is computed for all the classified pixels for each class and for each input image; producing a text file where each row contains the average pixel values for that class in each volume.

Imgs is a list of spatially correlated images.

Numimgs is the number of images in *Imgs*

ClassImage is an AVW_Image of DataType AVW_UNSIGNED_CHAR and the same size as the images in *Imgs* in which pixels to be used as training samples have non-zero values. This should be a classified image in which pixels with non-zero values are treated as belonging to a class.

CentroidFile is the name of the text file which is created by the function.

Sample File Contents

```
9.326923 0.717949
159.322922 131.093750
186.909088 159.636368
222.125000 223.620529
```

The first line of the file contains a signature identifying the contents of the file. The second line indicates how many classes are described in the file. The third row indicates the number of samples in a row.

RETURN VALUES

AVW_ClassifiedImageToCentroidFile() returns an *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifiedImageToCentroidFile() will fail if one or more of the following are true:

BADMAL
: Malloc Failed. A memory allocation failed.

ILLIMG
: Illegal Image. The images are not all the same dimension.

ILLDT
Illegal Datatype. At least one of the images is not AVW_UNSIGNED_CHAR.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO *AVW_ClassifiedVolumeToSampleFile(), AVW_UnsuperClassifyImage, AVW_UnsuperClassifyVolume, AVW_Image*

NAME

AVW_ClassifiedVolumeToCentroidFile – creates a multi-spectral centroid file

SYNOPSIS

```
#include "AVW.h"

int AVW_ClassifiedVolumeToCentroidFile(vols, numvols, classVolume, CentroidFile)
AVW_Volume **vols;
int numvols;
AVW_Volume *classVolume;
char *CentroidFile;
```

DESCRIPTION

AVW_ClassifiedVolumeToCentroidFile() generates an AVW multi-spectral centroid file from a classified volume and a list of corresponding spatially correlated volumes. This file can be used to perform image or volume classification with the Unsupervised Classification functions; *AVW_UnsuperClassifyImage()* and *AVW_UnsuperClassifyVolume()*.

An arithmetic average is computed for all the classified voxels for each class and for each input volume; producing a text file where each row contains the average voxel values for that class in each volume.

Vols is a list of spatially correlated volumes.

Numvols is the number of vols in *vols*

ClassVolume is an AVW_Volume of DataType AVW_UNSIGNED_CHAR and the same size as the volumes in *vols* in which voxels to be counted in the centroid calculations for each clas have non-zero values. This should be a classified volume in which voxels with non-zero values are treated as belonging to a class.

CentroidFile is the name of the text file which is created by the function.

Sample File Contents

```
9.326923 0.717949
159.322922 131.093750
186.909088 159.636368
222.125000 223.620529
```

RETURN VALUES

AVW_ClassifiedVolumeToCentroidFile() returns an *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifiedVolumeToCentroidFile() will fail if one or more of the following are true:

BADMAL
: Malloc Failed. A memory allocation failed.

ILLVOL
: Illegal Volume. The volumes are not all the same dimension.

ILLDT
: Illegal Datatype. At least one of the volumes is not AVW_UNSIGNED_CHAR.

INSPEC

Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO

AVW_ClassifiedImageToCentroidFile(), AVW_MaskImageToSampleFile(), AVW_UnsuperClassifyImage, AVW_UnsuperClassifyVolume, AVW_Volume

NAME

AVW_ClassifyFromScattergram – classifies pixels from a scattergram

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ClassifyFromScattergram(image1, image2, scattergram, out_image)
AVW_Image *image1;
AVW_Image *image2;
AVW_Image *scattergram;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ClassifyFromScattergram() uses a classified scattergram image, *scattergram*, and two input images, *image1* and *image2*, to very rapidly classify the pixels in the *out_image*. Classified pixels in the scattergram image map to pairs of values in the input images and specific pixels in the *out_image*. The pixel values of the *scattergram* image are taken to be class numbers. Pixels with a value of zero are taken to be unclassified.

Image1 and *image2* must be the same size and of data type *AVW_UNSIGNED_CHAR*. *Scattergram* must be created by using *AVW_Scattergram()*.

RETURN VALUES

If successful *AVW_ClassifyFromScattergram()* returns an *AVW_Image*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyFromScattergram() will fail if one or more of the following are true:

ICPIMG
Incompatible input images. Images are not the same size or not AVW_UNSIGNED_CHAR data type.

BDSCGR
Bad scattergram.

SEE ALSO

AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_Scattergram(), AVW_Image

NAME

AVW_ClassifyImage – classifies pixels from multi-spectral data sets

SYNOPSIS

```
#include     "AVW.h"

AVW_Image *AVW_ClassifyImage(imgs, numimgs, train_img,
                autotype, maxdist, sigma,
                kvalue,  epochs,  hiddenepochs,
                out_image)

AVW_Image **imgs;
int numimgs;
AVW_Image *train_img;
int autotype;
double maxdist;
double sigma;
double kvalue;
int epochs;
int hiddenepochs;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ClassifyImage() classifies pixels given multi-spectral data consisting of several input images under the supervision of *train_img* training samples in which some pixels are marked as belonging to various classes.

Several different algorithms can be used to classify unclassified pixels from the input mask image.

***Imgs* is a list of spatially correlated images, each of which represents one spectra of the multi-spectral data set. These input images must be of the same dimension and of data type *AVW_UNSIGNED_CHAR*. This input data is unchanged by the classification process.

Numimgs is the number of images in the input image list *imgs*.

Train_img is a set of training samples for the various automatic classification algorithms. It must be of the same dimensions as the images from the input image list *imgs*. Its data type must be *AVW_UNSIGNED_CHAR*. The values of pixels in *train_img* are taken to be known class numbers of pixels in the input images. Pixels with the value of 0 are taken as unknown and are targets for the classification process. Pixels with the value of 1 are taken as belonging to class number 1, pixels with value of 2 are taken belonging to class number 2, etc. *Train_img* must be prepared prior to invoking *AVW_ClassifyImage()*.

Autotype specificies the automatic classification algorithm used to classify the image. Acceptable values are *AVW_GAUSSIAN_CLUSTER*, *AVW_NEURAL_NETWORK*, *AVW_NEAREST_NEIGHBOR*, *AVW_K_NEAREST_NEIGHBOR*, and *AVW_PARZEN_WINDOWS* which are defined in *AVW.h*.

The *AVW_GAUSSIAN_CLUSTER* technique calculates means and standard deviations for each class along each feature, and then models the class' probabilty distribution as a Gaussian function with the specified parameters. An unknown pixel is then classified by calculating the probability that it belongs to each class and choosing the best result. A pixel is left unclassified if it lies farther than a certain number of standard deviations from all the class centers (controlled by the *maxdist* parameter).

The *AVW_NEURAL_NETWORK* classifier builds a standard 3-layer feed forward neural network, of the kind commonly trained by backpropagation, and trains it on the class samples with a more advanced technique known as conjugate gradient optimization [1].

The *AVW_NEAREST_NEIGHBOR* technique simply takes each unclassified pixel, looks for its closest neighbor in feature space among the class samples, and assigns it to that same class. If two samples from different classes are equally close, the pixel is left unclassified

AVW_K_NEAREST_NEIGHBOR extends the previous method to the *kvalue* nearest neighbors among the class samples, giving each of these an equal vote. In both cases, pixels are left unclassified whose best match samples are farther than the distance given by the *maxdist* parameter.

The *AVW_PARZEN_WINDOWS* technique is similar to *AVW_GAUSSIAN_CLUSTER* but makes a more sophisticated estimate of each class' underlying probability distribution function from the samples. It uses these estimates to draw near-optimal decision boundaries, but is much slower than *AVW_GAUSSIAN_CLUSTER* classification. This latter technique is implemented in the manner described in [2].

In each of the classification types, a pixel is assigned to the most likely class (as estimated by the classifier), and this is done for each pixel independently. This can result in a "noisy" classification if the data is noisy or if classes actually overlap in feature space.

Maxdist specifies the maximum distance in feature space that an unknown pixel can be from a sample of a known class to be considered as a possible member of that class. When *autotype* is *AVW_GAUSSIAN_CLUSTERING* it is in units of standard deviations. This parameter is meaningful and used only when *autotype* is *AVW_GAUSSIAN_CLUSTERING, AVW_NEAREST_NEIGHBOR*, or *AVW_K_NEAREST_NEIGHBOR.*

Sigma is only used for the *AVW_PARZEN_WINDOW* technique, and is ignored for all others. It specifies the standard deviation of the Gaussian window which is used to smooth the input samples to estimate the probability distribution. A default value of 5.0 is suggested.

Kvalue specifies the number of nearest neighbors in feature space to be considered when using *AVW_K_NEAREST_NEIGHBOR* classification. This parameter is ignored for all other values of *autotype.*

Epochs specifies the maximum number of passes through the class samples while training the neural network. This parameter is used only when using *AVW_NEURAL_NETWORK,* it is ignored for all other values of *autotype.*

Hiddenepochs specifies the number of hidden units in the (single hidden layer) neural network. This parameter is used only when using *AVW_NEURAL_NETWORK,* it is ignored for all other values of *autotype.*

Out_image is the returned classified image. Pixels which have been successfully classified are set to values of pixels from the training sample mask image which they are most similar to. Pixels with a value of 0 were not classified by the function.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ClassifyImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyImage will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLIMG
Illegal Image. The images are not all the same dimension.

ILLDT
Illegal Datatype. At least one of the images is not AVW_UNSIGNED_CHAR.

BDTRSM
Bad Training Sample. The supplied training samples were unusable.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO

AVW_ClassifyVolume() AVW_ClassifyFromScattergram(), AVW_Scattergram(), AVW_Image

REFERENCES

[1] A. Kramer and A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks", in Advanced in Neural Information Processing Systems 1, pp 40-48, Morgan-Kauffman, San Mateo, CA 1989

[2] D. F. Specht, "Probabilistic Neural Networks, Neural Networks, Vol 3, pp 109-118, 1990.

[3] M. Morrison and Y. Attikiouzel, "A Probabilistic Network Based Image Segmentation Network for Magnetic Resonance Images", Proc. Intl. Joint Conference on Neural Networks, Baltimore, MD, June 7-11, 1992, pp III-60 - III-65, IEEE Press, 1992

NAME

AVW_ClassifyImageFromSampleFile – classifies pixels from multi-spectral data sets

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ClassifyImageFromSampleFile(imgs, numimgs, SampleFile,
                autotype, maxdist, sigma,
                kvalue, epochs, hiddenepochs,
                out_image)

AVW_Image **imgs;
int numimgs;
char *SampleFile;
int autotype;
double maxdist;
double sigma;
double kvalue;
int epochs;
int hiddenepochs;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ClassifyImageFromSampleFile() classifies pixels given multi-spectral data consisting of several input images under the supervision of *SampleFile,* a text file which contains a list of class numbers and corresponding multi-spectral values.

Several different algorithms can be used to classify unclassified pixels from the input mask image.

***Imgs* is a list of spatially correlated images, each of which represents one spectra of the multi-spectral data set. These input images must be of the same dimension. This input data is unchanged by the classification process.

Numimgs is the number of images in the input image list *imgs.*

SampleFile is a set of training samples for the various automatic classification algorithms. This is a text file created by *AVW_MaskVolumeToSampleFile()* or *AVW_MaskImageToSampleFile().*

Autotype specificies the automatic classification algorithm used to classify the image. Acceptable values are *AVW_GAUSSIAN_CLUSTER, AVW_NEURAL_NETWORK, AVW_NEAREST_NEIGHBOR, AVW_K_NEAREST_NEIGHBOR, AVW_PARZEN_WINDOWS AVW_NEAREST_MEAN* and *AVW_NEAREST_MEDIAN* which are defined in *AVW.h.*

The *AVW_GAUSSIAN_CLUSTER* technique calculates means and standard deviations for each class along each feature, and then models the class’ probabilty distribution as a Gaussian function with the specified parameters. An unknown pixel is then classified by calculating the probability that it belongs to each class and choosing the best result. A pixel is left unclassified if it lies farther than a certain number of standard deviations from all the class centers (controlled by the *maxdist* parameter).

The *AVW_NEURAL_NETWORK* classifier builds a standard 3-layer feed forward neural network, of the kind commonly trained by backpropagation, and trains it on the class samples with a more advanced technique known as conjugate gradient optimization [1].

The *AVW_NEAREST_NEIGHBOR* technique simply takes each unclassified pixel, looks for its closest neighbor in feature space among the class samples, and assigns it to that same class. If two samples from different classes are equally close, the pixel is left unclassified

AVW_K_NEAREST_NEIGHBOR extends the previous method to the *kvalue* nearest neighbors among the class samples, giving each of these an equal vote. In both cases, pixels are left unclassified whose best match samples are farther than the distance given by the *maxdist* parameter.

The *AVW_PARZEN_WINDOWS* technique is similar to *AVW_GAUSSIAN_CLUSTER* but makes a more sophisticated estimate of each class' underlying probability distribution function from the samples. It uses these estimates to draw near-optimal decision boundaries, but is much slower than *AVW_GAUSSIAN_CLUSTER* classification. This latter technique is implemented in the manner described in [2].

The *AVW_NEAREST_MEAN* technique calculates the mean vector for each training class. Each voxel to be classified is assigned to the class with the closest mean vector within the value specified by *Maxdist.*

The *AVW_NEAREST_MEDIAN* technique works in the same way as the *AVW_NEAREST_MEAN* except that the median vector for each class is used.

In each of the classification types, a pixel is assigned to the most likely class (as estimated by the classifier), and this is done for each pixel independently. This can result in a "noisy" classification if the data is noisy or if classes actually overlap in feature space.

Maxdist specifies the maximum distance in feature space that an unknown pixel can be from a sample of a known class to be considered as a possible member of that class. When *autotype* is *AVW_GAUSSIAN_CLUSTERING* it is in units of standard deviations. This parameter is meaningful and used only when *autotype* is *AVW_GAUSSIAN_CLUSTERING, AVW_NEAREST_NEIGHBOR, AVW_K_NEAREST_NEIGHBOR, AVW_NEAREST_MEAN* or *AVW_NEAREST_MEDIAN* are used.

Sigma is only used for the *AVW_PARZEN_WINDOW* technique, and is ignored for all others. It specifies the standard deviation of the Gaussian window which is used to smooth the input samples to estimate the probability distribution. A default value of 5.0 is suggested.

Kvalue specifies the number of nearest neighbors in feature space to be considered when using *AVW_K_NEAREST_NEIGHBOR* classification. This parameter is ignored for all other values of *autotype.*

Epochs specifies the maximum number of passes through the class samples while training the neural network. This parameter is used only when using *AVW_NEURAL_NETWORK,* it is ignored for all other values of *autotype.*

Hiddenepochs specifies the number of hidden units in the (single hidden layer) neural network. This parameter is used only when using *AVW_NEURAL_NETWORK,* it is ignored for all other values of *autotype.*

Out_image is the returned classified image. Pixels which have been successfully classified are set to values of pixels from the training sample mask image which they are most similar to. Pixels with a value of 0 were not classified by the function.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ClassifyImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyImage will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLIMG
Illegal Image. The images are not all the same dimension.

BDTRSM
Bad Training Sample. The supplied training samples were unusable.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO

AVW_ClassifyVolumeFromSampleFile(), AVW_MaskVolumeToSampleFile(), AVW_MaskImageToSampleFile(), AVW_ClassifyVolume(), AVW_ClassifyImage(), AVW_Image

REFERENCES

[1] A. Kramer and A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks", in Advanced in Neural Information Processing Systems 1, pp 40-48, Morgan-Kauffman, San Mateo, CA 1989

[2] D. F. Specht, "Probabilistic Neural Networks, Neural Networks, Vol 3, pp 109-118, 1990.

NAME

AVW_ClassifyScattergram – classifies a scattergram

SYNOPSIS

```
#include       "AVW.h"

AVW_Image *AVW_ClassifyScattergram(mask, autotype,
                        maxdist, sigma, kvalue,
                        epochs,  hiddenunits,
                        out_image)
AVW_Image *mask;
int autotype;
double maxdist;
double sigma;
double kvalue;
int epochs;
int hiddenunits;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ClassifyScattergram() classifies additional pixels in a scattergram produced by *AVW_Scattergram()*. Several different algorithms can be used to classify unclassified pixels from the input *mask.*

Mask is a scattergram produced with training samples with *AVW_Scattergram()*. Its data type must be *AVW_UNSIGNED_CHAR*. The values of pixels in *mask* are taken to be known class numbers of pixels in the input images. Pixels with the value of 0 are taken as unknown and are targets for the classification process. Pixels with the value of 1 are taken as belonging to class number 1, pixels with value of 2 are taken belonging to class number 2, etc. *Mask* must be prepared prior to invoking *AVW_ClassifyScattergram().*

Autotype specificies the automatic classification algorithm used to classify the image. Acceptable values are *AVW_GAUSSIAN_CLUSTER*, *AVW_NEURAL_NETWORK*, *AVW_NEAREST_NEIGHBOR*, *AVW_K_NEAREST_NEIGHBOR*, and *AVW_PARZEN_WINDOWS* which are defined in *AVW.h.*

The *AVW_GAUSSIAN_CLUSTER* technique calculates means and standard deviations for each class along each feature, and then models the class' probabilty distribution as a Gaussian function with the specified parameters. An unknown pixel is then classified by calculating the probability that it belongs to each class and choosing the best result. A pixel is left unclassified if it lies farther than a certain number of standard deviations from all the class centers (controlled by the *maxdist* parameter).

The *AVW_NEURAL_NETWORK* classifier builds a standard 3-layer feed forward neural network, of the kind commonly trained by backpropagation, and trains it on the class samples with a more advanced technique known as conjugate gradient optimization [1].

The *AVW_NEAREST_NEIGHBOR* technique simply takes each unclassified pixel, looks for its closest neighbor in feature space among the class samples, and assigns it to that same class. If two samples from different classes are equally close, the pixel is left unclassified

AVW_K_NEAREST_NEIGHBOR extends the previous method to the *kvalue* nearest neighbors among the class samples, giving each of these an equal vote. In both cases, pixels are left unclassified whose best match samples are farther than the distance given by the *maxdist* parameter.

The *AVW_PARZEN_WINDOWS* technique is similar to *AVW_GAUSSIAN_CLUSTER* but makes a more sophisticated estimate of each class' underlying probability distribution function from the samples. It uses these estimates to draw near-optimal decision boundaries, but is much slower than *AVW_GAUSSIAN_CLUSTER* classification. This latter technique is implemented in the matter described in [2].

In each of the classification types, a pixel is assigned to the most likely class (as estimated by the classifier), and this is done for each pixel independently. This can result in a "noisy" classification if the data is noisy or if classes actually overlap in feature space.

Maxdist specifies the maximum distance in feature space that an unknown pixel can be from a sample of a known class to be considered as a possible member of that class. When *autotype* is *AVW_GAUSSIAN_CLUSTERING* it is in units of standard deviations. This parameter is meaningful and used only when *autotype* is *AVW_GAUSSIAN_CLUSTERING*, *AVW_NEAREST_NEIGHBOR*, or *AVW_K_NEAREST_NEIGHBOR*.

Sigma is only used for the *AVW_PARZEN_WINDOW* technique, and is ignored for all others. It specifies the standard deviation of the Gaussian window which is used to smooth the input samples to estimate the probability distribution. A default value of 5.0 is suggested.

Kvalue specifies the number of nearest neighbors in feature space to be considered when using *AVW_K_NEAREST_NEIGHBOR* classification. This parameter is ignored for all other values of *autotype.*

Epochs specifies the maximum number of passes through the class samples while training the neural network. This parameter is used only when using *AVW_NEURAL_NETWORK*, it is ignored for all other values of *autotype.*

Hiddenepochs specifies the number of hidden units in the (single hidden layer) neural network. This parameter is used only when using *AVW_NEURAL_NETWORK,* it is ignored for all other values of *autotype.*

Out_image is the returned classified image. Pixels which have been successfully classified are set to values of pixels from the training sample mask image which they are most similar to. Pixels with a value of 0 were not classified by the function.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ClassifyScattergram()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyScattergram will fail if one or more of the following are true:

BADMAL
: Malloc Failed. A memory allocation failed.

ILLIMG
: Illegal Image. The images are not all the same dimension.

ILLDT
: Illegal Datatype. At least one of the images is not AVW_UNSIGNED_CHAR.

BDTRSM
Bad Training Sample. The supplied training samples were unusable.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO

AVW_ClassifyFromScattergram(), AVW_Scattergram(), AVW_Image

REFERENCES

[1] A. Kramer and A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks", in Advanced in Neural Information Processing Systems 1, pp 40-48, Morgan-Kauffman, San Mateo, CA 1989

[2] D. F. Specht, "Probabilistic Neural Networks, Neural Networks, Vol 3, pp 109-118, 1990.

[3] M. Morrison and Y. Attikiouzel, "A Probabilistic Network Based Image Segmentation Network for Magnetic Resonance Images", Proc. Intl. Joint Conference on Neural Networks, Baltimore, MD, June 7-11, 1992, pp III-60 - III-65, IEEE Press, 1992

NAME

AVW_ClassifyVolume – classifies voxels from multi-spectral data sets

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ClassifyVolume(vols, numvols, train_vol,
                autotype, maxdist, sigma,
                kvalue,  epochs,  hiddenepochs,
                out_volume)

AVW_Volume **vols;
int numvols;
AVW_Volume *train_vol;
int autotype;
double maxdist;
double sigma;
double kvalue;
int epochs;
int hiddenepochs;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ClassifyVolume() classifies voxels given multi-spectral data consisting of several input volumes under the supervision of *train_volume* training samples in which some voxels are marked as belonging to various classes.

Several different algorithms can be used to classify unclassified voxels from the input mask volume.

***Vols* is a list of spatially correlated volumes, each of which represents one spectra of the multi-spectral data set. These input volumes must be of the same dimension and of data type *AVW_UNSIGNED_CHAR*. This input data is unchanged by the classification process

Numvols is the number of volumes in the input volume list *vols.*

Train_vol is a set of training samples for the various automatic classification algorithms. It must be of the same dimensions as the volumes from the input volume list *vols.* Its data type must be *AVW_UNSIGNED_CHAR*. The values of voxels in *train_vol* are taken to be known class numbers of voxels in the input volumes. Voxels with the value of 0 are taken as unknown and are targets for the classification process. Voxels with the value of 1 are taken as belonging to class number 1, voxels with value of 2 are taken belonging to class number 2, etc. *Train_vol* must be prepared prior to invoking *AVW_ClassifyVolume().*

Autotype specificies the automatic classification algorithm used to classify the volume. Acceptable values are *AVW_GAUSSIAN_CLUSTER*, *AVW_NEURAL_NETWORK*, *AVW_NEAREST_NEIGHBOR*, *AVW_K_NEAREST_NEIGHBOR*, and *AVW_PARZEN_WINDOWS*.

The *AVW_GAUSSIAN_CLUSTER* technique calculates means and standard deviations for each class along each feature, and then models the class' probabilty distribution as a Gaussian function with the specified parameters. An unknown voxel is then classified by calculating the probability that it belongs to each class and choosing the best result. A voxel is left unclassified if it lies farther than a certain number of standard deviations from all the class centers (controlled by the *maxdist* parameter).

The *AVW_NEURAL_NETWORK* classifier builds a standard 3-layer feed forward neural network, of the kind commonly trained by backpropagation, and trains it on the class samples with a more advanced technique known as conjugate gradient optimization [1].

The *AVW_NEAREST_NEIGHBOR* technique simply takes each unclassified voxel, looks for its closest neighbor in feature space among the class samples, and assigns it to that same class. If two samples from different classes are equally close, the voxel is left unclassified

AVW_K_NEAREST_NEIGHBOR extends the previous method to the *kvalue* nearest neighbors among the class samples, giving each of these an equal vote. In both cases, voxels are left unclassified whose best match samples are farther than the distance given by the *maxdist* parameter.

The *AVW_PARZEN_WINDOWS* technique is similar to *AVW_GAUSSIAN_CLUSTER* but makes a more sophisticated estimate of each class' underlying probability distribution function from the samples. It uses these estimates to draw near-optimal decision boundaries, but is much slower than *AVW_GAUSSIAN_CLUSTER* classification. This latter technique is implemented in the matter described in [2].

In each of the classification types, a voxel is assigned to the most likely class (as estimated by the classifier), and this is done for each voxel independently. This can result in a "noisy" classification if the data is noisy or if classes actually overlap in feature space.

Maxdist specifies the maximum distance in feature space that an unknown voxel can be from a sample of a known class to be considered as a possible member of that class. When *autotype* is *AVW_GAUSSIAN_CLUSTER* it is in units of standard deviations. This parameter is meaningful and used only when *autotype* is *AVW_GAUSSIAN_CLUSTER*, *AVW_NEAREST_NEIGHBOR*, or *AVW_K_NEAREST_NEIGHBOR*.

Sigma is only used for the *AVW_PARZEN_WINDOW* technique, and is ignored for all others. It specifies the standard deviation of the Gaussian window which is used to smooth the input samples to estimate the probability distribution. A default value of 5.0 is suggested.

Kvalue specifies the number of nearest neighbors in feature space to be considered when using *AVW_K_NEAREST_NEIGHBOR* classification. This parameter is ignored for all other values of *autotype.*

Epochs specifies the maximum number of passes through the class samples while training the neural network. This parameter is used only when using *AVW_NEURAL_NETWORK*, it is ignored for all other values of *autotype.*

Hiddenepochs specifies the number of hidden units in the (single hidden layer) neural network. This parameter is used only when using *AVW_NEURAL_NETWORK,* it is ignored for all other values of *autotype.*

Out_volume is the returned classified volume. Voxels which have been successfully classified are set to values of voxels from the training sample mask volume which they are most similar to. Voxels with a value of 0 were not classified by the function.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ClassifyVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyVolume will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLVOL
Illegal Volume. The volumes are not all the same dimension.

ILLDT
Illegal Datatype. At least one of the volumes is not AVW_UNSIGNED_CHAR.

BDTRSM
Bad Training Sample. The supplied training samples were unusable.

INSPEC
Insufficient Specifications. Fewer than two input volumes were supplied.

SEE ALSO

AVW_ClassifyImage(), AVW_ClassifyFromScattergram(), AVW_Scattergram(), AVW_Volume

REFERENCES

[1] A. Kramer and A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks", in Advanced in Neural Information Processing Systems 1, pp 40-48, Morgan-Kauffman, San Mateo, CA 1989

[2] D. F. Specht, "Probabilistic Neural Networks, Neural Networks, Vol 3, pp 109-118, 1990.

[3] M. Morrison and Y. Attikiouzel, "A Probabilistic Network Based Image Segmentation Network for Magnetic Resonance Images", Proc. Intl. Joint Conference on Neural Networks, Baltimore, MD, June 7-11, 1992, pp III-60 - III-65, IEEE Press, 1992

NAME

AVW_ClassifyVolumeFromSampleFile – classifies voxels from multi-spectral data sets

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ClassifyVolumeFromSampleFile(vols, numvols, SampleFile,
                autotype, maxdist, sigma,
                kvalue, epochs, hiddenepochs,
                out_vol)

AVW_Volume **vols;
int numvols;
char *SampleFile;
int autotype;
double maxdist;
double sigma;
double kvalue;
int epochs;
int hiddenepochs;
AVW_Image *out_vol;
```

DESCRIPTION

AVW_ClassifyVolumeFromSampleFile() classifies voxels given multi-spectral data consisting of several input volumes under the supervision of *SampleFile* which is a text file which contains a list of class numbers and corresponding multi-spectral values. Each training sample is a row in the file which consists of the class number and then tab separated values which correspond to the values for volume1, volume2, ...

Several different algorithms can be used to classify unclassified pixels from the input mask image.

***Vols* is a list of spatially correlated volumes, each of which represents one spectra of the multi-spectral data set. These input images must be of the same dimension. This input data is unchanged by the classification process.

Numvols is the number of volumess in the input image list *imgs.*

SampleFile is a set of training samples for the various automatic classification algorithms. This is a text file created by *AVW_MaskVolumeToSampleFile()* or *AVW_MaskImageToSampleFile().*

Autotype specificies the automatic classification algorithm used to classify the image. Acceptable values are *AVW_GAUSSIAN_CLUSTER, AVW_NEURAL_NETWORK, AVW_NEAREST_NEIGHBOR, AVW_K_NEAREST_NEIGHBOR, AVW_PARZEN_WINDOWS, AVW_NEAREST_MEAN,* and *AVW_NEAREST_MEDIAN* which are defined in *AVW.h.*

The *AVW_GAUSSIAN_CLUSTER* technique calculates means and standard deviations for each class along each feature, and then models the class' probabilty distribution as a Gaussian function with the specified parameters. An unknown pixel is then classified by calculating the probability that it belongs to each class and choosing the best result. A pixel is left unclassified if it lies farther than a certain number of standard deviations from all the class centers (controlled by the *maxdist* parameter).

The *AVW_NEURAL_NETWORK* classifier builds a standard 3-layer feed forward neural network, of the kind commonly trained by backpropagation, and trains it on the class samples with a more advanced technique known as conjugate gradient optimization [1].

The *AVW_NEAREST_NEIGHBOR* technique simply takes each unclassified pixel, looks for its closest neighbor in feature space among the class samples, and assigns it to that same class. If two samples from different classes are equally close, the pixel is left unclassified

AVW_K_NEAREST_NEIGHBOR extends the previous method to the *kvalue* nearest neighbors among the class samples, giving each of these an equal vote. In both cases, pixels are left unclassified whose best match samples are farther than the distance given by the *maxdist* parameter.

The *AVW_PARZEN_WINDOWS* technique is similar to *AVW_GAUSSIAN_CLUSTER* but makes a more sophisticated estimate of each class' underlying probability distribution function from the samples. It uses these estimates to draw near-optimal decision boundaries, but is much slower than *AVW_GAUSSIAN_CLUSTER* classification. This latter technique is implemented in the matter described in [2].

The *AVW_NEAREST_MEAN* technique calculates the mean vector for each training class. Each voxel to be classified is assigned to the class with the closest mean vector within the value specified by *Maxdist.*

The *AVW_NEAREST_MEDIAN* technique works in the same way as the *AVW_NEAREST_MEAN* except that the median vector for each class is used.

In each of the classification types, a pixel is assigned to the most likely class (as estimated by the classifier), and this is done for each pixel independently. This can result in a "noisy" classification if the data is noisy or if classes actually overlap in feature space.

Maxdist specifies the maximum distance in feature space that an unknown pixel can be from a sample of a known class to be considered as a possible member of that class. When *autotype* is *AVW_GAUSSIAN_CLUSTERING* it is in units of standard deviations. This parameter is meaningful and used only when *autotype* is *AVW_GAUSSIAN_CLUSTERING, AVW_NEAREST_NEIGHBOR, AVW_K_NEAREST_NEIGHBOR, AVW_NEAREST_MEDIAN,* and AVW_NEAREST_MEAN.

Sigma is only used for the *AVW_PARZEN_WINDOW* technique, and is ignored for all others. It specifies the standard deviation of the Gaussian window which is used to smooth the input samples to estimate the probability distribution. A default value of 5.0 is suggested.

Kvalue specifies the number of nearest neighbors in feature space to be considered when using *AVW_K_NEAREST_NEIGHBOR* classification. This parameter is ignored for all other values of *autotype.*

Epochs specifies the maximum number of passes through the class samples while training the neural network. This parameter is used only when using *AVW_NEURAL_NETWORK*, it is ignored for all other values of *autotype.*

Hiddenepochs specifies the number of hidden units in the (single hidden layer) neural network. This parameter is used only when using *AVW_NEURAL_NETWORK,* it is ignored for all other values of *autotype.*

Out_image is the returned classified image. Pixels which have been successfully classified are set to values of pixels from the training sample mask image which they are most similar to. Pixels with a value of 0 were not classified by the function.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ClassifyImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyImage will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLIMG
Illegal Image. The images are not all the same dimension.

BDTRSM
Bad Training Sample. The supplied Sample File was unusable.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO

AVW_ClassifyImageFromSampleFile(), AVW_MaskVolumeToSampleFile(), AVW_MaskImageToSampleFile(), AVW_ClassifyVolume(), AVW_ClassifyImage(), AVW_Image

REFERENCES

[1] A. Kramer and A. Sangiovanni-Vincentelli, "Efficient Parallel Learning Algorithms for Neural Networks", in Advanced in Neural Information Processing Systems 1, pp 40-48, Morgan-Kauffman, San Mateo, CA 1989

[2] D. F. Specht, "Probabilistic Neural Networks, Neural Networks, Vol 3, pp 109-118, 1990.

[3] M. Morrison and Y. Attikiouzel, "A Probabilistic Network Based Image Segmentation Network for Magnetic Resonance Images", Proc. Intl. Joint Conference on Neural Networks, Baltimore, MD, June 7-11, 1992, pp III-60 - III-65, IEEE Press, 1992

NAME

AVW_ClearHistogram – zeros out a histogram

SYNOPSIS

```
#include "AVW_Histogram.h"

int AVW_ClearHistogram(histo)
AVW_Histogram *histo;
```

DESCRIPTION

AVW_ClearHistogram() assigns zeros to each bin of the *AVW_Histogram histo.*

RETURN VALUES

If successful *AVW_ClearHistogram()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClearHistogram() fails if the following is true:

ILLHIS
Illegal Histogram. Nonvalid histogram was passed to this function.

SEE ALSO

AVW_CreateHistogram(), AVW_DestroyHistogram(), AVW_FlattenImageHistogram(), AVW_FlattenVolumeHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_MatchImageHistogram(), AVW_MatchVolumeHistogram(), AVW_NormalizeHistogram(), AVW_PreserveImageHistogram(), AVW_PreserveVolumeHistogram(), AVW_VerifyHistogram(), AVW_Histogram

NAME

AVW_ClipPointList2 – clips a point list at image bounds

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_ClipPointList2(list, lowx, lowy, highx, highy, out_list)
AVW_PointList2 *list;
int lowx, lowy;
int highx, highy;
AVW_PointList2 *out_list;
```

DESCRIPTION

AVW_ClipPointList2() clips the the x and y coordinates of the points in *list* so that all coodinates meet the following criteria:

lowx <= x <= highx
lowy <= y <= highy

Out_list is provided as a method of reusing an existing *AVW_PointList2.* If not reuseable *out_list* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ClipPointList2()* returns an *AVW_PointList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClipPointList2() will fail if:

BADMAL
Could not allocate enough memory for results.

SEE ALSO

AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_DrawPointList2(), AVW_ShiftPointList2(), AVW_TransformPoint2(), AVW_PointList2

NAME

AVW_CloseImageFile – closes an image file

SYNOPSIS

```
#include "AVW_ImageFile.h"

int AVW_CloseImageFile (imgfile)
AVW_ImageFile *imgfile;
```

DESCRIPTION

AVW_CloseImageFile() closes an *AVW_ImageFile*. All open files associated with the *AVW_ImageFile* are closed. All structures and memory buffers associated with the *AVW_ImageFile* are freed.

RETURN VALUES

If successful *AVW_CloseImageFile()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of failure.

ERRORS

AVW_CloseImageFile() will fail for the following reason:

BDCLS
Bad Close.

SEE ALSO

AVW_CreateImageFile(), AVW_ExtendImageFile(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume(), AVW_ImageFile

NAME

AVW_ClosestInPointList2 – finds the closest point in an AVW_PointList2

SYNOPSIS

```
#include "AVW.h"

int AVW_ClosestInPointList2(point2, plist)
AVW_Point2 *point2;
AVW_PointList2 *plist;
```

DESCRIPTION

AVW_ClosestInPointList2() searches *plist* for the closest point to *point2*. When found the index of that point is returned.

RETURN VALUES

If successful, *AVW_ClosestInPointList2()* returns the index of the closest point. On failure, it returns *-1* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClosestInPointList2() will fail if one or more of the following are true:

PTNFND
Pointlist didn't contain any points.

SEE ALSO

AVW_ClosestInImage(), AVW_Point2, AVW_Point2List2

NAME

AVW_ClosestPointInImage – finds the closest nonzero point in an image

SYNOPSIS

#include "AVW.h"

int AVW_ClosestPointInImage(image, point, value)
AVW_Image *image;
AVW_Point2 *point;
double value

DESCRIPTION

AVW_ClosestPointInImage() resets the the coordinates of *point* to the closest pixel with an intensity of *value* in *image* to *point*.

RETURN VALUES

If successful, *AVW_ClosestPointInImage()* returns *AVW_SUCCESS*. On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClosestPointInImage() will fail if one or more of the following are true:

ILLPAR
Point specified is outside of the image.

ILLDT
Illegal datatype.

EMPTYI
All pixels inthe image are zero.

PTNFND
Closest point not found.

SEE ALSO

AVW_ClosestInPointList2(), AVW_Image, AVW_Point2

NAME

AVW_Compute2DShapeStats – performs shape analysis

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_Compute2DShapeStats(mask_image, mask_value, stats)
AVW_Image *mask_image;
int mask_value:
AVW_2DShapeStats *stats;
```

DESCRIPTION

AVW_Compute2DShapeStats() performs shape analysis on *mask_image.* All pixels of value *mask_value* are taken as belonging to the object to be analyzed. *Mask_image* is an *AVW_Image* of data type *AVW_UNSIGNED_CHAR*. The shape statistics are returned in *stats*, which is an *AVW_2DShapeStats* structure, defined in *AVW_Measure.h.*

```
typedef struct
  {
  float  Area,                  /* pixel count of mask        */
         Perimeter,       /* distance around the shape   */
         MERAngle               /* in degrees                 */
         MERArea,               /* of minimum enclosing rectangle    */
         MERAspect,             /* aspect ratio of the
                                   minimum enclosing rectangle */
         RFF,                   /* rectangle fit factor       */
         Circularity;           /* ratio of region perimeter
                                   squared to its area        */
         AVW_FPoint2 Centroid;  /* x & y coords of centroid   */
         AVW_FPoint2 MER1;
         AVW_FPoint2 MER2;
         AVW_FPoint2 MER3;
         AVW_FPoint2 MER4;
  } AVW_2DShapeStats;
```

RETURN VALUES

If successful *AVW_Compute2DShapeStats()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_Compute2DShapeStats() will fail for the following reasons:

ILLMSK
Illegal Mask.

ILLPAR
Illegal Parameter.

SEE ALSO

AVW_ComputeCircularity(), AVW_ComputeImageCentroid(), AVW_ComputeImageFractalSig(), AVW_ComputeImageIntensityStats(), AVW_ComputeMER(), AVW_ComputePerimeter(), AVW_ComputeRFF(), AVW_2DShapeStats

NAME

AVW_ComputeCircularity – calculates the circularity of a 2D region

SYNOPSIS

```
#include "AVW_Measure.h"

double AVW_ComputeCircularity(mask_image, mask_value)
AVW_Image *mask_image;
int mask_value;
```

DESCRIPTION

AVW_ComputeCircularity() calculates the circularity of an two-dimensional region in an image.

Mask_image is an *AVW_Image* whose voxels are of data type *AVW_UNSIGNED_CHAR*.

Mask_value is the value of the pixels which define the region.

The circularity of a region is the ratio of the region's perimeter squared to its area. It takes on a minimum value of 4*pi for a disk, and takes on higher values for more complex shapes.

RETURN VALUES

AVW_ComputeCircularity() returns a double value equal to the region's perimeter squared divided by its area.

ERRORS

AVW_ComputeCircularity() will fail for the following reasons:

ILLIMG
Illegal Image.

ILLDT
Illegal Data Type.

ILLPAR
Illegal Parameter.

SEE ALSO

AVW_Compute2DShapeStats(), AVW_ComputeImageCentroid(), AVW_ComputeImageFractalSig(), AVW_ComputeImageIntensityStats(), AVW_ComputeMER(), AVW_ComputePerimeter(), AVW_ComputeRFF() , AVW_Image

NAME

AVW_ComputeDividedTrace – computes a new trace between two traces

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_ComputeDividedTrace(outside, inside, distance, middle)
AVW_PointList2 *outside, *inside, *middle;
double distance;
```

DESCRIPTION

AVW_ComputeDividedTrace() creates a trace, which is stored in an *AVW_PointList2*, between two other traces, *outside* and *inside*. The points of the computed trace are stored in *middle*.

The points of the computed trace are calculated by first determining the center of the inside trace. Rays are then cast in a full circle from this center point. The intersection of each ray with the *inside* trace and *outside* trace are computed, along with the distance between these two points. The new points of the divided trace are positioned on these rays a fractional *distance* from the inside trace. *Distance* should be between 0.0 and 1.0.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Middle is provided as a method of reusing an existing *AVW_PointList2*.

RETURN VALUES

If successful *AVW_ComputeDividedTrace()* returns an *AVW_PointList2*. On failure it returns *AVW_NULL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeDividedTrace() will fail if:

BADMAL
  Malloc Failed. Unable to allocate memory.

SEE ALSO

AVW_AddPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_FillPointList2(), AVW_PointList2, AVW_Point2

NAME

AVW_ComputeFullWidthHalfMax – calculates full width half max measurements

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeFullWidthHalfMax(list, base, search, voxel_width,
                    voxel_height, fwhm)
AVW_PointValueList *list;
double base;
int search;
double voxel_width, voxel_height;
AVW_FullWidthHalfMax *fwhm;
```

DESCRIPTION

AVW_ComputeFullWidthHalfMax() calculates full width half max measurements from the line profile provided in *list*. *Base* specifies the base value used in the calculations. *Search* specifies whether the half maximums are searched for from the ends of the line profile, *AVW_SEARCHFROMOUTSIDE*, or from where the maximum value of the line profile occurs, *AVW_SEARCHFROMMAXIMUM*. *Voxel_width* and *voxel_height* are the calibrated sizes of the voxel from which the line profile was computed. *Fwhm* is the *AVW_FullWidthHalfMax* structure in which all of measurements are stored.

RETURN VALUES

If successful *AVW_ComputeFullWidthHalfMax()* returns an *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeFullWidthHalfMax() will fail if the following is true:

HMXERR
: Valid half maximum value could not be computed for the line profile.

SEE ALSO

AVW_ComputeLineProfile(), AVW_ComputeThickLineProfile(), AVW_FullWidthHalfMax(), AVW_PointValueList, AVW_Image

NAME

AVW_ComputeImageCentroid – calculates the centroid of a 2D object

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeImageCentroid(mask_image, mask_value, centroid, count)
AVW_Image *mask_image;
int mask_value,
AVW_FPoint2 *centroid;
int *count;
```

DESCRIPTION

AVW_ComputeImageCentroid() computes the centroid of a 2D object. All pixels of value *mask_value* in *mask_image* are taken as belonging to the object to be analyzed. *Mask_image* is an *AVW_Image* where the pixels are of data type *AVW_UNSIGNED_CHAR.*

Centroid is set to the computed object center which is the average of the x, y, and z coordinates of the pixels equal to *mask_value.*

Count is set to the number of pixels in the object.

RETURN VALUES

If successful *AVW_ComputeImageCentroid()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_Compute2DShapeStats(), AVW_ComputeCircularity(), AVW_ComputeImageFractalSig(), AVW_ComputeImageIntensityStats(), AVW_ComputeMER(), AVW_ComputePerimeter(), AVW_ComputeRFF(), AVW_FPoint2, AVW_Image

NAME

AVW_ComputeImageFractalSig – calculates fractal signature

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeImageFractalSig(in_image, mask_image, mask_val,
        scalemax, scalemin, slope, coeff, reg_pts,
                bfloat, areas)
AVW_Image *in_image;
AVW_Image *mask_image;
int mask_value:
int scalemax;
int scalemin;
double *slope;
double *coeff;
float reg_pts[30], bfloat[30], areas[30];
```

DESCRIPTION

AVW_ComputeImageFractalSig() computes the fractal signature of *in_image* in the region defined by *mask_image.* The fractal signature is calculated using pixels in *mask_image* equal to *mask_val* over the range of scales specified by *scalemin* and *scalemax*. The return value *slope* is the best fit approximation of the fractal dimension, and the value of *coef* is the correlation coefficient of the fit.

Reg_pts is the set of regression points over the range of scales.

Areas is the log(area) measurements over range of scales.

Bfloat is the area measurements over

RETURN VALUES

AVW_ComputeImageFractalSig() returns *AVW_SUCCESS* or *AVW_FAIL.*

SEE ALSO

AVW_ComputeImageIntensityStats(), AVW_Compute2DShapeStats(), AVW_Image

NAME

AVW_ComputeImageIntensityStats – calculates image density statistics

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeImageIntensityStats(in_image, mask_image, mask_val,
                        sample_max, sample_min, stats)
AVW_Image *in_image;
AVW_Image *mask_image;
int mask_val;
double sample_max, sample_min;
AVW_IntensityStats *stats;
```

DESCRIPTION

AVW_ComputeImageIntensityStats() computes intensity statistics on *in_image.*

The measured statistics include Mean, Standard Deviation, Variance, Sum, Sum of Squares, Number of Voxels, Area, Volume, Highest Intensity, Highest Intensity Location, Lowest Intensity, Lowest Intensity Location, Number Below Range, Number In Range, Number Above Range, and Brightness Area Product.

The *mask_image* allows the user to only sample a masked part of an image. If *mask_image* is not equal to *NULL* only pixel locations equal to the *mask_val* will be sampled from the image. If the mask is *NULL* the entire image is sampled. If a mask is specified it must have the same dimensions as the input image and have AVW_UNSIGNED_CHAR data type.

Sample_max and *sample_min* parameters specify a grey scale subrange of the masked region in which a separate set of statistics will be computed (See the *Range* statistics below).

The intensity statistics are returned in *stats,* which is an *AVW_IntensityStats* structure defined in *AVW_Measure.h.*

```
typedef struct
        {
        double Mean;
        double StandardDeviation;
        double Variance;
        double Sum;
        double SumOfSquares;
        unsigned long NumberOfVoxels;
        double Area;
        double Volume;
        double HighestIntensity;
        AVW_Point3 HighestPoint;
        double LowestIntensity;
        AVW_Point3 LowestPoint;
        double RangeMaximum;
        double RangeMinimum;
        double MeanInRange;
        double StandardDeviationInRange;
        double VarianceInRange;
        double SumInRange;
        double SumOfSquaresInRange;
        unsigned long NumberBelowRange;
        unsigned long NumberInRange;
        unsigned long NumberAboveRange;
```

```
double AreaInRange;
double VolumeInRange;
double BrightnessAreaProduct;
} AVW_IntensityStats;
```

RETURN VALUES

If successful *AVW_ComputeImageIntensityStats()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeImageIntensityStats() will fail if one of the following is true:

ILLIMG
Specified image was not valid.

ILLMSK
Specified mask was not valid.

SEE ALSO

AVW_ComputeVolumeIntensityStats() AVW_ComputeImageFractalSig(), AVW_ComputeLineProfile(), AVW_Compute2DShapeStats(), AVW_ResetIntensityStats(), AVW_SumIntensityStats(), AVW_IntensityStats, AVW_Image

NAME

AVW_ComputeLineProfile – calculates the intensities along a line or trace

SYNOPSIS

#include "AVW_Measure.h"

```
AVW_PointValueList *AVW_ComputeLineProfile(image, trace, length)
AVW_Image *image;
AVW_PointList2 *trace;
double *length;
```

DESCRIPTION

AVW_ComputeLineProfile() finds the intensities in *image* corresponding to the coordinates specified in *trace*. The *trace* is interpolated in this function thus allowing the user to pass in endpoints of a line. The *length* of the trace is calculated and returned. The length is calibrated to the size of the voxel in *image*.

RETURN VALUES

If successful *AVW_ComputeLineProfile()* returns an *AVW_PointValueList* containing the intensities and image coordinates of the profile. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeLineProfile() will fail if the following is true:

BADMAL
: Could not allocate memory for the results.

ILLIMG
: Image provided was not valid.

SEE ALSO

AVW_ComputeThickLineProfile(), AVW_ComputeFullWidthHalfMax(), AVW_ComputeImageIntensityStats(), AVW_ComputeVolumeIntensityStats(), AVW_PointList2, AVW_PointValueList, AVW_Image

NAME

AVW_ComputeMEB – calculates the minimum enclosing brick of an object

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeMEB(mask_volume, mask_value, resoultion, volume, angle, boxext)
AVW_Volume *mask_volume;
int mask_value;
double resolution;
double *volume;
double angle[3];
double boxext[3];
```

DESCRIPTION

AVW_ComputeMEB() calculates the minimum enclosing brick for an object defined as all voxels in *mask_vol* of value *mask_value,* where *mask_volume* is of data type *AVW_UNSIGNED_CHAR.*

Resolution specifies step size in degrees of rotation to search for the minimum enclosing brick.

Volume is used to return the volume (number of voxels) contained by the minimum enclosing brick.

Angle[0], angle[1], and *angle[2]* are set to the degrees of rotation on the x, y, and z axes respectively of the minimum enclosing brick.

Boxext[0], boxext[1], and *boxext[2]* are set to the x, y, and z size of the minimum enclosing brick.

RETURN VALUES

If successful *AVW_ComputeMEB()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeMEB() will fail for the following reasons:

ILLMSK
Illegal Mask.

ILLPAR
Illegal Parameter.

SEE ALSO

AVW_ComputeCircularity(), AVW_ComputeImageCentroid(), AVW_ComputeMER(), AVW_ComputePerimeter(), AVW_ComputeRFF(), AVW_ComputeVolume(), AVW_ComputeVolumeCentroid(), AVW_GetMaskBoundary(), AVW_Volume

NAME

AVW_ComputeMER – calculates the minimum enclosing rectangle of an area

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeMER(boundary, resolution, area, angle, aspect, merpts)
AVW_PointList2 *boundary;
double resolution;
double *area;
double *angle;
double *aspect;
AVW_FPointList2 *merpts;
```

DESCRIPTION

AVW_ComputeMER() computes information about the minimum enclosing rectangle for the boundary of an arbitrary two dimensional region expressed as an *AVW_PointList2.*

Resolution is the number of degrees per step to increment in finding the minimum enclosing rectangle.

Area is the pixel count of the minimum enclosing rectangle.

Angle is the rotational angle of the long axis of the minimum enclosing rectangle.

Aspect is the aspect ratio of the minimum enclosing rectangle.

Merpts is an *AVW_FPointList2* containing the coordinates of the four corners of the minimum enclosing rectangle.

RETURN VALUES

If successful *AVW_ComputeMER()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeMER() will fail for the following reasons:

ILLPAR
Illegal Parameter.

SEE ALSO

AVW_ComputeCircularity(), AVW_ComputeImageCentroid(), AVW_ComputeMEB(), AVW_ComputePerimeter(), AVW_ComputeRFF(), AVW_ComputeVolume(), AVW_ComputeVolumeCentroid(), AVW_GetMaskBoundary(), AVW_PointList2

NAME

AVW_ComputeObjectsStats – returns statistics about a connected object.

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ObjectMap.h"

int AVW_ComputeObjectStats(in_volume, om, seed, min, max, bound_box, centroid, count)
AVW_Volume *in_volume;
AVW_ObjetcMap *om;
AVW_Point3 *seed;
double min, max;
AVW_Rect3 *bound_box;
AVW_Point3 *centroid;
int *count;
```

DESCRIPTION

AVW_ComputeObjectsStats() returns bounding box, centroid, and volume statistics for a connected object.

In_volume specifies the input *AVW_Volume* which contains the region to be deleted or extracted.

Object_map may be specified to constrain the connect to objects which have the Display component enabled. If *NULL* is specified only the threshold limits and the *connectivity* will constrain the 3D region grow.

Seed is a pointer to an *AVW_Point3* structure. This structure contains the starting point for the 3D region grow.

Min and *max* specify the threshold limits used to constrain the 6 neighbor region growing process.

Bound_box is a pointer to an *AVW_Rect3* structure. This structure is used to return the coordinates of the minimum bounding box required to contain the object.

centroid is a pointer to an *AVW_Point3* structure. This structure is used to return the coordinate of the centroid of the object. The centroid is calculated by averaging each coordinate in the 3D region grow.

RETURN VALUES

If successful *AVW_ComputeObjectsStats()* returns *AVW_TRUE.* On failure it returns *AVW_FALSE* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeObjectsStats() will fail if one or more of the following is true:

BADMAL
: Unable to allocate sufficient memory.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_ConnectAndDeleteVolume() AVW_ConnectAndKeepVolume(), AVW_DefineConnected(), AVW_FindVolumeConponents(), AVW_ObjectMap, AVW_Point3, AVW_Rect3, AVW_Volume

NAME

AVW_ComputePerimeter – calculates the perimeter of a region

SYNOPSIS

```
#include "AVW_Measure.h"

double AVW_ComputePerimeter(boundary)
AVW_PointList2 *boundary;
```

DESCRIPTION

AVW_ComputePerimeter() calculates the length in voxels of a *boundary* stored in an *AVW_PointList2.*

The algorithm measures the distance around a piecewise linear closed curve constructed between the centers of the outermost pixels of a region. The path always consists of N pixel-dimension steps plus M square-root-of-two pixel dimension steps. The perimeter value will change with orientation and will continue to increase in error for larger analytical regions because all of the "wandering" of the digital line about the "true" curve is known to be error.

RETURN VALUES

If successful *AVW_ComputePerimeter()* returns a floating point value equal to the perimeter of the *boundary*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_GetMaskBoundary(), AVW_ComputeImageCentroid(), AVW_ComputeVolumeCentroid(), AVW_ComputeRFF(), AVW_ComputeCircularity(), AVW_ComputeVolume(), AVW_ComputeMEB(), AVW_ComputeMER(). AVW_PointList2

NAME

AVW_ComputeRFF – calculates the rectangular fit factor of a 2D region

SYNOPSIS

```
#include "AVW_Measure.h"

double AVW_ComputeRFF(mask_image, mask_value)
AVW_Image *mask_image;
int mask_value;
```

DESCRIPTION

AVW_ComputeRFF() calculates the rectangular fit of a two-dimensional region in an image. *Mask_image* is an *AVW_Image* whose voxels are of data type *AVW_UNSIGNED_CHAR*. *Mask_value* is the value of the pixels which define the region. The rectangular fit value of a region is the ratio of the region's area to the area of its minimum enclosing rectangle.

RETURN VALUES

AVW_ComputeRFF() returns a double value equal to the region's area divided by the area of its minimum enclosing rectangle.

ERRORS

AVW_ComputeRFF() will fail for the following reasons:

ILLIMG
Illegal Image.

ILLDT
Illegal Data Type.

ILLPAR
Illegal Parameter.

SEE ALSO

AVW_ComputeCircularity(), AVW_ComputeImageCentroid(), AVW_ComputeMEB(), AVW_ComputeMER(), AVW_Image

NAME

AVW_ComputeThickLineProfile – calculates the intensities along a thick line

SYNOPSIS

#include "AVW_Measure.h"

AVW_PointValueList *AVW_ComputeThickLineProfile(image, line, thickness, length)
AVW_Image *image;
AVW_Line2 *line;
int thickness;
double *length;

DESCRIPTION

AVW_ComputeThickLineProfile() computes the intensities of a thick line in *image* corresponding to the coordinates specified in *line.* The *line* is interpolated in this function. Perpendicular pixels above and below the line within one half of the *thickness* of the line are averaged into the profile values. The *length* of the line is calculated and returned. The length is calibrated to the size of the voxel in *image.*

RETURN VALUES

If successful *AVW_ComputeThickLineProfile()* returns an *AVW_PointValueList* containing the averaged intensities and image coordinates of the profile. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeThickLineProfile() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

ILLIMG
Image provided was not valid.

SEE ALSO

AVW_ComputeLineProfile(), AVW_ComputeFullWidthHalfMax(), AVW_ComputeImageIntensityStats(), AVW_ComputeVolumeIntensityStats(), AVW_PointList2, AVW_PointValueList, AVW_Image

NAME

AVW_ComputeVolume – calculates the volume of an object

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeVolume(mask_volume, mask_value)
AVW_Volume *mask_volume;
int mask_value;
```

DESCRIPTION

AVW_ComputeVolume() calculates the volume of a mask in a volume.

Mask_volume is an *AVW_Volume* of data type *AVW_UNSIGNED_CHAR*.

Mask_value is the value of the voxels which define the object. The volume is an integer count of the number of voxels in the object.

RETURN VALUES

AVW_ComputeVolume() returns an integer value equal to the volume of the object.

SEE ALSO

AVW_ComputeImageCentroid(), AVW_ComputeImageIntensityStats(), AVW_ComputeMEB(), AVW_ComputeVolumeCentroid(), AVW_ComputeVolumeIntensityStats(), AVW_Volume

NAME

AVW_ComputeVolumeCentroid – calculates the centroid of a 3D object

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeVolumeCentroid(mask_volume, mask_value, centroid, count)
AVW_Volume *mask_volume;
int mask_value,
AVW_FPoint3 *centroid;
int *count;
```

DESCRIPTION

AVW_ComputeVolumeCentroid() computes the centroid of a 3D object. All voxels of value *mask_value* in *mask_volume* are taken as belonging to the object to be analyzed. *Mask_volume* is an *AVW_Volume* where the voxels are of data type *AVW_UNSIGNED_CHAR*.

Centroid is set to the computed object center which is the average of the x, y, and z coordinates of voxels equal to *mask_value.*

Count is set to the number of pixels or voxels in the object.

RETURN VALUES

If successful *AVW_ComputeVolumeCentroid()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_ComputeImageCentroid(), AVW_ComputeCircularity(), AVW_ComputeMEB(), AVW_ComputeMEB(), AVW_ComputeRFF(), AVW_ComputeVolume(), AVW_FPoint3, AVW_Volume

NAME

AVW_ComputeVolumeIntensityStats – calculates volume density statistics

SYNOPSIS

```
#include "AVW_Measure.h"

int AVW_ComputeVolumeIntensityStats(in_vol, mask_vol, mask_val,
                        sample_max, sample_min, stats)
AVW_Volume *in_vol;
AVW_Volume *mask_vol;
int mask_val;
double sample_max, sample_min;
AVW_IntensityStats *stats;
```

DESCRIPTION

AVW_ComputeVolumeIntensityStats() computes intensity statistics on *in_vol.*

The measured statistics include Mean, Standard Deviation, Variance, Sum, Sum of Squares, Number of Voxels, Area, Volume, Highest Intensity, Highest Intensity Location, Lowest Intensity, Lowest Intensity Location, Number Below Range, Number In Range, Number Above Range, and Brightness Area Product.

The *mask_vol* allows the user to only sample a masked part of a volume. If *mask_vol* is not equal to *NULL* only voxel locations equal to the *mask_val* will be sampled from the volume. If the mask is *NULL* the entire volume is sampled. If a mask is specified it must have the same dimensions as the input volume and have AVW_UNSIGNED_CHAR data type.

Sample_max and *sample_min* parameters specify a grey scale subrange of the masked region in which a separate set of statistics will be computed (See the *Range* statistics below).

The intensity statistics are returned in *stats,* which is an *AVW_IntensityStats* structure.

```
typedef struct
        {
        double Mean;
        double StandardDeviation;
        double Variance;
        double Sum;
        double SumOfSquares;
        unsigned long NumberOfVoxels;
        double Area;
        double Volume;
        double HighestIntensity;
        AVW_Point3 HighestPoint;
        double LowestIntensity;
        AVW_Point3 LowestPoint;
        double RangeMaximum;
        double RangeMinimum;
        double MeanInRange;
        double StandardDeviationInRange;
        double VarianceInRange;
        double SumInRange;
        double SumOfSquaresInRange;
        unsigned long NumberBelowRange;
        unsigned long NumberInRange;
        unsigned long NumberAboveRange;
```

```
double AreaInRange;
double VolumeInRange;
double BrightnessAreaProduct;
} AVW_IntensityStats;
```

RETURN VALUES

If successful *AVW_ComputeVolumeIntensityStats()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ComputeVolumeIntensityStats() will fail if one of the following is true:

ILLVOL
Specified volume was not valid.

ILLMSK
Specified mask was not valid.

SEE ALSO

AVW_ComputeImageIntensityStats(), AVW_Compute2DShapeStats(), AVW_ComputeLineProfile(), AVW_ComputeMEB(), AVW_ComputeVolume(), AVW_ComputeVolumeCentroid(), AVW_ResetIntenstyStats(), AVW_SumIntensityStats(), AVW_IntensityStats, AVW_Volume

NAME

AVW_ConditionalDilateGreyVolume – conditionally dilates a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ConditionalDilateGreyVolume(in_volume, cond_volume,
        thresh_max, thresh_min, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *cond_volume;
double thresh_max, thresh_min;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ConditionalDilateGreyVolume() performs binary morphological dilation on *in_volume* using *element* as the structuring element and *cond_volume* as a conditional volume. *Cond_volume* may be of any data type and *thresh_max* and *thresh_min* are used to determine which voxels form the conditional volume. This function performs conditional dilation without requiring the existance of a thresholded conditional volume.

In_volume does not have to be a binary valued but all nonzero voxels are treated as ones. *In_volume*, *out_volume*, and *element* must be of the data type *AVW_UNSIGNED_CHAR*. *AVW_ConditionalDilateGreyVolume()* will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

The conditional dilation can be thought of as an enlargement of the binary objects contained in the input data limited by the *cond_volume*. In simplest terms the conditional dilation process takes place by translating the structuring element so that its centerpoint lies on every point of the input data. At each point, if a nonzero voxel in the structuring element corresponds to a nonzero voxel in the input data and the corresponding point in the conditional data is nonzero, the point in the output data is set to one. Otherwise the voxel is unchanged. The output data will only contain ones and zeros.

AVW_CreateStructuringVolume() or *AVW_CreateVolume()*, *AVW_SetVolume()*, and *AVW_PutVoxel()* may be used to create a structuring element. The structuring element is flipped on the X, Y, and Z axes prior to dilation within the routine.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConditionalDilateGreyVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConditionalDilateGreyVolume() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLVOL
: An illegal volume was passed to the function.

SEE ALSO

AVW_ConditionalDilateImage(), AVW_ConditionalDilateVolumeImage(), AVW_CreateStructuringVolume(), AVW_CreateVolume(), AVW_ErodeVolume(), AVW_DilateVolume(), AVW_MorphOpenVolume(), AVW_MorphCloseVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume(), AVW_PutVoxel(), AVW_SetVolume(), AVW_ULtimateErosionVolume(), AVW_Volume

NAME

AVW_ConditionalDilateImage – conditionally dilates an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ConditionalDilateImage(in_image, cond_image, element,
                        out_image)
AVW_Image *in_image;
AVW_Image *cond_Image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ConditionalDilateImage() performs binary morphological dilation on *in_image* using *element* as the structuring element and *cond_image* as a conditional image.

In_image does not have to be a binary valued but all nonzero pixels are treated as ones. *In_image, cond_image, out_image,* and *element* must be of the data type *AVW_UNSIGNED_CHAR. AVW_ConditionalDilateImage()* will allocate temporary storage space for the result if *in_image* and *out_image* are the same.

The conditional dilation can be thought of as an enlargement of the binary objects contained in the input data limited by the *cond_image.* In simplest terms the conditional dilation process takes place by translating the structuring element so that its centerpoint lies on every point of the input data. At each point, if a nonzero pixel in the structuring element corresponds to a nonzero pixel in the input data and the corresponding point in the conditional data is nonzero, the point in the output data is set to one. Otherwise the pixel is unchanged. The output data will only contain ones and zeros.

AVW_CreateStructuringImage() or *AVW_CreateImage(), AVW_SetImage(),* and *AVW_PutPixel()* may be used to create a structuring element. The structuring element is flipped on the X and Y axes prior to dilation within the routine.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConditionalDilateImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConditionalDilateImage() will fail if:

BADMAL
Malloc failed. Unable to allocate memory for results.

ILLDT
Data type is not defined or supported.

ILLIMG
An illegal image was passed to the function.

SEE ALSO

AVW_ConditionalDilateVolume(), AVW_CreateStructuringImage(), AVW_CreateImage(), AVW_DilateImage(), AVW_ErodeImage(), AVW_MorphCloseImage(), AVW_MorphOpenImage(), AVW_MorphMaxImage(), AVW_MorphMinImage(), AVW_SetImage(), AVW_PutPixel(), AVW_UltimateErosionImage(), AVW_Image

NAME

AVW_ConditionalDilateVolume – conditionally dilates a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ConditionalDilateVolume(in_volume, cond_volume,
                    element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *cond_volume;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ConditionalDilateVolume() performs binary morphological dilation on *in_volume* using *element* as the structuring element and *cond_volume* as a conditional volume.

In_volume does not have to be a binary valued but all nonzero voxels are treated as ones. *In_volume, cond_volume, out_volume,* and *element* must be of the data type *AVW_UNSIGNED_CHAR. AVW_ConditionalDilateVolume()* will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

The conditional dilation can be thought of as an enlargement of the binary objects contained in the input data limited by the *cond_volume.* In simplest terms the conditional dilation process takes place by translating the structuring element so that its centerpoint lies on every point of the input data. At each point, if a nonzero voxel in the structuring element corresponds to a nonzero voxel in the input data and the corresponding point in the conditional data is nonzero, the point in the output data is set to one. Otherwise the voxel is unchanged. The output data will only contain ones and zeros.

AVW_CreateStructuringVolume() or *AVW_CreateVolume(), AVW_SetVolume(),* and *AVW_PutVoxel()* may be used to create a structuring element. The structuring element is flipped on the X, Y, and Z axes prior to dilation within the routine.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConditionalDilateVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConditionalDilateVolume() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLVOL
: An illegal volume was passed to the function.

SEE ALSO

AVW_ConditionalDilateImage(), AVW_CreateStructuringVolume(), AVW_CreateVolume(), AVW_ErodeVolume(), AVW_DilateVolume(), AVW_MorphOpenVolume(), AVW_MorphCloseVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume(), AVW_PutVoxel(), AVW_SetVolume(), AVW_ULtimateErosionVolume(), AVW_Volume

NAME

AVW_ConnectAndDeleteImage – deletes 2D regions based on connectivity

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ConnectAndDeleteImage(in_image, seeds, connectivity, min,
max, deleted_value, count, out_image)
AVW_Image *in_image;
AVW_PointList2 *seeds;
int connectivity;
double min, max, deleted_value;
int *count;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ConnectAndDeleteImage() is used to set a connected region from an *AVW_Image* to a specified value.

In *AVW_ConnectAndDeleteImage()* threshold limits, *max* and *min*, are used to constrain the 4 or 8 neighbor region growing process which is started at every seed point in the *seeds* structure.

Connectivity may be either *AVW_4_CONNECTED* or *AVW_8_CONNECTED* and specifies the neighbors to be used to determine the connected components.

In_image specifies the input *AVW_Image* which contains the region to be deleted or extracted.

Seeds is a pointer to an *AVW_PointList2* structure. This structure must contain at least one point which will be used as the starting point(s) for the region growing process.

Deleted_value specifies the value to which each deleted pixel is set.

Count contains the number of pixels deleted if the function returns successfully. If the function fails *count* will contain the number at the time of failure.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConnectAndDeleteImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConnectAndDeleteImage() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_ConnectAndDeleteVolume() AVW_ConnectAndKeepImage(), AVW_FindImageComponents(), AVW_FindImageEdges(), AVW_Image, AVW_PointList2

NAME

AVW_ConnectAndDeleteVolume – deletes 3D regions based on connectivity

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ObjectMap.h"

AVW_Volume *AVW_ConnectAndDeleteVolume(in_volume, object_map, seeds,
        connectivity, min, max, deleted_value, count, out_volume)
AVW_Volume *in_volume;
AVW_ObjectMap *object_map;
AVW_PointList3 *seeds;
int connectivity;
double min, max, deleted_value;
int *count;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ConnectAndDeleteVolume() is used to set a connected region from an *AVW_Volume* to a specified value.

In *AVW_ConnectAndDeleteVolume()* the *AVW_ObjectMap*, if specified, further constrains the connect.

In_volume specifies the input *AVW_Volume* which contains the region to be deleted or extracted.

Object_map may be specified to constrain to connect to objects which have the Display component enabled. If *NULL* is specified only the threshold limits and the *connectivity* will constrain the 3D region grow.

Seeds is a pointer to an *AVW_PointList3* structure. This structure must contain at least one point which will be used as the starting point(s) for the 3D region grow.

Connectivity may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED* and specifies the neighbors to be used to determine the connected components.

Min and *max* specify the threshold limits used to constrain the 6 or 26 neighbor region growing process.

Deleted_value specifies the value that each deleted is changed to.

Count contains the number of voxels deleted if the function returns successfully. If the function fails count will contain the number at the time of failure.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConnectAndDeleteVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConnectAndDeleteVolume() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO *AVW_ConnectAndDeleteImage() AVW_ConnectAndKeepVolume(), AVW_DefineConnected(), AVW_FindVolumeConponents(), AVW_FindVolumeEdges(), AVW_ObjectMap, AVW_PointList3, AVW_Volume*

NAME

AVW_ConnectAndKeepImage – extracts 2D regions based on connectivity

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ConnectAndKeepImage(in_image, seeds, connectivity, min, max,
                deleted_value, count, out_image)
AVW_Image *in_image;
AVW_PointList2 *seeds;
int connectivity;
double min, max, deleted_value;
int *count;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ConnectAndKeepImage() is used to extract a connected region from an *AVW_Image.* The non-connected pixels are set to a specified value. A count of connected pixels is returned and can be used for area calculations.

In_image specifies the input *AVW_Image* which contains the region to be extracted.

Seeds is a pointer to an *AVW_PointList2* structure. This structure must contain at least one point which will be used as the starting point(s) for the region growing process.

Connectivity may be either *AVW_4_CONNECTED* or *AVW_8_CONNECTED* and specifies the neighbors to be used to determine the connected components.

Min and *max* specify the threshold limits used to constrain the region growing process. Voxels outside the threshold range are left unchanged.

Deleted_value specifies the value to which each non-extracted pixel is set.

Count contains the number of pixels extracted if the function returns successfully. If the function fails, count will contain the number at the time of failure.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConnectAndKeepImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConnectAndKeepImage() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_ConnectAndKeepVolume(), AVW_ConnectAndDeleteImage(), AVW_FindImageComponents(), AVW_PointList2, AVW_Image

NAME

AVW_ConnectAndKeepVolume – extracts 3D regions based on connectivity

SYNOPSIS

```
#include    "AVW.h"
#include    "AVW_ObjectMap.h"

AVW_Volume *AVW_ConnectAndKeepVolume(in_volume, object_map, seeds, connectivity,
                min, max, deleted_value, count, out_volume)
AVW_Volume *in_volume;
AVW_ObjectMap *object_map;
AVW_PointList3 *seeds;
int connectivity;
double min, max, deleted_value;
int *count;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ConnectAndKeepVolume() is used to extract a connected region from an *AVW_Volume*. The non-connected voxels are set to a specified value. A count of connected voxels is returned and can be used for volume calculations.

In_volume specifies the input *AVW_Volume* which contains the region to be extracted.

Seeds is a pointer to an *AVW_PointList3* structure. This structure must contain at least one point which will be used as the starting point(s) for the region growing process.

may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED* and specifies the neighbors to be used to determine the connected components.

Min and *max* specify the threshold limits used to constrain the region growing process. Voxels outside the threshold range are left unchanged.

Deleted_value specifies the value to which each non-extracted pixel is set.

Count contains the number of voxels extracted if the function returns successfully. If the function fails, count will contain the number at the time of failure.

If *Object_map* in not *NULL* the connect will be constrained to objects defined in the *object_map* which have the *DisplayFlag* component of the *AVW_Object* structure enabled. If *NULL* is specified only the threshold limits and the *connectivity* will constrain the 3D region growing process.

Seeds is a pointer to an *AVW_PointList3* structure. This structure must contain at least one point which will be used as the starting point(s) for the 3D, 6 neighbor region grow.

Min and *max* specify the threshold limits used to constrain the 3D region growing process.

Deleted_value specifies the value to which each non-extracted voxel is set.

Count contains the number of voxels deleted or extracted if the function returns successfully. If the function fails count will contain the number at the time of failure.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConnectAndKeepVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConnectAndKeepVolume() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_ConnectAndKeepImage() AVW_ConnectAndDeleteVolume(), AVW_DefineConnected(), AVW_FindVolumeComponents(), AVW_Object, AVW_ObjectMap, AVW_PointList3, AVW_Volume

NAME

AVW_ConstantOpImage – transforms an image mathematically

SYNOPSIS

#include "AVW_Parse.h"

```
AVW_Image *AVW_ConstantOpImage(value, operation, in_image, out_image)
double value;
int operation;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ConstantOpImage() applies the *operation* and *in_image* to *value* and returns the resulting image:

out_image = *value operation in_image*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

The following operations are defined in *AVW_Parse.h:*

AVW_OP_ADD

AVW_OP_SUB

AVW_OP_MUL

AVW_OP_DIV

AVW_OP_LT

AVW_OP_GT

AVW_OP_LE

AVW_OP_GE

AVW_OP_EQ

AVW_OP_NE

AVW_OP_AND

AVW_OP_OR

AVW_OP_MOD

RETURN VALUES

If successful *AVW_ConstantOpImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConstantOpImage() will fail if one or more of the following are true:

NOTSUP
Operation is not supported.

DIVZER
Division by zero.

BADMAL
Memory could not be allocated for results.

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_ConstantOpVolume(), AVW_ImageOpConstant(), AVW_ImageOpImage(), AVW_FunctionImage(), AVW_Image,

NAME

AVW_ConstantOpVolume – transforms a volume mathematically

SYNOPSIS

```
#include "AVW_Parse.h"

AVW_Volume *AVW_ConstantOpVolume(value, operation, in_volume, out_volume)
double value;
int operation;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ConstantOpVolume() applies the *operation* and *in_volume* to *value* and returns the resulting volume:

out_volume = *value operation in_volume*

AVW_ContantOpVolume() calls *AVW_ConstantOpImage()* with each slice in the input volume to produce the output volume.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

The following operations are defined in *AVW_Parse.h:*

AVW_OP_ADD

AVW_OP_SUB

AVW_OP_MUL

AVW_OP_DIV

AVW_OP_LT

AVW_OP_GT

AVW_OP_LE

AVW_OP_GE

AVW_OP_EQ

AVW_OP_NE

AVW_OP_AND

AVW_OP_OR

AVW_OP_MOD

RETURN VALUES

If successful *AVW_ConstantOpVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConstantOpVolume() will fail if one or more of the following are true:

NOTSUP
Operation is not supported.

DIVZER
Division by zero.

BADMAL
Memory could not be allocated for results.

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_ConstantOpImage(), AVW_VolumeOpConstant(), AVW_VolumeOpVolume(), AVW_FunctionVolume(), AVW_Volume

NAME

AVW_ConvertImage – converts a volume to a new data type

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ConvertImage(in_image, datatype, out_image)
AVW_Image *in_image;
int datatype;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ConvertImage() converts *in_image* to the specified *datatype.* Acceptable values for *datatype,* as defined in *AVW.h,* are: *AVW_UNSIGNED_CHAR, AVW_SIGNED_CHAR, AVW_UNSIGNED_SHORT, AVW_SIGNED_SHORT, AVW_UNSIGNED_INT, AVW_SIGNED_INT, AVW_FLOAT, AVW_COMPLEX, AVW_COLOR.*

If the volume is converted to a data type which can hold less information, or information in a different range, the data is clipped at the maximum and minimum values possible for the new data type. No intensity scaling is attempted during the conversion process.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConvertImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConvertImage() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for output volume.

ILLDT
Unknown or unsupported input or output datatype.

SEE ALSO

AVW_ConvertVolume(), AVW_DitherImage(), AVW_IntensityScaleImage(), AVW_MakeGrayImage() AVW_Image

NAME

AVW_ConvertSurfaceToVolume – draws a wiregrid surface into a volume

SYNOPSIS

```
#include       "AVW_Model.h"

AVW_Volume* AVW_ConvertSurfaceToVolume(surface, faces, truncated, fill)
AVW_TiledSurface *surface;
int* faces;
int* truncated;
int fill;
```

DESCRIPTION

AVW_ConvertSurfaceToVolume() creates a volume where all voxels along the polygonal edges defined in *surface* are set to 1. The remaining voxels are set to 0. *faces* returns the number of polygons drawn into the volume. *truncated* returns the number of polygons that were truncated by the volume's edges.

RETURN VALUES

If successful *AVW_DrawTiledSurface()* returns a volume. NULL is returned in the event of an error.

SEE ALSO

AVW_DrawVolumeLine(), AVW_DrawImageLine(), AVW_PutPixel(), AVW_PutVoxel(), AVW_Volume, AVW_Point3

NAME

AVW_ConvertVolume – converts a volume to a new data type

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ConvertVolume(in_volume, datatype, out_volume)
AVW_Volume *in_volume;
int datatype;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ConvertVolume() converts *in_volume* to the specified *datatype.* Acceptable values for *datatype,* as defined in *AVW.h,* are: *AVW_UNSIGNED_CHAR, AVW_SIGNED_CHAR, AVW_UNSIGNED_SHORT, AVW_SIGNED_SHORT, AVW_UNSIGNED_INT, AVW_SIGNED_INT, AVW_FLOAT, AVW_COMPLEX, AVW_COLOR.*

If the volume is convert to a data type which can hold less information, or information in a different range, the data is clipped at the maximum and minimum values possible for the new data type. No intensity scaling is attempted during the conversion process.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConvertVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConvertVolume() will fail if one or more of the following are true:

BADMAL
: Malloc Failed. Unable to allocate memory for output volume.

ILLDT
: Unknown or unsupported input or output datatype.

SEE ALSO

AVW_ConvertImage(), AVW_DitherVolume(), AVW_IntensityScaleVolume(), AVW_MakeGrayVolume() AVW_Volume

NAME

AVW_ConvolveImage – performs spatial convolution of an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ConvolveImage(in_image, psf, out_image)
AVW_Image *in_image;
AVW_Image *psf;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ConvolveImage() performs spatial convolution of an *AVW_Image. Psf* is the point spread function, and must have odd extents in all dimensions.

Since this convolution is performed in the spatial domain, the processing time will increase with the extent of the Psf. If the Psf is a significant fraction of the extent of *in_image,* it is more efficient to perform the convolution in the Fourier domain (see *AVW_FFT2D()).*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConvolveImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConvolveImage() will fail if the following is true:

BADMAL
: Malloc Failed. Could not allocate memory for the results.

ILLVOL
: Psf has even extent.

ILLDT
: Psf or in_image is of data type AVW_COMPLEX or AVW_COLOR.

SEE ALSO

AVW_ConvolveVolume(), AVW_CorrelateImage(), AVW_CorrelateVolume(), AVW_FFT2D(), AVW_Image

NAME

AVW_ConvolveVolume – performs spatial convolution of a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ConvolveVolume(in_volume, psf, out_volume)
AVW_Volume *in_volume;
AVW_Volume *psf;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ConvolveVolume() performs spatial convolution of an *AVW_Volume. Psf* is the point spread function, and must have odd extents in all dimensions.

Since this convolution is performed in the spatial domain, the processing time will increase with the extent of the Psf. If the Psf is a significant fraction of the extent of *in_volume,* it is more efficient to perform the convolution in the Fourier domain (see *AVW_FFT3D()).*

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ConvolveVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ConvolveVolume() will fail if the following is true:

BADMAL
Malloc Failed. Could not allocate memory for the results.

ILLVOL
Psf has even extent.

ILLDT
Psf or in_volume is of data type AVW_COMPLEX or AVW_COLOR.

SEE ALSO

AVW_ConvolveImage(), AVW_CorrelateVolume(), AVW_CorrelateImage, AVW_Volume()

NAME

AVW_CopyColormap – creates a copy of a colormap

SYNOPSIS

```
#include "AVW.h"

AVW_Colormap *AVW_CopyColormap(in_colormap, out_colormap)
AVW_Colormap *in_colormap, *out_colormap;
```

DESCRIPTION

AVW_CopyColormap() returns a copy of *in_colormap.*

Out_colormap is provided as a method of reusing an existing *AVW_Colormap.* Reuse is possible only if the size of the provided *out_colormap* meet the requirements of the function. In this case the pointer to *out_colormap* is returned by the function. If not reuseable *out_colormap* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyColormap()* returns an *AVW_Colormap.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyColormap() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CreateColormap(), AVW_DestroyColormap(). AVW_Colormap

NAME

AVW_CopyFPointList2 – creates a copy of a point list

SYNOPSIS

#include "AVW.h"

AVW_FPointList2 *AVW_CopyFPointList2(in_pl, out_pl)
AVW_FPointList2 *in_pl, *out_pl;

DESCRIPTION

AVW_CopyFPointList2() returns a copy of *in_pl*.

Out_pl is provided as a method of reusing an existing *AVW_FPointList2*. Reuse is possible only if the size and data type of the provided *out_pl* meet the requirements of the function. In this case the pointer to *out_pl* is returned by the function. If not reuseable *out_pl* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyFPointList2()* returns an *AVW_FPointList2*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyFPointList2() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyIPointList2(), AVW_CopyPointList2(), AVW_CreateFPointList2(), AVW_DestroyFPointList2() AVW_FPointList2()

NAME

AVW_CopyFPointList3 – creates a copy of a point list

SYNOPSIS

```
#include "AVW.h"

AVW_FPointList3 *AVW_CopyFPointList3(in_pl, out_pl)
AVW_FPointList3 *in_pl, *out_pl;
```

DESCRIPTION

AVW_CopyFPointList3() returns a copy of *in_pl.*

Out_pl is provided as a method of reusing an existing *AVW_FPointList3.* Reuse is possible only if the size and data type of the provided *out_pl* meet the requirements of the function. In this case the pointer to *out_pl* is returned by the function. If not reuseable *out_pl* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyFPointList3()* returns an *AVW_FPointList3.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyFPointList3() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyIPointList3(), AVW_CopyPointList3(), AVW_CreateFPointList3(), AVW_DestroyFPointList3() AVW_FPointList3()

NAME

AVW_CopyIPointList2 – creates a copy of a point list

SYNOPSIS

```
#include "AVW.h"

AVW_IPointList2 *AVW_CopyIPointList2(in_pl, out_pl)
AVW_IPointList2 *in_pl, *out_pl;
```

DESCRIPTION

AVW_CopyIPointList2() returns a copy of *in_pl.*

Out_pl is provided as a method of reusing an existing *AVW_IPointList2.* Reuse is possible only if the size and data type of the provided *out_pl* meet the requirements of the function. In this case the pointer to *out_pl* is returned by the function. If not reuseable *out_pl* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyIPointList2()* returns an *AVW_IPointList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyIPointList2() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyFPointList2(), AVW_CopyPointList2(), AVW_CreateIPointList2(), AVW_DestroyIPointList2() AVW_IPointList2()

NAME

AVW_CopyIPointList3 – creates a copy of a point list

SYNOPSIS

#include "AVW.h"

AVW_IPointList3 *AVW_CopyIPointList3(in_pl, out_pl)
AVW_IPointList3 *in_pl, *out_pl;

DESCRIPTION

AVW_CopyIPointList3() returns a copy of *in_pl*.

Out_pl is provided as a method of reusing an existing *AVW_IPointList3*. Reuse is possible only if the size and data type of the provided *out_pl* meet the requirements of the function. In this case the pointer to *out_pl* is returned by the function. If not reuseable *out_pl* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyIPointList3()* returns an *AVW_IPointList3*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyIPointList3() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyFPointList3(), AVW_CopyPointList3(), AVW_CreateIPointList3(), AVW_DestroyIPointList3() AVW_IPointList3()

NAME

AVW_CopyImage – creates a copy of an image

SYNOPSIS

#include "AVW.h"

AVW_Image *AVW_CopyImage(in_image, out_image)
AVW_Image *in_image, *out_image;

DESCRIPTION

AVW_CopyImage() returns a copy of *in_image.*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyImage() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyVolume(), AVW_CreateImage(), AVW_DestroyImage() AVW_Image()

NAME

AVW_CopyMatrix – copies a transformation matrix

SYNOPSIS

#include "AVW.h"

AVW_Matrix *AVW_CopyMatrix(in_matrix, out_matrix)
AVW_Matrix *in_matrix, *out_matrix;

DESCRIPTION

AVW_CopyMatrix() returns a copy of *in_matrix*.

Out_matrix is provided as a method of reusing an existing *AVW_Matrix*. *(See Memory Usage* in the *AVW Programmer's Guide.)*

SEE ALSO

AVW_CreateMatrix(), AVW_InvertMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis(), AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_SetIdentityMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_Matrix

NAME

AVW_CopyObjectMap – creates a copy of an object map

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ObjectMap.h"

AVW_ObjectMap *AVW_CopyObjectMap(in_map, out_map)
AVW_ObjectMap *in_map, *out_map;
```

DESCRIPTION

AVW_CopyObjectMap() returns a copy of *in_map.*

Out_map is provided as a method of reusing an existing *AVW_ObjectMap.* Reuse is possible only if the size of the provided *out_map* meet the requirements of the function. In this case the pointer to *out_map* is returned by the function. If not reuseable *out_map* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyObjectMap()* returns an *AVW_ObjectMap.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyObjectMap() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CreateObjectMap(), AVW_DestroyObjectMap() AVW_ObjectMap()

NAME

AVW_CopyPointList2 – creates a copy of a point list

SYNOPSIS

#include "AVW.h"

AVW_PointList2 ∗AVW_CopyPointList2(in_pl, out_pl)
AVW_PointList2 ∗in_pl, ∗out_pl;

DESCRIPTION

AVW_CopyPointList2() returns a copy of *in_pl.*

Out_pl is provided as a method of reusing an existing *AVW_PointList2.* Reuse is possible only if the size of the provided *out_pl* meet the requirements of the function. In this case the pointer to *out_pl* is returned by the function. If not reuseable *out_pl* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyPointList2()* returns an *AVW_PointList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyPointList2() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyFPointList2(), AVW_CopyIPointList2(), AVW_CreatePointList2(), AVW_DestroyPointList2() AVW_PointList2()

NAME

AVW_CopyPointList3 – creates a copy of a point list

SYNOPSIS

```
#include "AVW.h"

AVW_PointList3 *AVW_CopyPointList3(in_pl, out_pl)
AVW_PointList3 *in_pl, *out_pl;
```

DESCRIPTION

AVW_CopyPointList3() returns a copy of *in_pl*.

Out_pl is provided as a method of reusing an existing *AVW_PointList3*. Reuse is possible only if the size of the provided *out_pl* meet the requirements of the function. In this case the pointer to *out_pl* is returned by the function. If not reuseable *out_pl* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyPointList3()* returns an *AVW_PointList3*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyPointList3() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyFPointList3(), AVW_CopyIPointList3(), AVW_CreatePointList3(), AVW_DestroyPointList3() AVW_PointList3()

NAME

AVW_CopyPointValueList – creates a copy of a point list

SYNOPSIS

#include "AVW.h"

AVW_PointValueList *AVW_CopyPointValueList(in_pl, out_pl)
AVW_PointValueList *in_pl, *out_pl;

DESCRIPTION

AVW_CopyPointValueList() returns a copy of *in_pl.*

Out_pl is provided as a method of reusing an existing *AVW_PointValueList.* Reuse is possible only if the size of the provided *out_pl* meet the requirements of the function. In this case the pointer to *out_pl* is returned by the function. If not reuseable *out_pl* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyPointValueList()* returns an *AVW_PointValueList.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyPointValueList() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyFPointList(), AVW_CopyIPointList(), AVW_CopyPointList(), AVW_CreatePointValueList(), AVW_DestroyPointValueList() AVW_PointValueList()

NAME

AVW_CopyRenderedImage – creates a copy of an rendered image structure

SYNOPSIS

#include "AVW.h"

AVW_RenderedImage *AVW_CopyRenderedImage(in_rendered, out_rendered)
AVW_RenderedImage *in_rendered, *out_rendered;

DESCRIPTION

AVW_CopyRenderedImage() returns a copy of *in_rendered.*

Out_rendered is provided as a method of reusing an existing *AVW_RenderedImage.* Reuse is possible only if the size and data type of the provided *out_rendered* meet the requirements of the function. In this case the pointer to *out_rendered* is returned by the function. If not reuseable *out_rendered* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyRenderedImage()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyRenderedImage() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_RenderVolume(), AVW_RenderOblique(), AVW_CubeSections(), AVW_IntersectingSections(), AVW_DestroyRenderedImage() AVW_RenderedImage()

NAME

AVW_CopyVolume – creates a copy of a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_CopyVolume(in_volume, out_volume)
AVW_Volume *in_volume, *out_volume;
```

DESCRIPTION

AVW_CopyVolume() returns a copy of *in_volume.*

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CopyVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CopyVolume() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_CopyImage(), AVW_DestroyVolume(), AVW_CreateVolume(), AVW_Volume

NAME

AVW_CorrelateImage – performs spatial correlation on two images

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_CorrelateImage(in_image, psf, out_image)
AVW_Image *in_image;
AVW_Image *psf;
AVW_Image *out_image;
```

DESCRIPTION

AVW_CorrelateImage() performs spatial cross-correlation of two images. *Psf* is the point spread function, and must have odd extent in all dimensions.

Since this correlation is performed in the spatial domain, the processing time will increase with the extent of the Psf. If the Psf is a significant fraction of the extent of *in_image*, it is more efficient to perform the correlation in the Fourier domain (see *AVW_FFT2D()).*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CorrelateImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CorrelateImage() will fail if the following is true:

BADMAL
Malloc Failed. Could not allocate memory for the results.

ILLIMG
Psf has even extent.

ILLDT
Psf or in_image is AVW_COMPLEX or AVW_COLOR data type.

SEE ALSO

AVW_CorrelateVolume(), AVW_ConvolveImage(), AVW_ConvolveVolume(), AVW_Image

NAME

AVW_CorrelateVolume – performs spatial correlation on two volumes

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_CorrelateVolume(in_volume, psf, out_volume)
AVW_Volume *in_volume;
AVW_Volume *psf;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_CorrelateVolume() performs spatial cross-correlation of two volumes. *Psf* is the point spread function, and must have odd extent in all dimensions.

Since this correlation is performed in the spatial domain, the processing time will increase with the extent of the Psf. If the Psf is a significant fraction of the extent of *in_volume*, it is more efficient to perform the correlation in the Fourier domain (see *AVW_FFT3D())*.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CorrelateVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CorrelateVolume() will fail if the following is true:

BADMAL
: Malloc Failed. Could not allocate memory for the results.

ILLVOL
: Psf has even extent.

ILLDT
: Psf or in_volume is AVW_COMPLEX or AVW_COLOR data type.

SEE ALSO

AVW_CorrelateImage(), AVW_ConvolveImage(), AVW_ConvolveVolume(), AVW_Volume

NAME

AVW_Counter – indicates current progress

SYNOPSIS

```
#include "AVW.h"

int AVW_Counter(count)
int count;
```

DESCRIPTION

AVW_Counter() is called by many *AVW* functions to indicate their current progress. *Count* is a positive value indicating the current status of the calling procedure. This function will not normally be called unless *AVW* has been extended, and the programmer wishes to report the progress of the extended routine. Use of this function should be limited to functions which would normally require many seconds to execute. Functions which report progress, should not call other functions which report progress. Nested progresses are not supported.

When *AVW_Counter()* is called, the *count* value is passed to a user defined function. This function is indicated to *AVW* by the *AVW_CounterFunction()* call.

The return value is used to indicate a interrupt status. If non-zero value (1) is returned, the progress function indicates the process should continue. If zero (0) is returned, the progress function is requesting that the function be terminated.

RETURN VALUES

If a call to *AVW_Counter()* returns zero, the function should be terminated. A non-zero return value indicates that processing should continue.

SEE ALSO

AVW_CounterFunction(), AVW_Progress(), AVW_ConnectAndDeleteVolume(), AVW_ConnectAndKeepVolume(), AVW_DefineConnected(), AVW_ExtendExternalLibs(), AVW_FindVolumeComponents(), AVW_LabelVolumeFromEdges()

NAME

AVW_CounterFunction – indicates the function which reports progress

SYNOPSIS

```
#include "AVW.h"

void AVW_CounterFunction(int (*function)())
```

DESCRIPTION

Many *AVW* routines require many seconds to execute. The status of these routines is generally not reported until the function completes. To allow such things as progress graphs to be generated while the function executes, many *AVW* functions call the *AVW_Counter()* function to indicate their status. When the *AVW_Counter()* function is called, the count value is passed on to a user specified function. The *AVW_CounterFunction()* is used to indicate which user supplied function is to be called.

By returning zero (0), the user supplied routine can indicate a desire to interrupt the function. A non-zero value indicates the processing should continue.

SEE ALSO

AVW_Counter(), AVW_ProgressFunction(), AVW_ConnectAndDeleteVolume(), AVW_ConnectAndKeepVolume(), AVW_DefineConnected(), AVW_FindVolumeComponents(), AVW_LabelVolumeFromEdges()

NAME

AVW_CreateButterworthCoeffs – creates a list of coefficients for a Butterworth filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_FilterCoeffs *AVW_CreateButterworthCoeffs(type, f1, f2,
            order, numsamples, coeffs)
int type;
int f1;
int f2;
int order;
int numsamples;
AVW_FilterCoeffs *coeffs;
```

DESCRIPTION

AVW_CreateButterworthCoeffs() creates a list of coefficients for a Butterworth filter. These coefficients may be used to create a 2-D or 3-D transfer function for Fourier-domain image processing. If *coeffs* is NULL or *coeffs->number* is not equal to *numsamples,* the function allocates new memory for the returned *coeffs. Type* may be *AVW_LOWPASS, AVW_HIGHPASS, AVW_BANDPASS,* or *AVW_NOTCH.*

The parameters *f1* and *f2* are corner-frequency indices. Only *f1* is used for AVW_HIGHPASS or AVW_LOWPASS filters.

Order determines how rapidly the filter response falls from 1. Butterworth filters fall off at the rate of one order of magnitude per decade of frequency per order, so a filter of order 2 drops two orders of magnitude per decade while a third order filter drops three orders of magnitude per decade.

RETURN VALUES

If successful *AVW_CreateButterworthCoeffs()* returns a *AVW_FilterCoeffs* filled with real frequency space coefficients for the specified Butterworth filter. The coefficients are designed to be used to create a 2D or 3D transfer function, and *numsamples* should equal (X/2 + 1) for an image of dimension X. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CreateButterworthCoeffs() will fail if the following is true:

BADMAL
: Malloc Failed. Could not allocate memory for the results.

SEE ALSO

AVW_CalculateSphericalMTF(), AVW_CreateCoeffs(), AVW_CreateCircularMTF(), AVW_CalculateGaussianCoeffs(), AVW_CreateCoeffs(), AVW_DestroyCoeffs(), AVW_FilterCoeffs

NAME

AVW_CreateCircularMTF – converts filter coefficients to circular frequency filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_CreateCircularMTF(coeffs, out_image)
AVW_FilterCoeffs *coeffs;
AVW_Image *out_image;
```

DESCRIPTION

AVW_CreateCircularMTF() converts filter coefficients *coeff* into a circularly symmetrical modulation transfer function.

An *AVW_FilterCoeffs* structure is a list of floating point coefficients describing a one-dimensional frequency domain filter. *AVW_CreateCircularMTF()* replicates the coefficients in a circular pattern within a 2-D image formatted to match the results of *AVW_FFT2D()*.

The *out_image* will be *coeffs->numsamples* pixels wide and 2*(*coeffs->numsamples* -1) pixels high.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CreateCircularMTF()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CreateCircularMTF() will fail if the following is true:

BADMAL
Malloc Failed. Could not allocate memory for the results.

SEE ALSO

AVW_CreateButterWorthCoeffs(), AVW_CreateCoeffs(), AVW_CreateGaussianCoeffs(), AVW_DestroyCoeffs(), AVW_CreateSphericalMTF(), AVW_FilterCoeffs, AVW_Image

NAME

AVW_CreateCoeffs – creates a list of filter coefficients

SYNOPSIS

```
#include       "AVW_Filter.h"

AVW_FilterCoeffs *AVW_CreateCoeffs(numsamples, coeffs)
int numsamples;
AVW_FilterCoeffs *coeffs;
```

DESCRIPTION

AVW_CreateCoeffs() creates a list of coefficients, *coeffs*, for a frequency domain filter. An *AVW_FilterCoeffs* structure is a list of floating point coefficients describing a one-dimensional frequency domain filter. These coefficients may be used to create a 2-D or 3-D transfer function for Fourier-domain image processing.

Numsamples is the number of samples in the *AVW_FilterCoeff*. *Coeffs* is provided as a method of reusing an existing *AVW_FilterCoeffs*. Reuse is possible only if the size and data type of the provided *coeffs* meet the requirements of the function. In this case the pointer to *coeffs* is returned by the function. If not reusable *coeffs* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CreateCoeffs()* returns an *AVW_FilterCoeffs*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CreateCoeffs() will fail if the following is true:

BADMAL
    Malloc Failed. Could not allocate memory for the results.

SEE ALSO

AVW_CreateSphericalMTF(), AVW_CreateCircularMTF(), AVW_CreateCoeffs(), AVW_CreateGaussianCoeffs(), AVW_DestroyCoeffs(), AVW_CreateButterworthCoeffs(), AVW_FilterCoeffs

NAME

AVW_CreateColormap – creates a colormap

SYNOPSIS

```
#include "AVW.h"

AVW_Colormap *AVW_CreateColormap(size)
int size;
```

DESCRIPTION

AVW_CreateColormap() is used to create an *AVW_Colormap.*

Size indicates the number of colors to be allocated. The value must be greater than or equal to 2. The returned colormap is initialized to a linear gray scale from 0 to 255.

RETURN VALUES

AVW_CreateColormap() returns an *AVW_Colormap.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateColormap() will fail if one or more of the following are true:

ILLPAR
Illegal Parameter. The passed sized parameter was out of range.

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_DestroyColormap(), AVW_CopyColormap(), AVW_Colormap

NAME

AVW_CreateCompositeInfo – initializes a composite info structure.

SYNOPSIS

#include "AVW_CompositeInfo.h"

AVW_CompositeInfo *AVW_CreateCompositeInfo()

DESCRIPTION

AVW_CreateCompositeInfo() creates, initializes, and returns an *AVW_CompositeInfo* structure. The An *AVW_CompositeInfo* is a structure which identifies tissues in a volume.

RETURN VALUES

If successful *AVW_CreateCompositeInfo()* returns a pointer to an *AVW_CompositeInfo* structure. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. The initial composite info structure contains a greyscale volor and opacities for the intensities between 0 and 255.

ERRORS

AVW_CreateCompositeInfo() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

SEE ALSO

AVW_DestroyCompositeInfo(), AVW_LoadCompositeInfo(), AVW_SaveCompositeInfo(), AVW_CompositeInfo

NAME

AVW_CreateFPointList2 – initializes an AVW list structure

SYNOPSIS

#include "AVW.h"

AVW_FPointList2 *AVW_CreateFPointList2(block_size)
int block_size;

DESCRIPTION

The *AVW* point list structures, defined in *AVW.h*, can be used for a variety of things including traces and stacks. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed.

AVW_CreateFPointList2() allocates memory for and initializes an *AVW_FPointList2*.

RETURN VALUES

If successful *AVW_CreateFPointList2()* returns an *AVW_FPointList2*.

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateFPointList2(), will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_CreateFPointList3(), AVW_CreateIPointList2(), AVW_CreateIPointList3(), AVW_CreatePointList2(), AVW_CreatePointList3(), AVW_CreatePointValueList(), AVW_AddFPoint2(), AVW_DestroyFPointList2(), AVW_FPointList2, AVW_FPoint2

NAME

AVW_CreateFPointList3 – initializes an AVW list structure

SYNOPSIS

#include "AVW.h"

AVW_FPointList3 *AVW_CreateFPointList3(block_size)
int block_size;

DESCRIPTION

The *AVW* point list structures, defined in *AVW.h*, can be used for a variety of things including traces and stacks. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed.

AVW_CreateFPointList3() allocates memory for and initializes an *AVW_FPointList3*.

RETURN VALUES

If successful *AVW_CreateFPointList3()* returns an *AVW_FPointList3*.

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateFPointList3() will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_CreateFPointList2(), AVW_CreateIPointList2(), AVW_CreateIPointList3(), AVW_CreatePointList2(), AVW_CreatePointList3(), AVW_CreatePointValueList(), AVW_AddFPoint3(), AVW_DestroyFPointList3(), AVW_FPointList3, AVW_FPoint3

NAME

AVW_CreateGaussianCoeffs – creates a list of coefficients for a Gaussian filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_FilterCoeffs *AVW_CreateGaussianCoeffs(dev, numsamples, coeffs)
double dev;
int numsamples;
AVW_FilterCoeffs *coeffs;
```

DESCRIPTION

AVW_CreateGaussianCoeffs() creates a list of coefficients for a frequency domain Gaussian Filter. *Dev* determines the standard deviation of the of the frequency-space Gaussian. *Numsamples* determines the number of samples in the *AVW_FilterCoeffs structure.* These coefficients may be used to create a 2-D or 3-D transfer function for Fourier-domain image processing. The Gaussian function is zero-mean with a standard deviation (in samples) as specified by the input variable. This will effect a Gaussian low-pass filter when used with *AVW_CreateSphericalMTF() or AVW_CreateCircularMTF().*

Note: in order to create a filter for a NxN image, a N / 2 + 1 sample *AVW_FilterCoeffs* structure is required.

Coeffs is provided as a method of reusing an existing *AVW_FilterCoeffs.* Reuse is possible only if the size of the provided *coeffs* meet the requirements of the function. In this case the pointer to *coeffs* is returned by the function. If not reusable *coeffs* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CreateGaussianCoeffs()* returns an *AVW_FilterCoeffs.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CreateGaussianCoeffs() will fail if the following is true:

BADMAL
Malloc Failed. Could not allocate memory for the results.

SEE ALSO

AVW_CreateSphericalMTF(), AVW_CreateCircularMTF(), AVW_CreateCoeffs(), AVW_DestroyCoeffs(), AVW_CreateButterworthCoeffs(), AVW_FilterCoeffs

NAME

AVW_CreateHistogram – creates an AVW_Histogram

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Histogram *AVW_CreateHistogram(mem, max, min, step)
double *mem;
double max;
double min;
double step;
```

DESCRIPTION

AVW_CreateHistogram() creates an *AVW_Histogram* of the requested size. Memory for the histogram may be specified in the call or allocated by the function but must be of type double.

Max, *min* and *step* specify the size of the histogram.

Mem is the actual memory used to store the histogram. The *Mem* element of the *AVW_Histogram* is set to this address. The memory pointed at by *mem* must be large enough to accomodate the size of the specified *AVW_Histogram*.

If *NULL* is provided for *mem* then *AVW_CreateHistogram* will allocate memory for the histogram.

RETURN VALUES

If successful *AVW_CreateHistogram()* returns an *AVW_Histogram*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateHistogram() will fail if one or more of the following are true:

BADMAL
: Malloc Failed. Unable to allocate memory for structure and / or histogram.

SEE ALSO

AVW_ClearHistogram(), AVW_DestroyHistogram(), AVW_FlattenImageHistogram(), AVW_FlattenVolumeHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_MatchImageHistogram(), AVW_MatchVolumeHistogram(), AVW_NormalizeHistogram(), AVW_PreserveImageHistogram(), AVW_PreserveVolumeHistogram(), AVW_VerifyHistogram(), AVW_Histogram

NAME

AVW_CreateIPointList2 – initializes an AVW list structure

SYNOPSIS

#include "AVW.h"

AVW_IPointList2 *AVW_CreateIPointList2(block_size)
int block_size;

DESCRIPTION

The *AVW* point list structures, defined in *AVW.h*, can be used for a variety of things including traces and stacks. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed.

AVW_CreateIPointList2() allocates memory for and initializes an *AVW_IPointList2*.

RETURN VALUES

If successful *AVW_CreateIPointList2()* returns an *AVW_IPointList2*.

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateIPointList2(), will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_CreateIPointList3(), AVW_CreateFPointList2(), AVW_CreateFPointList3(), AVW_CreatePointList2(), AVW_CreatePointList3(), AVW_CreatePointValueList(), AVW_AddIPoint2(), AVW_DestroyIPointList2(), AVW_IPointList2, AVW_IPoint2

NAME

AVW_CreateIPointList3 – initializes an AVW list structure

SYNOPSIS

#include "AVW.h"

AVW_IPointList3 *AVW_CreateIPointList3(block_size)
int block_size;

DESCRIPTION

The *AVW* point list structures, defined in *AVW.h,* can be used for a variety of things including traces and stacks. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed.

AVW_CreateIPointList3() allocates memory for and initializes an *AVW_IPointList3.*

RETURN VALUES

If successful *AVW_CreateIPointList3()* returns an *AVW_IPointList3.*

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateIPointList3() will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_CreateIPointList2(), AVW_CreateFPointList2(), AVW_CreateFPointList3(), AVW_CreatePointList2(), AVW_CreatePointList3(), AVW_CreatePointValueList(), AVW_AddIPoint3(), AVW_DestroyIPointList3(), AVW_IPointList3, AVW_IPoint3

NAME

AVW_CreateImage – creates an AVW_Image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_CreateImage(mem, width, height, type)
void *mem;
int width, height, type;
```

DESCRIPTION

AVW_CreateImage() creates *AVW_Image* of the specified size and data type. Memory for the pixel data may be specified in the call or allocated by the function.

Width and *height* specify the x and y dimensions of the image in pixels.

Type specifies the AVW data type of the pixels. Acceptable values are: *AVW_UNSIGNED_CHAR, AVW_SIGNED_CHAR, AVW_UNSIGNED_SHORT, AVW_SIGNED_SHORT, AVW_UNSIGNED_INT, AVW_SIGNED_INT, AVW_FLOAT, AVW_COMPLEX, AVW_COLOR.*

Mem is the actual memory used to store the pixel data for this image. The *Mem* element of the *AVW_Image* is set to this address. The memory pointed at by *mem* must be large enough to accomodate the size and data type of the specified *AVW_Image.*

If *NULL* is provided for *mem* then *AVW_CreateImage* will allocate memory for the pixel data.

The memory that is allocated by *AVW_CreateImage* is contiguous. Pixels in the memory are referenced from *image*->*Mem.*

Additionally *YTable* is an allocated offset array in the structure which enables efficient addressing of individual pixels within the space. This offset array is indexed as any other 'c' array. This means that the offset to the first line of the image is *YTable[0].* These values can be added to the Mem element of the image structure to reference any line from an *AVW_Image.*

RETURN VALUES

If successful *AVW_CreateImage()* returns an *AVW_Image.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateImage() will fail if one or more of the following are true:

ILLDT
Illegal data type given in type.

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_CreateVolume(), AVW_DestroyImage(), AVW_Image

NAME

AVW_CreateImageFile – creates a new AVW_ImageFile

SYNOPSIS

```
#include "AVW_ImageFile.h"

AVW_ImageFile *AVW_CreateImageFile (filename, format, width, height, depth, datatype)
char *filename;
char *format;
int width;
int height;
int depth;
int datatype;
```

DESCRIPTION

AVW_CreateImageFile() allocates an *AVW_ImageFile* structure, initializes its attributes to the call parameters and then calls the appropriate create function for the specified *format.*

A list of acceptable formats which *AVW* supports for writing data is obtained from *AVW_ListFormats(AVW_SUPPORT_WRITE);*. Built-in supported file formats for writing data are:
AnalyzeAVW
AVW_VolumeFile

These writable formats are available through dynamically loaded libraries
AnalyzeImage(7.5)
AnalyzeScreen
SunRaster
PPM
PGM
BMP
RGB
SGIrgb
PIC

The *filename* parameter is the name of the file which is being created.

Width and *height* are the dimensions of the images in the file. *Depth* is the number of images in a volume. This value is 1 for data formats which do not support 3D. *Datatype* is the *AVW_DataType* of the images which will be written to the file.

This function should be used instead of *AVW_OpenImageFile()* to create a new file.

RETURN VALUES

If successful *AVW_CreateImageFile()* returns an *AVW_ImageFile.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of failure.

ERRORS

AVW_CreateImageFile() will fail for these reasons:

BADMAL
Bad Malloc. Unable to allocate sufficient memory.

BDCRT
Bad Create. Unable to create file.

ILLEGAL_DATATYPE
Illegal Datatype. The file format does not support the indicated datatype.

INCOMPATIBLE_VOLUME
Incompatible Volume. The file format does not support 3D.

NOPERMISSION
File Permission. Cannot be written by user.

NOSPACE
No Space On Device. Disk is full.

RDONLFS
Read Only File System.

UNKNOWN_FORMAT
Unknown Format.

SEE ALSO *AVW_CloseImageFile(), AVW_ExtendImageFile(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume(), AVW_ImageFile*

NAME

AVW_CreateList – creates an AVW_List

SYNOPSIS

```
#include "AVW.h"

AVW_List *AVW_CreateList(NumberOfEntries)
int NumberOfEntries;
```

DESCRIPTION

AVW_CreateList() creates *AVW_List* of the specified number of entries.

The *AVW_List* structure provides an easy way to pass and return lists.

NumberOfEntries

is the number of entries in the list.

Entry is an array of (char

For example ...

```
list = AVW_CreateList(3);
list->Entry[0] = "List Element 1"
list->Entry[1] = "List Element 2"
list->Entry[2] = "List Element 3"
```

RETURN VALUES

If successful *AVW_CreateList()* returns an *AVW_List*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateImage() will fail if the following is true:

BADMAL
    Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_DestroyList(), AVW_ListFormats(), AVW_List

NAME

AVW_CreateMatrix – creates an AVW_Matrix

SYNOPSIS

#include "AVW.h"

AVW_Matrix *AVW_CreateMatrix(rows, columns)
int rows, columns;

DESCRIPTION

AVW_CreateMatrix() creates an *AVW_Matrix* of the specified dimensions. All values in the matrix are set to 0's expect the locations where the row and column indexs are the same, these are set to 1's, thus creating a Identity Matrix by default.

Rows and *Columns* can be any positive value, but most *AVW* function currently require 4 X 4.

RETURN VALUES

If successful *AVW_CreateMatrix()* returns an *AVW_Matrix*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateMatrix() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_DestroyMatrix(), AVW_Matrix

NAME

AVW_CreateObjectMap – initializes an object map

SYNOPSIS

```
#include "AVW_ObjectMap.h"

AVW_ObjectMap *AVW_CreateObjectMap(width, height, depth)
int width, height, depth;
```

DESCRIPTION

AVW_CreateObjectMap() creates, initializes, and returns an *AVW_ObjectMap* structure. The *AVW_ObjectMap* returned has one object called *Original. Width, height*, and *depth* specify the dimension of the *object_map->Volume.* This size must match the size of the *AVW_Volume* which will be rendered. An *AVW_ObjectMap* is a structure which identifies voxels in a volume as belonging to various structures and can be used by the *AVW_RenderVolume()* routine.

RETURN VALUES

If successful *AVW_CreateObjectMap()* returns a pointer to an *AVW_ObjectMap* structure. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CreateObjectMap() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_CopyObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_GetObject(), AVW_LoadObjectMap(), AVW_PutObject(), AVW_SaveObjectMap(), AVW_RenderVolume(), AVW_ObjectMap

NAME

AVW_CreatePointList2 – initializes an AVW list structure

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_CreatePointList2(block_size)
int block_size;
```

DESCRIPTION

The *AVW* point list structures can be used for a variety of things including traces and stacks. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed. See *AVW.h* for a definition of the structures.

AVW_CreatePointList2() allocates memory for and initializes an *AVW_PointList2*.

RETURN VALUES

If successful *AVW_CreatePointList2()* returns an *AVW_PointList2*.

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreatePointList2() will fail if the following is true:

BADMAL
: Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_CreateFPointList2(), AVW_CreateFPointList3(), AVW_CreateIPointList2(), AVW_CreateIPointList3(), AVW_CreatePointList3(), AVW_CreatePointValueList(), AVW_AddPoint2(), AVW_DestroyPointList2(), AVW_PointList2

NAME

AVW_CreatePointList3 – initializes an AVW list structure

SYNOPSIS

```
#include "AVW.h"

AVW_PointList3 *AVW_CreatePointList3(block_size)
int block_size;
```

DESCRIPTION

The *AVW* point list structures can be used for a variety of things including traces and stacks. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed. See *AVW.h* for a definition of the structures.

AVW_CreatePointList3() allocates memory for and initializes an *AVW_PointList3.*

RETURN VALUES

If successful *AVW_CreatePointList3()* returns an *AVW_PointList3.*

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreatePointList3() will fail if the following is true:

BADMAL
: Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_CreateFPointList2(), AVW_CreateFPointList3(), AVW_CreateIPointList2(), AVW_CreateIPointList3(), AVW_CreatePointList2(), AVW_CreatePointValueList(), AVW_AddPoint3(), AVW_DestroyPointList3(), AVW_PointList3

NAME

AVW_CreatePointValueList – initializes an AVW list structure

SYNOPSIS

```
#include "AVW.h"

AVW_PointValueList *AVW_CreatePointValueList(block_size)
int block_size;
```

DESCRIPTION

The *AVW* point list structures can be used for a variety of things including traces and stacks. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed. See *AVW.h* for a definition of the structures.

AVW_CreatePointValueList() allocates memory for and initializes an *AVW_PointValueList*.

RETURN VALUES

If successful *AVW_CreatePointValueList()* returns an *AVW_PointValueList*.

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreatePointValueList(), will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_CreateFPointList2(), AVW_CreateFPointList3(), AVW_CreateIPointList2(), AVW_CreateIPointList3(), AVW_CreatePointList2(), AVW_CreatePointList3(), AVW_AddPointValue(), AVW_DestroyPointValueList() AVW_PointValueList

NAME

AVW_CreateSphericalMTF – converts filter coefficients to spherical frequency filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_CreateSphericalMTF(coeffs, out_vol)
AVW_FilterCoeffs *coeffs;
AVW_Volume *out_vol;
```

DESCRIPTION

AVW_CreateSphericalMTF() converts filter coefficients *coeff* into a spherically symmetrical modulation transfer function.

An *AVW_FilterCoeffs* structure is a list of floating point coefficients describing a one-dimensional frequency domain filter. *AVW_CreateSphericalMTF()* replicates the coefficients in a spherical pattern within a 3-D volume formatted to match the results of *AVW_FFT3D().*

The *out_vol* will be coeffs->numsamples voxels wide and 2*(coeffs->numsamples-1) voxels high, and 2*(coeffs->numsamples-1) voxels deep.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CreateSphericalMTF()* returns an *AW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CreateSphericalMTF() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

SEE ALSO

AVW_CreateCircularMTF(), AVW_CreateCoeffs(), AVW_CreateButterworthCoeffs(), AVW_CreateGaussianCoeffs() AVW_CreateStoksethMTF() AVW_DestroyCoeffs(), AVW_FilterCoeffs, AVW_Volume

NAME

AVW_CreateStoksethMTF – creates a Stokseth 2D frequency-space filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_CreateStoksethMTF(xnum, slice_no, slice_thick,
            numap, pixel_width, wavelength,
            refr_index, focal_dist, out_image)
int xnum, slice_no;
double slice_thick, numap, pixel_width;
double wavelength, refr_index, focal_dist;
AVW_Image *out_image;
```

DESCRIPTION

AVW_CreateStoksethMTF() creates a 2D frequency-space filter which corresponds to the modulation transfer function of an optical microscope (see reference below). The filter will be of data type *AVW_COMPLEX* and of dimensions *xnum* / 2 + 1, *xnum*.

The input parameter *xnum* refers to the corresponding image dimensions and must be a power of 2.

Slice_no is zero if the filter is to be calculated for the in-focus slice and a positive number if the filter is to be calculated for an out-of-focus slice that many slices above or below the in-focus slice.

The distance between slices is given by *slice_thick,* measured in microns.

Numap is the numerical aperture of the objective lens.

Pixel_width is the distance between adjacent pixels in x and y.

Wavelength is the wavelength of the light being used (in microns).

Refr_index is the index of refraction of the immersion medium.

Focal_dist is the distance between the objective and the focal plane (in microns).

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CreateStoksethMTF()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CreateStoksethMTF() will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for return volume or image.

ILLPAR
Illegal Parameter. An invalid value was given for an input parameter.

REFERENCES

Agard, David A., *Optical Sectioning Microscopy: Cellular Architecture in Three Dimensions* 1984, Annual Review of Biophysics and Bioengineering 13: 191-219.

SEE ALSO

AVW_DeconvDivideImage(), AVW_DeconvWienerImage(), AVW_IterDeconvImage(), AVW_NearestNeighborDeconv(), AVW_DestroyCoeffs(), AVW_FilterCoeffs, AVW_Volume

NAME

AVW_CreateStructuringImage – creates a structuring element for morphology operations

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_CreateStructuringImage(type, width, height, out_image)
int type;
int width;
int height;
AVW_Image *out_image;
```

DESCRIPTION

AVW_CreateStructuringImage() creates an *AVW_Image* of the specified *type*, *width* and *height*. The generated image can be used as the stucturing element in calls to morphology routines.

Type may be either AVW_4_CONNECTED or *AVW_8_CONNECTED*. *AVW_4_CONNECTED* specifies that a 2D cross, i.e. ones on the center row and column of the image, will be generated. *AVW_8_CONNECTED* specifies that a solid rectangle, i.e., ones in the entire image, will be generated.

Width and *height* specify the x and y dimensions of the structuring element.

The returned image will be of datatype *AVW_UNSIGNED_CHAR*.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CreateStructuringImage()* returns an *AVW_Image*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

SEE ALSO

AVW_CreateStructuringVolume(), AVW_CreateImage(), AVW_DestroyImage(), AVW_ErodeImage(), AVW_DilateImage(), AVW_ConditionalDilateImage(), AVW_MorphCloseImage(), AVW_MorphOpenImage(), AVW_MorphMaxImage(), AVW_MorphMinImage() AVW_NonMaxImage() AVW_UltimateErosionImage(), AVW_Image

NAME

AVW_CreateStructuringVolume – creates a structuring element for morphology operastions

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_CreateStructuringVolume(type, width, height, depth,
                    out_volume)
int type;
int width;
int height;
int depth;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_CreateStructuringVolume() creates an *AVW_Volume* of the specified *type*, *width*, *height* and *depth*. The generated volume can be used as the stucturing element in calls to morphology routines.

Type may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED*. *AVW_6_CONNECTED* specifies that a 3D "jack shaped" structuring element will be generated. The structuring element will have ones on the center row and column of the center image of the volume, and ones on the center voxel of every image in the volume. *AVW_26_CONNECTED* specifies that a solid box, i.e.,ones in the entire volume, will be generated.

Width, *height* and *depth* specify the x, y, and z dimensions of the structuring element.

The returned volume will be of data type *AVW_UNSIGNED_CHAR*.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CreateStructuringVolume()* returns an *AVW_Volume*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

SEE ALSO

AVW_CreateStructuringImage(), AVW_CreateVolume(), AVW_DestroyVolume(), AVW_ErodeVolume(), AVW_DilateVolume(), AVW_ConditionalDilateVolume(), AVW_MorphCloseVolume(), AVW_MorphopenVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume() AVW_NonMaxVolume(), AVW_UltimateErosionVolume(), AVW_Volume

NAME

AVW_CreateTree – initializes an AVW tree structure

SYNOPSIS

#include "AVW.h"

AVW_Tree *AVW_CreateTree(block_size)
int block_size;

DESCRIPTION

The *AVW* tree structure is used to store points and related information of skeletal trees. They are initially allocated to store *block_size* points and are increased by *block_size* points as needed. See *AVW_Tree.h* for a definition of the structures.

AVW_CreateTree() allocates memory for and initializes an *AVW_Tree.*

RETURN VALUES

If successful *AVW_CreateTree()* returns an *AVW_Tree.*

On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateTree() will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for structure.

SEE ALSO

AVW_AddTreeChild(), AVW_DestroyTree(), AVW_FindTreeIndex(), AVW_LoadTree(), AVW_SaveTree(), AVW_TreePoint, AVW_Tree

NAME

AVW_CreateVolume – creates an AVW_Volume

SYNOPSIS

```
#include    "AVW.h"

AVW_Volume *AVW_CreateVolume(mem, width, height, depth, type)
void *mem;
int width, height, depth, type;
```

DESCRIPTION

AVW_CreateVolume() creates an *AVW_Volume* of the specified size and data type. The *Mem* element of the *AVW_Volume* is set to the address *mem.*

Width, height, and *depth* specify the x, y, and z dimensions in voxels of the requested volume.

Type specifies the *AVW* data type of the voxels. Acceptable values are: *AVW_UNSIGNED_CHAR, AVW_SIGNED_CHAR, AVW_UNSIGNED_SHORT, AVW_SIGNED_SHORT, AVW_UNSIGNED_INT, AVW_SIGNED_INT, AVW_FLOAT, AVW_COMPLEX, AVW_COLOR.*

Mem is the actual memory used to store the voxel data. The *Mem* element of the *AVW_Volume* is set to this address. The memory pointed at by *mem* must be large enough to accomodate the size and data type of *AVW_Volume* specified by *width, height, depth,* and *type.* If *NULL* is provided for *mem* then *AVW_CreateVolume* allocates memory for the voxel data.

The memory that is allocated by *AVW_CreateVolume* is contiguous. Voxels in the memory are referenced from *vol->Mem, where vol* is the returned *AVE_Volume.*

Additionally *ZTable* and *YTable* are allocated offset arrays in the structure which enable efficient addressing of individual voxels within the space. These offset arrays are indexed the same as any other C language arrays. This means that the offset to the first image in a volume is *ZTable[0].* The offset of the last image in a volume is at *ZTable[depth-1].*

ZTable[] is a table of offset in voxels to each image slice. For example, to reference the start of the 10th image in a floating point *AVW_Volume:*

```
address = (float *)vol->Mem + vol->ZTable[9];
```

YTable[] is a table of offsets in voxels to each line in the first image of *Mem*. These values can be added to any image start to reference any line from any image in an *AVW_Volume.*

For example, to reference the first voxel in 10th image and 25th line in an integer *AVW_Volume:*

```
address = (int *)vol->Mem + vol->ZTable[9] + vol->YTable[24];
```

ZTable and *YTable* are created even if *mem* was preallocated and supplied by the user.

RETURN VALUES

If successful *AVW_CreateVolume()* returns an *AVW_Volume.* On failure *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_CreateVolume() will fail if:

BADMAL

Malloc Failed. Unable to allocate memory for structure and / or voxels.

SEE ALSO *AVW_CreateImage(), AVW_DestroyVolume(), AVW_Volume*

NAME

AVW_CropImage – automatically crops a border from an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_CropImage(in_image, out_image)
AVW_Image *in_image, *out_image;
```

DESCRIPTION

AVW_CropImage() returns a smaller cropped version of *in_image.*

The value of the first pixel in the image is used to crop the left and bottom sides of the image. The value of the last pixel in the image is used to crop the top amd right sides of the image. All rows and columns of the image in which each pixel is equal to the test value is removed in the returned. *out_image.*

If AVW_CropImage was able to remove bordering pixels and entry in the Info string for the returned *out_image* is made with the tag "AVW_CropImage" followed by a string containing four integers defining the cropped sub region: left column, bottom row, right column, top row. If AVW_CropImage is unable to remove bordering pixels a copy of *in_image is returned and entry is*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CropImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CropImage() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_GetSubImage(), AVW_Image()

NAME

AVW_CubeSections – renders intersecting sections

SYNOPSIS

```
#include "AVW_Render.h"

AVW_RenderedImage *AVW_CubeSections(volume, lowx, lowy, lowz, highx, highy, highz,
                        matrix, interpolate_flag, shading_fraction,
                        last_rendered)
AVW_Volume *volume;
int lowx, lowy, lowz, highx, highy, highz
AVW_Matrix *matrix;
int interpolate_flag;
double shading_fraction;
AVW_RenderedImage *last_rendered;
```

DESCRIPTION

AVW_CubeSections() returns an *AVW_RenderedImage* showing all sides of a cube representing a specified sub-volume.

Volume specifies the *AVW_Volume* the edge sections are extracted from.

Lowx, *lowy*, *lowz*, *highx*, *highy*, and *highz* specify the sub-volume to show.

Matrix is used to specify any rotation or scale factors.

The *interpolate_flag* specifies if tri-linear interpolations should be used when generating the image. Setting the *interpolate_flag* to *AVW_FALSE* causes *Nearest Neighbor* to be used, which is much faster, but lacks some of the quality.

Last_rendered is provided as a method of reusing an existing *AVW_RenderedImage.* Reuse is possible only if the size and data type of the provided *last_rendered* meet the requirements of the function. In this case the pointer to *last_rendered* is returned by the function. If not reusable *last_rendered* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_CubeSections()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_CubeSections() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_IntersectingSections(), AVW_DestroyRenderedImage(), AVW_RenderedImage

NAME

AVW_CubicSplineInterpolatedPixel – returns pixel value at a floating point location

SYNOPSIS

#include "AVW.h"

double AVW_CubicSplineInterpolatedPixel(image, point)
AVW_Image *image;
AVW_FPoint2 *point;

DESCRIPTION

Given a floating point location *point* within *image, AVW_CubicSplineInterpolatedPixel()* returns the calculated pixel value at the floating point location.

Uses a Cubic Spline algorithm to estimate the pixel value at the floating point location.

Points outside the image will return a value of 0.0.

SEE ALSO

AVW_GetPixel(), AVW_GetErrorNumner(), AVW_InterpolatedVoxel(), AVW_NearestNeighborPixel(), AVW_InterpolatedPixel(), AVW_SincInterpolatedPixel(), AVW_Image, AVW_FPoint2

NAME

AVW_CubicSplineInterpolatedVoxel – returns voxel value at a floating point location

SYNOPSIS

#include "AVW.h"

double AVW_CubicSplineInterpolatedVoxel(volume, point)
AVW_Volume *volume;
AVW_FPoint3 *point;

DESCRIPTION

Given a floating point location *point* within *volume, AVW_CubicSplineInterpolatedVoxel()* returns the calculated voxel value at the floating point location.

A Cubic Spline algorithm is used to estimate the voxel value at the floating point location.

Points outside the volume will return a value of 0.0.

SEE ALSO

AVW_GetVoxel(), AVW_GetErrorNumner(), AVW_InterpolatedPixel(), AVW_NearestNeighborVoxel(), AVW_InterpolatedVoxel(), AVW_SincInterpolatedVoxel(), AVW_FPoint3, AVW_Volume

NAME

AVW_DataTypeToBands – returns number of bands in the specified data type

SYNOPSIS

```
#include "AVW.h"

int AVW_DataTypeToBands(type)
int type;
```

DESCRIPTION

AVW_DataTypeToBands() returns the number of bands in the specified data type.

Type is used to specify the AVW data type, acceptable values are:
AVW_UNSIGNED_CHAR, AVW_SIGNED_CHAR, AVW_UNSIGNED_SHORT, AVW_SIGNED_SHORT, AVW_UNSIGNED_INT, AVW_SIGNED_INT, AVW_FLOAT, AVW_COMPLEX, AVW_COLOR.

RETURN VALUES

If successful *AVW_DataTypeToBands()* returns 1 for all data types except *AVW_COLOR*, for which it returns 3. *AVW_FAIL* is returned to indicate a illegal input data type.

SEE ALSO

AVW_DataTypeToBytes()

NAME

AVW_DataTypeToBytes – computes the number of bytes for a voxel

SYNOPSIS

```
#include "AVW.h"

int AVW_DataTypeToBytes(type)
int type;
```

DESCRIPTION

AVW_DataTypeToBytes() returns the number of bytes needed for one voxel of any AVW data type. *Type* is used to specify the AVW data type, acceptable values are: *AVW_UNSIGNED_CHAR*, *AVW_SIGNED_CHAR*, *AVW_UNSIGNED_SHORT*, *AVW_SIGNED_SHORT*, *AVW_UNSIGNED_INT*, *AVW_SIGNED_INT*, *AVW_FLOAT*, *AVW_COMPLEX*, *AVW_COLOR*.

RETURN VALUES

If successful *AVW_DataTypeToBytes()* returns the number of bytes required for one voxel. On failure *AVW_FAIL (0)* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

The return from this function should be checked, as the meaning of the return value for an unrecognized *type* is not that it requires zero bytes storage.

ERRORS

AVW_DataTypeToBytes() will fail if:

ILLDT
Illegal data type.

SEE ALSO

AVW_DataTypeToBands()

NAME

AVW_DecompressWaveletBuffer – returns an AVW_Image from a wavelet compressed data buffer.

SYNOPSIS

#include "AVW.h"

AVW_Image *AVW_DecompressWaveletBuffer(buff)
unsigned char *buff;

DESCRIPTION

AVW_DecompressWaveletBuffer() decompresses buffer produced by *AVW_WaveletCompressImage.* All the required information concerning the images data type and dimensions are contained in the buffer.

Buff is a memory buffer returned by the function *AVW_WaveletCompressImage().*

RETURN VALUES

If successful *AVW_DecompressWaveletBuffer()* returns a pointer to an *AVW_Image.* On failure it returns *AVW_NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DecompressWaveletBuffer will fail if the buffer is not from *AVW_WaveletCompressImage.*

BADMAL
Malloc Failed. A memory allocation failed.

ILLPAR
Illegal parameter. Buffer is not from AVW_WaveletCompressImage().

SEE ALSO

AVW_WaveletCompressImage() AVW_WaveletCompressImageFile() AVW_WaveletCompressAndDecompressImage(),

REFERENCE

Manduca, A. (1997) "Compressing Images with Wavelet / Subband Coding", IEEE Engineering in Medicine and Biology, 14(5), 639-646.

NAME

AVW_DeconvDivideImage – divides a spectrum by a transfer function

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_DeconvDivideImage(spectrum, transfer_func, fmin, out_image)
AVW_Image *spectrum;
AVW_Image *transfer_func;
double fmin;
AVW_Image *out_image;
```

DESCRIPTION

AVW_DeconvDivideImage() divides *spectrum*, which must be of data type *AVW_COMPLEX*, by *transfer_func*, which must be of data type *AVW_FLOAT* or *AVW_COMPLEX*. This routine can be used to carry out a simple deconvolution, where *spectrum* is the Fourier transform of an image and *transfer_func* is the Fourier transform of a point spread function. The result, returned in *out_image*, will be of data type *AVW_COMPLEX* and will be the Fourier transfom of the deconvolved image.

The width of the *spectrum* must be a power of 2 plus 1 and the height must be a power of 2 (since it is the spectrum of an image whose dimensions are all a power of 2 - See *AVW_FFT2D()*).

If *transfer_func* is of data type *AVW_FLOAT*, each item in *transfer_func* is checked and, if its absolute value is less than *fmin*, the corresponding item in *spectrum* is divided by *fmin* rather than by the item in *transfer_func*.

If *transfer_func* is of data type *AVW_COMPLEX*, each item in *transfer_func* is checked and, if its magnitude is less than *fmin*, it is scaled to a magnitude of *fmin* before the division. If the item is (0,0), the division is performed by (*fmin*, 0). The values in *transfer_func* are not actually altered by this checking.

Fmin must be a positive float number and will be ignored and the value .0001 used in its place if it is less than or equal to zero.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_DeconvDivideImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DeconvDivideImage() will fail if the following is true:

ILLDT
: Illegal data type. Spectrum is not AVW_COMPLEX or transfer_func is not AVW_COMPLEX or AVW_FLOAT.

CFLSZ
: Spectrum and transfer_func are not the same size.

BDSPCT
: Width of spectrum is not a power of 2 plus 1 or height is not a power of two.

SEE ALSO *AVW_DeconvDivideVolume(), AVW_DeconvWienerImage(), AVW_CreateStoksethMTF(), AVW_FFT2D(), AVW_ImageOpImage(), AVW_IterDeconvImage(), AVW_NearestNeighborDeconv(), AVW_Image*

NAME

AVW_DeconvDivideVolume – divides a spectrum by a transfer function

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_DeconvDivideVolume(spectrum, transfer_func, fmin, out_volume)
AVW_Volume *spectrum;
AVW_Volume *transfer_func;
double fmin;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_DeconvDivideVolume() divides *spectrum,* which must be of data type *AVW_COMPLEX,* by *transfer_func,* which must be of data type *AVW_FLOAT* or *AVW_COMPLEX*. This routine can be used to carry out a simple deconvolution, where *spectrum* is the Fourier transform of a volume and *transfer_func* is the Fourier transform of a point spread function. The result, returned in *out_volume,* will be of data type *AVW_COMPLEX* and will be the Fourier transfom of the deconvolved volume.

The width of the *spectrum* must be a power of 2 plus 1 and the height and depth must be a power of 2 (since it is the spectrum of a volume whose dimensions are all a power of 2 - See *AVW_FFT3D()*).

If *transfer_func* is of data type *AVW_FLOAT,* each item in *transfer_func* is checked and, if its absolute value is less than *fmin,* the corresponding item in *spectrum* is divided by *fmin* rather than by the item in *transfer_func.*

If *transfer_func* is of data type *AVW_COMPLEX,* each item in *transfer_func* is checked and, if its magnitude is less than *fmin,* it is scaled to a magnitude of *fmin* before the division. If the item is (0,0), the division is performed by (*fmin,* 0). The values in *transfer_func* are not actually altered by this checking.

Fmin must be a positive float number and will be ignored and the value .0001 used in its place if it is less than or equal to zero.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_DeconvDivideVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DeconvDivideVolume() will fail if the following is true:

ILLDT
: Illegal data type. Spectrum is not AVW_COMPLEX or transfer_func is not AVW_COMPLEX or AVW_FLOAT.

CFLSZ
: Spectrum and transfer_func are not the same size.

BDSPCT
: Width of spectrum is not a power of 2 plus 1 or, height or width is not a power of two.

SEE ALSO *AVW_DeconvDivideImage(), AVW_DeconvWienerVolume(), AVW_FFT3D(), AVW_IterDeconvVolume(), AVW_VolumeOpVolume(), AVW_Volume*

NAME

AVW_DeconvWienerImage – performs a Wiener-filtering operation

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_DeconvWienerImage(spectrum, transfer_func, alpha, out_image)
AVW_Image *spectrum;
AVW_Image *transfer_func;
double alpha;
AVW_Image *out_image;
```

DESCRIPTION

AVW_DeconvWienerImage() performs a Wiener-filtering operation on *spectrum*, which must be of data type *AVW_COMPLEX*, with the filter *transfer_func*, which must be of data type *AVW_FLOAT* or *AVW_COMPLEX*.

This routine can be used to carry out Wiener deconvolution, where *spectrum* is the Fourier transform of an image and *transfer_func* is the Fourier transform of a point spread function. The result, returned in *out_image*, will be of data type *AVW_COMPLEX* and will be the Fourier transform of the deconvolved image.

The width of the *spectrum* must be a power of 2 plus 1 and the height must be a power of 2 (since it is the spectrum of an image whose dimensions are all a power of 2 - See *AVW_FFT2D()*).

If *transfer_func* is of data type *AVW_Float*, each item in *spectrum* is multiplied by the quantity:

f / (f**2 + alpha),

where f is the value of the corresponding item in *transfer_func*.

If *transfer_func* is of data type *AVW_Complex*, each item in *spectrum* is multiplied by the quantity

f’ / (|f|**2 + alpha)

where f is the value of the corresponding item in *transfer_func*, f’ is the complex conjugate of that item, and |f|**2 is its magnitude squared.

Alpha must be a positive float number and will be ignored and the value .0001 used in its place if it is less than or equal to zero.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer’s Guide.)*

RETURN VALUES

If successful *AVW_DeconvWienerImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DeconvWienerImage() will fail if the following is true:

ILLDT

Illegal data type. Spectrum is not AVW_COMPLEX or transfer_func is not AVW_COMPLEX or AVW_FLOAT.

CFLSZ
Spectrum and transfer_func are not the same size.

BDSPCT
Width of spectrum is not a power of 2 plus 1 or height is not a power of two.

SEE ALSO *AVW_DeconvWienerVolume(), AVW_DeconvDivideImage(), AVW_CreateStoksethMTF(), AVW_FFT2D(), AVW_ImageOpImage(), AVW_IterDeconvImage(), AVW_NearestNeighborDeconv(), AVW_Image*

NAME

AVW_DeconvWienerVolume – performs a Wiener-filtering operation

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_DeconvWienerVolume(spectrum, transfer_func, alpha, out_volume)
AVW_Volume *spectrum;
AVW_Volume *transfer_func;
double alpha;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_DeconvWienerVolume() performs a Wiener-filtering operation on *spectrum,* which must be of data type *AVW_COMPLEX,* with the filter *transfer_func,* which must be of data type *AVW_FLOAT* or *AVW_COMPLEX*.

This routine can be used to carry out Wiener deconvolution, where *spectrum* is the Fourier transform of a volume and *transfer_func* is the Fourier transform of a point spread function. The result, returned in *out_volume,* will be of data type *AVW_COMPLEX* and will be the Fourier transform of the deconvolved volume.

The width of the *spectrum* must be a power of 2 plus 1 and the height and depth must be a power of 2 (since it is the spectrum of a volume whose dimensions are all a power of 2 - See *AVW_FFT3D()*).

If *transfer_func* is of data type *AVW_Float,* each item in *spectrum* is multiplied by the quantity:

f / (f**2 + alpha),

where f is the value of the corresponding item in *transfer_func.*

If *transfer_func* is of data type *AVW_Complex,* each item in *spectrum* is multiplied by the quantity

f’ / (|f|**2 + alpha)

where f is the value of the corresponding item in *transfer_func,* f’ is the complex conjugate of that item, and |f|**2 is its magnitude squared.

Alpha must be a positive float number and will be ignored and the value .0001 used in its place if it is less than or equal to zero.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer’s Guide.)*

RETURN VALUES

If successful *AVW_DeconvWienerVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DeconvWienerVolume() will fail if the following is true:

ILLDT

Illegal data type. Spectrum is not AVW_COMPLEX or transfer_func is not AVW_COMPLEX or AVW_FLOAT.

CFLSZ
Spectrum and transfer_func are not the same size.

BDSPCT
Width of spectrum is not a power of 2 plus 1 or, height or depth is not a power of two.

SEE ALSO *AVW_DeconvWienerImage(), AVW_DeconvDivideVolume(), AVW_FFT3D(), AVW_IterDeconvVolume(), AVW_VolumeOpVolume(), AVW_Volume()*

NAME

AVW_DefineConnected – defines 3D connected regions

SYNOPSIS

```
#include       "AVW_ObjectMap.h"

int AVW_DefineConnected(in_volume, object_map, seeds, connectivity,
                min, max, defined_object, count)
AVW_Volume *in_volume;
AVW_ObjectMap *object_map;
AVW_PointList3 *seeds;
int connectivity;
double min, max;
int defined_object;
int *count;
```

DESCRIPTION

AVW_DefineConnected() is used to define a 3D connected region from an *AVW_Volume* and *AVW_ObjectMap.* An *AVW_ObjectMap* and threshold limits are used to constrain the 3D region grow. A *count* of defined voxels is returned. Only voxels contained in objects which are turned "on" in the object map will be connected. (See *AVW_Object)*

In_volume and *object_map* specify the *AVW_Volume* and *AVW_ObjectMap* which contain the region to be defined.

Seeds is a pointer to an *AVW_PointList3* structure. This structure must contain at least one point which will be used as the starting point(s) for the 3D region grow.

Min and *max* specifies the threshold limits used to constrain the 3D region grow.

Defined_object specifies the object to which the connected voxels will be assigned.

Count contains the number of voxels defined if the function returns successfully. If the function fails count will contain the number at the time of failure.

RETURN VALUES

If successful *AVW_DefineConnected()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and set the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DefineConnected() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_ConnectAndDeleteVolume(), AVW_ConnectAndKeepVolume(), AVW_FindVolumeComponents(), AVW_ObjectMap, AVW_Object, AVW_Volume, AVW_PointList3

NAME

AVW_DefineUnconnected – defines 3D connected regions

SYNOPSIS

```
#include "AVW_ObjectMap.h"

int AVW_DefineUnconnected(in_volume, object_map, seeds, connectivity,
        min, max, defined_object, count)
AVW_Volume *in_volume;
AVW_ObjectMap *object_map;
AVW_PointList3 *seeds;
int connectivity;
double min, max;
int defined_object;
int *count;
```

DESCRIPTION

AVW_DefineUnconnected() is used to define a 3D connected region from an *AVW_Volume* and *AVW_ObjectMap.*

All voxels which within the threhold range and are turned "on" in the object map, and are NOT connected to the seeds, are defined.

An *AVW_ObjectMap* and threshold limits are used to constrain the 3D region grow. A *count* of defined voxels is returned. Only voxels contained in objects which are turned "on" in the object map will be connected. (See *AVW_Object)*

In_volume and *object_map* specify the *AVW_Volume* and *AVW_ObjectMap* which contain the region to be defined.

Seeds is a pointer to an *AVW_PointList3* structure. This structure must contain at least one point which will be used as the starting point(s) for the 3D region grow.

Min and *max* specifies the threshold limits used to constrain the 3D region grow.

Defined_object specifies the object to which the connected voxels will be assigned.

Count contains the number of voxels defined if the function returns successfully. If the function fails count will contain the number at the time of failure.

RETURN VALUES

If successful *AVW_DefineUnconnected()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and set the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DefineUnconnected() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_DefineConnected(), AVW_ConnectAndDeleteVolume(), AVW_ConnectAndKeepVolume(), AVW_FindVolumeComponents(), AVW_ObjectMap, AVW_Object, AVW_Volume, AVW_PointList3

NAME

AVW_DeleteObject – deletes an object from an object map

SYNOPSIS

```
#include "AVW_ObjectMap.h"

int AVW_DeleteObject(object_map, object)
AVW_ObjectMap *object_map;
int object;
```

DESCRIPTION

AVW_DeleteObject() removes the *AVW_Object* specified by *object* from *object_map*. Each *AVW_Object* with a value greater than *object* will be reduced by 1.

RETURN VALUES

If successful *AVW_DeleteObject()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DeleteObject() will fail if the following is true:

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_DestroyObjectMap(), AVW_CreateObjectMap(), AVW_GetObject(), AVW_PutObject() AVW_LoadObjectMap(), AVW_SaveObjectMap(), AVW_RemoveUnusedObjects(), AVW_RenderVolume(), AVW_ObjectMap, AVW_Object

NAME

AVW_DestroyCoeffs – frees filter coefficient structure memory

SYNOPSIS

```
#include "AVW_Filter.h"

void AVW_DestroyCoeffs(coeffs)
AVW_FilterCoeffs *coeffs;
```

DESCRIPTION

AVW_DestroyCoeffs() destroys a list of filter coefficients and frees the memory associated with it.

Note that *AVW_DestroyCoeffs()* does not set the passed pointer to *NULL*. It is important that this pointer be set to *NULL* before it is used again. For example:

```
AVW_DestroyCoeffs(coeffs);
coeffs = NULL;
```

Alternatively, the macro *AVW_DESTROYCOEFFS()* defined in *AVW_Filter.h* which automatically sets the pointer to *NULL* after freeing the memory may be used.

SEE ALSO

AVW_CreateCoeffs(), AVW_CreateButterworthCoeffs(), AVW_CreateGaussianCoeffs(), AVW_DESTROYCOEFFS(), AVW_FilterCoeffs,

NAME

AVW_DestroyColormap – frees colormap structure memory

SYNOPSIS

```
#include    "AVW.h"

void AVW_DestroyColormap(colormap)
AVW_Colormap *colormap;
```

DESCRIPTION

AVW_DestroyColormap() deallocates the memory asssociated with an *AVW_Colormap.*

An *AVW_Colormap* is a member of the *AVW_Image*, *AVW_Volume* and *AVW_ImageFile* structures and defines how intensity values are mapped to colors.

Note that *AVW_DestroyColormap()* does not set the passed pointer to *NULL*. Since the *AVW_Colormap* will usually be part of another *AVW* structure it is important to set this pointer to *NULL* after destroying it. For example:

```
AVW_DestroyColormap(img->Colormap);
img->Colormap = NULL;
```

Alternatively, the macro *AVW_DESTROYCOLORMAP()* defined in *AVW.h* which automatically sets the pointer to *NULL* after freeing the memory may be used.

SEE ALSO

AVW_CreateColormap(), AVW_CopyColormap(), AVW_DESTROYCOLORMAP(), AVW_Colormap

NAME

AVW_DestroyCompositeInfo – frees composite info structure memory

SYNOPSIS

```
#include "AVW_CompositeInfo.h"

void AVW_DestroyCompositeInfo(cinfo)
AVW_CompositeInfo *cinfo;
```

DESCRIPTION

AVW_DestroyCompositeInfo() frees all memory associated with *cinfo.*

Cinfo must be set to *NULL* after calling this function before it is used again.

SEE ALSO

AVW_CreateCompositeInfo(), AVW_LoadCompositeInfo(), AVW_SaveCompositeInfo(),

NAME

AVW_DestroyContourSurface – frees ContourSurface structure memory

SYNOPSIS

```
#include "AVW_Model.h"

int AVW_DestroyContourSurface(contourSurface)
AVW_ContourSurface *contourSurface;
```

DESCRIPTION

AVW_ContourSurface() destroys the memory allocated to the elements in *contourSurface.*

RETURN VALUES

AVW_ContourSurface() returns *AVW_SUCCESS* if successful. On failure, *AVW_FAIL* will be returned and I AVW_ErrorNumber and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure.

SEE ALSO

AVW_SliceVolume(), AVW_SaveContourSurface(), AVW_ContourSurface

NAME

AVW_DestroyFPointList2 – frees list structure memory

SYNOPSIS

#include "AVW.h"

void AVW_DestroyFPointList2(trace)
AVW_FPointList2 *trace;

DESCRIPTION

AVW_DestroyFPointList2() frees memory associated with an *AVW_FPointList2* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Trace must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYFPOINTLIST2(trace)*, defined in *AVW.h* may be used to free the memory and set the pointer to *NULL*.

SEE ALSO

AVW_DestroyFPointList3(), AVW_DestroyIPointList2(), AVW_DestroyIPointList3(), AVW_DestroyPointList2(), AVW_DestroyPointList3(), AVW_DestroyPointValueList(), AVW_CreateFPointList2(), AVW_AddFPoint2(), AVW_AddFPoint3(), AVW_AddPoint2(), AVW_AddPointValue(), AVW_DESTROYFPOINTLIST2(), AVW_FPointList2

NAME

AVW_DestroyFPointList3 – frees list structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyFPointList3(trace)
AVW_FPointList3 *trace;
```

DESCRIPTION

AVW_DestroyFPointList3() frees memory associated with an *AVW_FPointList3* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Trace must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYFPOINTLIST3(trace)*, defined in *AVW.h* may be used to free the memory and set the pointer to *NULL*.

SEE ALSO

AVW_DestroyFPointList2(), AVW_DestroyIPointList2(), AVW_DestroyIPointList3(), AVW_DestroyPointList2(), AVW_DestroyPointList3(), AVW_DestroyPointValueList(), AVW_CreateFPointList3(), AVW_AddFPoint3(), AVW_DESTROYFPOINTLIST3(), AVW_FPointList3

NAME

AVW_DestroyGradients – frees gradient structure memory

SYNOPSIS

```
#include "AVW_Render.h"

void AVW_DestroyGradients(surface)
AVW_Gradients *surface;
```

DESCRIPTION

AVW_DestroyGradients() is called to free *surface* and all its associated memory.

surface must be set to NULL after calling this function before it is used again.

SEE ALSO

AVW_ExtractGradients(), AVW_RenderGradients(), AVW_Gradients

NAME

AVW_DestroyHistogram – frees histogram structure memory

SYNOPSIS

```
#include "AVW_Histogram.h"

void AVW_DestroyHistogram(histo)
AVW_Histogram *histo;
```

DESCRIPTION

AVW_DestroyHistogram() is called to free *histo* and all its associated memory.

Histo must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYHISTOGRAM(histo),* defined in *AVW_Histogram.h* may be used to free the memory and set the pointer to *NULL.*

This function will also free the *Mem* element of the *AVW_Histogram.* If you do not wish to free the memory set the *Mem* element to *NULL* before calling the function:

```
...
pointer = histo->Mem;
histo->Mem = NULL;
AVW_DestroyHistogram(histo);
...
```

Note that *AVW_DestroyHistogram()* does not set the passed pointer to *NULL.* It is important to set this pointer to *NULL* after destroying it before using it again. For example:

```
AVW_DestroyHistogram(histo);
histo = NULL;
```

Alternatively, the macro *AVW_DESTROYHISTOGRAM()* defined in *AVW_Histogram.h* which automatically sets the pointer to *NULL* after freeing the memory and structure may be used.

SEE ALSO

AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_FlattenVolumeHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_MatchImageHistogram(), AVW_MatchVolumeHistogram(), AVW_NormalizeHistogram(), AVW_PreserveImageHistogram(), AVW_PreserveVolumeHistogram(), AVW_VerifyHistogram(), AVW_DESTROYHISTOGRAM(), AVW_Histogram

NAME

AVW_DestroyIPointList2 – frees list structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyIPointList2(trace)
AVW_IPointList2 *trace;
```

DESCRIPTION

AVW_DestroyIPointList2() frees memory associated with an *AVW_IPointList2* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Trace must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYIPOINTLIST2(trace),* defined in *AVW.h* may be used to free the memory and set the pointer to *NULL.*

SEE ALSO

AVW_DestroyIPointList3(), AVW_DestroyFPointList2(), AVW_DestroyFPointList3(), AVW_DestroyPointList2(), AVW_DestroyPointList3(), AVW_DestroyPointValueList(), AVW_CreateIPointList2(), AVW_AddIPoint2(), AVW_DESTROYIPOINTLIST2(), AVW_IPointList2

NAME

AVW_DestroyIPointList3 – frees list structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyIPointList3(trace)
AVW_IPointList3 *trace;
```

DESCRIPTION

AVW_DestroyIPointList3() frees memory associated with an *AVW_IPointList3* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Trace must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYIPOINTLIST3(trace),* defined in *AVW.h* may be used to free the memory and set the pointer to *NULL.*

SEE ALSO

AVW_DestroyIPointList2(), AVW_DestroyFPointList2(), AVW_DestroyFPointList3(), AVW_DestroyPointList2(), AVW_DestroyPointList3(), AVW_DestroyPointValueList(), AVW_CreateIPointList3(), AVW_AddIPoint3(), AVW_DESTROYIPOINTLIST3(), AVW_IPointList3

NAME

AVW_DestroyImage – frees image structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyImage(image)
AVW_Image *image;
```

DESCRIPTION

AVW_DestroyImage() is called to free *image* and all its associated memory.

Image must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYIMAGE(image)*, defined in *AVW.h* may be used to free the memory and set the pointer to *NULL*.

These function will also free the *Mem* element of the *AVW_Image*. If you do not wish to free *Mem*, set the *Mem* element to *NULL* before calling the functions.

```
pointer = image->Mem;
image->Mem = NULL;
AVW_DestroyImage(image);
```

SEE ALSO

AVW_DestroyVolume(), AVW_CreateImage(), AVW_CreateVolume(), AVW_DESTROYIMAGE(), AVW_Image

NAME

AVW_DestroyInstructions – frees instructions structure memory

SYNOPSIS

#include "AVW_Parse.h"

void AVW_DestroyInstructions(instructions)
AVW_Instructions *instructions;

DESCRIPTION

AVW_DestroyInstructions() frees all memory associated with *instructions.*

Instructions must be set to *NULL* after calling this function before it is used again.

SEE ALSO

AVW_Parse(), AVW_DoInstructions(), AVW_Instructions

NAME

AVW_DestroyList –frees list structure memory

SYNOPSIS

```
#include       "AVW.h"

void AVW_DestroyList(list)
AVW_List *list;
```

DESCRIPTION

AVW_DestroyList() frees memory used for an *AVW_List*.

List must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYLIST(list)*, defined in *AVW.h* may be used to free the memory and set the pointer to *NULL*.

SEE ALSO

AVW_ListFormats(), *AVW_ListInfo()*, *AVW_DESTROYLIST()*, *AVW_List*

NAME

AVW_DestroyMatchVoxelParams – frees the *AVW_MatchVoxelParams* structure memory

SYNOPSIS

#include "AVW_MatchVoxels.h"

void AVW_DestroyMatchVoxelParams(param)
AVW_MatchVoxelParams *param;

DESCRIPTION

AVW_DestroyMatchVoxelParams() destroys the memory allocated to the match voxel parameters given in *param*.

SEE ALSO

AVW_MatchVoxels(), AVW_MatchVoxelsParams, AVW_InitializeMatchVoxelParams().

NAME

AVW_DestroyMatrix – frees matrix structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyMatrix(matrix)
AVW_Matrix *matrix;
```

DESCRIPTION

AVW_DestroyMatrix() is called to free *matrix* and all its associated memory.

Matrix must be set to *NULL* after calling this function before it is used again.

SEE ALSO

AVW_CreateMatrix(), AVW_Matrix

NAME

AVW_DestroyMergedMap – frees merged info structure memory

SYNOPSIS

```
#include "AVW_MergedMap.h"

void AVW_DestroyMergedMap(merged_map)
AVW_MergedMap *merged_map;
```

DESCRIPTION

AVW_DestroyMergedMap() frees all memory associated with *merged_map.*

Merged_info must be set to *NULL* after calling this function before it is used again.

SEE ALSO

AVW_MergeRendered(), AVW_MergedMap

NAME

AVW_DestroyMultiList2 – frees multilist structure memory

SYNOPSIS

#include "AVW.h"

void AVW_DestroyMultiList2(mlst)
AVW_MultiList2 *mlst;

DESCRIPTION

AVW_DestroyMultiList2() frees memory associated with an *AVW_MultiList2* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Mlst must be set to *NULL* after calling this function before it is used again.
AVW_DestroyPointList2(), AVW_MultiList2, AVW_PointList2

NAME

AVW_DestroyObjectMap – frees object map structure memory

SYNOPSIS

```
#include "AVW_ObjectMap.h"

void AVW_DestroyObjectMap(object_map)
AVW_ObjectMap *object_map;
```

DESCRIPTION

AVW_DestroyObjectMap() frees all memory associated with *object_map*.

Object_map must be set to *NULL* after calling this function before it is used again.

SEE ALSO

AVW_AddObject(), AVW_DeleteObject(), AVW_CreateObjectMap(), AVW_LoadObjectMap(), AVW_SaveObjectMap(), AVW_RemoveUnusedObjects(), AVW_RenderVolume(), AVW_ObjectMap

NAME

AVW_DestroyPointList2 – frees list structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyPointList2(trace)
AVW_PointList2 *trace;
```

DESCRIPTION

AVW_DestroyPointList2() frees memory associated with an *AVW_PointList2* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Trace must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYPOINTLIST2(trace)*, defined in *AVW.h* may be used to free the memory and set the pointer to *NULL*.

SEE ALSO

AVW_DestroyFPointList2(), AVW_DestroyFPointList3(), AVW_DestroyIPointList2(), AVW_DestroyIPointList3(), AVW_DestroyPointList3(), AVW_DestroyPointValueList(), AVW_CreatePointList2(), AVW_AddPoint2(), AVW_DESTROYPOINTLIST2(), AVW_PointList3, AVW_Point3

NAME

AVW_DestroyPointList3 – frees list structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyPointList3(trace)
AVW_PointList3 *trace;
```

DESCRIPTION

AVW_DestroyPointList3() frees memory associated with an *AVW_PointList3* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Trace must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYPOINTLIST3(trace)*, defined in *AVW.h* may be used to free the memory and set the pointer to *NULL*.

SEE ALSO

AVW_DestroyFPointList2(), AVW_DestroyFPointList3(), AVW_DestroyIPointList2(), AVW_DestroyIPointList3(), AVW_DestroyPointList2(), AVW_DestroyPointValueList(), AVW_CreatePointList3(), AVW_AddPoint3(), AVW_DESTROYPOINTLIST3(), AVW_PointList3, AVW_Point3

NAME

AVW_DestroyPointValueList – frees list structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyPointValueList(trace)
AVW_PointValueList *trace;
```

DESCRIPTION

AVW_DestroyPointValueList() frees memory associated with an *AVW_PointValueList* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Trace must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYPOINTVALUELIST(trace),* defined in *AVW.h* may be used to free the memory and set the pointer to *NULL.*

SEE ALSO

AVW_DestroyFPointList2(), AVW_DestroyFPointList3(), AVW_DestroyIPointList2(), AVW_DestroyIPointList3(), AVW_DestroyPointList2(), AVW_DestroyPointList3(), AVW_CreatePointValueList(), AVW_AddPointValue(), AVW_DESTROYPOINTVALUELIST(), AVW_PointValueList, AVW_Point

NAME

AVW_DestroyRPParam – frees tile parameter structure memory

SYNOPSIS

```
#include "AVW_Model.h"

int AVW_DestroyRPParam(rp_param)
AVW_RPParam *rp_param;
```

DESCRIPTION

AVW_DestroyRPParam() destroys the memory allocated to the tiling parameters given in *tile_param*.

RETURN VALUES

AVW_DestroyRPParam() returns *AVW_SUCCESS* if successful. On failure, *AVW_FAIL* will be returned and I AVW_ErrorNumber and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure.

SEE ALSO

AVW_SliceVolume(), AVW_InitializeRPParam(), AVW_RPParam

NAME

AVW_DestroyRenderParameters – frees rendering structure memory

SYNOPSIS

```
#include "AVW_Render.h"

void AVW_DestroyRenderParameters(render_param)
AVW_RenderParameters *render_param;
```

DESCRIPTION

AVW_DestroyRenderParameters() is called to free *render_param* and all its associated memory.

Render_param must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYRENDERPARAMETERS(render_param),* defined in *AVW.h* may be used to free the memory and set the pointer to *NULL.*

SEE ALSO

AVW_DestroyRenderedImage(), AVW_InitializeRenderParameters(), AVW_RenderVolume(), AVW_DESTROYRENDERPARAMETERS(), AVW_RenderParameters

NAME

AVW_DestroyRenderedImage – frees rendered image structure memory

SYNOPSIS

```
#include       "AVW_Render.h"

void AVW_DestroyRenderedImage(image)
AVW_RenderedImage *rendered_image;
```

DESCRIPTION

AVW_DestroyRenderedImage() is called to free *rendered_image* and all its associated memory.

Rendered_image must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYRENDEREDIMAGE(rendered_image),* defined in *AVW_Render.h* may be used to free the memory and set the pointer to *NULL.*

SEE ALSO

AVW_DestroyRenderParameters(), AVW_InitializeRenderParameters(), AVW_RenderVolume(), AVW_DESTROYRENDEREDIMAGE(), AVW_RenderedImage

NAME

AVW_DestroyTileParameters – frees tile parameter structure memory

SYNOPSIS

#include "AVW_Model.h"

int AVW_DestroyTileParameters(tile_param)
AVW_TileParameters *tile_param;

DESCRIPTION

AVW_DestroyTileParameters() destroys the memory allocated to the tiling parameters given in *tile_param*.

RETURN VALUES

AVW_DestroyTileParameters() returns *AVW_SUCCESS* if successful. On failure, *AVW_FAIL* will be returned and I AVW_ErrorNumber and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure.

SEE ALSO

AVW_LoadTiledSurface(), AVW_SaveTiledSurface(), AVW_DrawTiledSurface(), AVW_TileVolume(), AVW_TiledSurface

NAME

AVW_DestroyTiledSurface – frees ordered surface structure memory

SYNOPSIS

```
#include "AVW_Model.h"

int AVW_DestroyTiledSurface(srfc)
AVW_TiledSurface *srfc;
```

DESCRIPTION

AVW_DestroyTiledSurface() destroys the memory allocated to the ordered surface given in *srfc*.

RETURN VALUES

AVW_DestroyTiledSurface() returns *AVW_SUCCESS* if successful. On failure, *AVW_FAIL* will be returned and I AVW_ErrorNumber and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure.

SEE ALSO

AVW_LoadTiledSurface(), AVW_SaveTiledSurface(), AVW_DrawTiledSurface(), AVW_TileVolume(), AVW_TiledSurface

NAME

AVW_DestroyTree – frees tree structure memory

SYNOPSIS

```
#include "AVW_Tree.h"

void AVW_DestroyTree(tree)
AVW_Tree *trace;
```

DESCRIPTION

AVW_DestroyTree() frees memory associated with an *AVW_Tree* structure and the structure itself.

The list structures are managed i.e., memory for the structures is reallocated as needed. These structures are defined in *AVW_Tree.h* and are used to store skeletal tree information.

Tree must be set to *NULL* after calling this function before it is used again.

SEE ALSO

AVW_AddTreeChild(), AVW_CreateTree(), AVW_FindTreeIndex(), AVW_LoadTree(), AVW_SaveTree(), AVW_TreePoint, AVW_Tree

NAME

AVW_DestroyVisibleSurface – frees visible surface structure memory

SYNOPSIS

```
#include "AVW_Render.h"

void AVW_DestroyVisibleSurface(surface)
AVW_VisibleSurface *surface;
```

DESCRIPTION

AVW_DestroyVisibleSurface() is called to free *surface* and all its associated memory.

surface must be set to NULL after calling this function before it is used again.

SEE ALSO

AVW_ExtractVisibleSurface(), AVW_RenderVisibleSurface(), AVW_ExtractSurface(), AVW_VisibleSurface

NAME

AVW_DestroyVolume – frees volume structure memory

SYNOPSIS

```
#include "AVW.h"

void AVW_DestroyVolume(volume)
AVW_Volume *volume;
```

DESCRIPTION

AVW_DestroyVolume() is called to free *volume* and all its associated memory.

Volume must be set to *NULL* after calling this function before it is used again. Alternatively, the macro *AVW_DESTROYVOLUME(volume),* defined in *AVW.h* may be used to free the memory and set the pointer to *NULL.*

These function will also free the *Mem* element of the *AVW_Image* or *AVW_Volume.* If you do not wish to free *Mem*, set the *Mem* element to *NULL* before calling the functions.

```
pointer = volume->Mem;
volume->Mem = NULL;
AVW_DestroyVolume(volume);
```

SEE ALSO

AVW_DestroyImage(), AVW_CreateVolume(), AVW_DESTROYVOLUME(), AVW_Volume

NAME

AVW_DilateImage– morphologically dilates an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_DilateImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_DilateImage() performs binary morphological dilation on *in_image* using *element* as the structuring element.

In_image does not have to be a binary valued but all nonzero voxels are treated as ones. *In_image*, *out_image*, and *element* must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

The dilation can be thought of as an enlargement of the binary objects contained in the data. In simplest terms the dilation process takes place by translating the structuring element so that its centerpoint lies on every point of the data. At each point in the data, if a nonzero pixel in the structuring element corresponds to a nonzero pixel in the data the point in the result data is set to one. Otherwise the data is unchanged. The result data will only contain ones and zeros.

AVW_CreateStructuringImage(), *AVW_CreateImage()*, *AVW_SetImage()*, and *AVW_PutPixel()* may be used to create a structuring element. The structuring element is flipped on the X and Y axes prior to dilation.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_DilateImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DilateImage() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLIMG
: An illegal image was passed to the function.

SEE ALSO

AVW_DilateVolume(), AVW_CreateImage(), AVW_MorphCloseImage(), AVW_ErodeImage(), AVW_MorphOpenImage(), AVW_ConditionalDilateImage(), AVW_MorphMaxImage(), AVW_MorphMinImage(), AVW_SetImage(), AVW_PutPixel(), AVW_CreateStructuringImage(), AVW_Image

NAME

AVW_DilateVolume – morphologically dilates a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_DilateVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_DilateVolume() performs binary morphological dilation on *in_volume* using *element* as the structuring element.

In_volume does not have to be a binary valued but all nonzero voxels are treated as ones. *In_volume, out_volume,* and *element* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

The dilation can be thought of as an enlargement of the binary objects contained in the data. In simplest terms the dilation process takes place by translating the structuring element so that its centerpoint lies on every point of the data. At each point in the data, if a nonzero voxel in the structuring element corresponds to a nonzero voxel in the data the point in the result data is set to one. Otherwise the data is unchanged. The result data will only contain ones and zeros.

AVW_CreateStructuringVolume(), AVW_CreateVolume(), AVW_SetVolume(), and *AVW_PutVoxel()* may be used to create a structuring element. The structuring element is flipped on the X, Y and Z axes prior to dilation and unflipped afterwards within the routine.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_DilateVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DilateVolume() will fail if:

BADMAL
Malloc failed. Unable to allocate memory for results.

ILLDT
Data type is not defined or supported.

ILLVOL
An illegal volume was passed to the function.

SEE ALSO

AVW_CreateStructuringVolume(), AVW_DilateImage(), AVW_CreateVolume(), AVW_MorphCloseVolume(), AVW_ErodeVolume(), AVW_MorphOpenVolume(), AVW_ConditionalDilateVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume(), AVW_SetVolume(), AVW_PutVoxel(), AVW_Volume

NAME

AVW_DisableImageFileFormat – disables I/O support for an image file format

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ImageFile.h"

int AVW_DisableImageFileFormat(fileformat)
char *fileformat;
```

DESCRIPTION

AVW_DisableImageFileFormat() disables support of a named image file format.

Fileformat is the name by which the file format is identified in AVW and is the Description element of the AVW_ExtendIO structure with which the format was extend.

The Properties element of the AVW_ExtendIO structure for this format is set to 0, indicating that no data types or operations are supported. The *AVW_ExtendIO* entry for this format remains in the list of extended file formats but is marked as disabled. This *fileformat* string identifier may not be used to identify another format with a user extended call to *AVW_ExtendImageFile().*

RETURN VALUES

AVW_DisableImageFileFormat() returns the value of the Properties element of the AVW_EntendIO structure which indicates the properties supported in the format.

This value can be used as an argument with *AVW_EnableImageFileFormat()* at a later time to re-enable the image file format.

SEE ALSO

AVW_ExtendExternalLibs(), AVW_EnableImageFileFormat(), AVW_CreateImageFile(), AVW_CloseImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume() AVW_ExtendIO

NAME

AVW_DisableProgress – temporarily disables progress

SYNOPSIS

#include "AVW.h"

void AVW_DisableProgress()

DESCRIPTION

AVW_DisableProgress() is called to temporarily disable all calls made using *AVW_Progress()* to the function specified by *AVW_ProgressFunction()*. This is normally done within functions which report progress, but call other *AVW* functions which report progress.

SEE ALSO

AVW_Progress(), AVW_ProgressFunction(), AVW_EnableProgress()

NAME

AVW_DitherImage – dithers an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_DitherImage(in_image, ncolors, out_image)
AVW_Image *in_image;
int ncolors;
AVW_Image *out_image;
```

DESCRIPTION

AVW_DitherImage() converts a 24-bit *AVW_Image*, *in_image*, to an 8-bit *AVW_Image* with an associated colormap. The Colormap is an element of the returned AVW_Image.

Ncolors is the number of colors to which *out_image* will be reduced.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_DitherImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DitherImage() will fail if:

ILLDT
Illegal Datatype. In_image must be of type AVW_COLOR.

SEE ALSO

AVW_DitherVolume(), AVW_ConvertImage(), AVW_MakeGrayImage(), AVW_MakeMonoImage(), AVW_Image, AVW_Colormap

NAME

AVW_DitherVolume – dithers a volume

SYNOPSIS

```
#include       "AVW.h"

AVW_Volume *AVW_DitherVolume(in_volume, ncolors, out_volume)
AVW_Volume *in_volume;
int ncolors;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_DitherVolume() converts a 24-bit *AVW_Volume, in_volume*, to an 8-bit *AVW_Volume* with an associated colormap. The conversion is done in 2D. The Colormap is and element of the return AVW_Volume.

Ncolors is the number of colors to which *out_volume* will be reduced.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_DitherVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DitherVolume() will fail if:

ILLDT
Illegal Datatype. In_volume must be of type AVW_COLOR.

SEE ALSO

AVW_ConvertVolume(), AVW_MakeGrayVolume(), AVW_Colormap

NAME

AVW_DoInstructions – evaluates an algebraic-style formula

SYNOPSIS

```
#include "AVW_Parse.h"

int AVW_DoInstructions(instructions)
AVW_Instructions *instructions;
```

DESCRIPTION

AVW_DoInstructions() executes the information stored in *instructions.*

RETURN VALUES

If successful, *AVW_DoInstructions()* returns *AVW_SUCCESS*. On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_DoInstructions() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

BADOP
Unknown Instruction.

SYNTAX
Syntax Error In Formula.

TCMPLX
Formula Too Complex To Be Parsed.

SEE ALSO

AVW_Parse(), AVW_DestroyInstructions(), AVW_Instructions

NAME

AVW_DrawFilledPointList2 – draws a filled trace into an image

SYNOPSIS

```
#include "AVW.h"

int AVW_DrawFilledPointList2(image, list, value)
AVW_Image *image;
AVW_PointList2 *list;
double value;
```

DESCRIPTION

AVW_DrawFilledPointList2() sets the locations in *image* specified by *list* to *value*. If *list* specifies a simple or complex closed trace, the interior of the trace is also set to *value*.

For *AVW_COLOR* images the red, green, and blue components need to be packed into *value*. See *AVW_MAKERGB()*.

RETURN VALUES

If successful *AVW_DrawFilledPointList2()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned.

ERRORS

AVW_DrawFilledPointList2() will fail if one or more of the following are true:

ILLPAR
A NULL image or list was specified.

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_DrawPointList2(), AVW_DrawPointList3(), AVW_PutVoxel(), AVW_GetPixel(), AVW_GetVoxel(), AVW_Image, AVW_Point2, AVW_MAKERGB()

NAME

AVW_DrawImageLine – draws a line into an image

SYNOPSIS

```
#include "AVW.h"

int AVW_DrawImageLine(image, pt1, pt2, value)
AVW_Image *image;
AVW_Point2 *pt1, *pt2;
double value;
```

DESCRIPTION

AVW_DrawImageLine() sets all pixels in *image* along a line from *pt1* to *pt2* to *value.*

RETURN VALUES

If successful *AVW_DrawImageLine()* returns *AVW_SUCCESS. AVW_FAIL* is returned if all points were outside the image.

SEE ALSO

AVW_DrawVolumeLine(), AVW_PutPixel(), AVW_PutVoxel(), AVW_Image, AVW_Point2

NAME

AVW_DrawImageText – draws a text string into an image

SYNOPSIS

```
#include "AVW.h"

int AVW_DrawImageText(image, string, pt, value)
AVW_Image *image;
char *string;
AVW_Point2 *pt;
double value;
```

DESCRIPTION

AVW_DrawImageText() draws the text specified by *string* into an *image* at the location *pt*. *Image* is an *AVW_Image*.

String is a zero terminated ascii string.

Pt specifies where the drawing begins. Text can only be drawn from left to right.

Value indicates the value that each pixel is set to.

A small crude font is all that is available.

RETURN VALUES

If successful *AVW_DrawImageText()* returns *AVW_SUCCESS*. *AVW_FAIL* is returned if starting position was outside the image.

SEE ALSO

AVW_DrawImageLine(), AVW_PutPixel(), AVW_PutVoxel(), AVW_Image, AVW_Point2

NAME

AVW_DrawPointList2 – puts a value at the locations in an image sepcified by a point list

SYNOPSIS

```
#include "AVW.h"

int AVW_DrawPointList2(image, list, value)
AVW_Image *image;
AVW_PointList2 *list;
double value;
```

DESCRIPTION

AVW_DrawPointList2() sets the locations in *image* specified by *list* to *value.*

For *AVW_COLOR* images the red, green, and blue components need to be packed into *value.* See *AVW_MAKERGB().*

RETURN VALUES

If successful *AVW_DrawPointList2()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned.

ERRORS

AVW_DrawPointList2() will fail if one or more of the following are true:

ILLPAR
A NULL image or list was specified.

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_DrawPointList3(), AVW_PutVoxel(), AVW_GetPixel(), AVW_GetVoxel(), AVW_Image, AVW_Point2, AVW_MAKERGB()

NAME

AVW_DrawPointList3 – puts a value at the locations in a volume specified by a point list

SYNOPSIS

```
#include "AVW.h"

int AVW_DrawPointList3(volume, list, value)
AVW_Volume *volume;
AVW_Point3 *list;
double value;
```

DESCRIPTION

AVW_DrawPointList3() sets the locations in *volume* specified by *list* to *value.*

For *AVW_COLOR* volumes the red, green, and blue components need to be packed into *value.* See *AVW_MAKERGB().*

RETURN VALUES

If successful *AVW_DrawPointList3()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned.

ERRORS

AVW_DrawPointList3() will fail if

ILLPAR
A NULL list or volume was specified.

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_DrawPointList2(), AVW_PutPixel(), AVW_GetVoxel(), AVW_Point3, AVW_Volume, AVW_MAKERGB()

NAME

AVW_DrawRenderedbackDrop – draws a line into rendered space

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_DrawRenderedBackDrop(rendered, value)
AVW_RenderedImage *rendered;
double value;
```

DESCRIPTION

AVW_DrawRenderedbackDrop() draws a shaded backdrop at each pixel of the rendering where *render->Depth* is contained in the *rendered->PBuffer* value for that pixel.

In other words where no output value was put into the rendering.

AVW_DrawRenderedPoint(), AVW_DrawRenderedLine(), AVW_RenderVolume(), AVW_RenderedImage,

NAME

AVW_DrawRenderedLine – draws a line into rendered space

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_DrawRenderedLine(rendered, start, end, value)
AVW_RenderedImage *rendered;
AVW_FPoint3 *start, *end;
double value;
```

DESCRIPTION

AVW_DrawRenderedLine() transforms the line, specified by end points *start* and *end*, to the rendered space and changes all pixels along the transformed line to *value*.

RETURN VALUES

AVW_DrawRenderedLine() returns *AVW_SUCCESS* if the entire line can be successfully transformed and draw into the rendered space. *AVW_FAIL* is returned if any point along the line could not be transformed.

SEE ALSO

AVW_DrawRenderedPoint(), AVW_FindRenderedPoint(), AVW_RenderVolume(), AVW_RenderedImage, AVW_FPoint3

NAME

AVW_DrawRenderedPoint – draws a point into rendered space

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_DrawRenderedPoint(rendered, in, value)
AVW_RenderedImage *rendered;
AVW_FPoint3 *in;
double value;
```

DESCRIPTION

AVW_DrawRenderedPoint() transforms the point *in* to the rendered space and changes the pixel at that location within the rendered image to *value.*

RETURN VALUES

AVW_DrawRenderedPoint() returns *AVW_SUCCESS* if *in* can be successfully transformed to a location within the rendered space. It returns *AVW_FAIL* if the point transforms to a location outside the rendered space.

SEE ALSO

AVW_DrawRenderedLine(), AVW_FindRenderedPoint(), AVW_RenderVolume(), AVW_RenderedImage, AVW_FPoint3

NAME

AVW_DrawVolumeLine – draws a line into a volume

SYNOPSIS

```
#include "AVW.h"

int AVW_DrawVolumeLine(volume, pt1, pt2, value)
AVW_Volume *volume;
AVW_Point3 *pt1, *pt2;
double value;
```

DESCRIPTION

AVW_DrawVolumeLine() sets all voxels in *volume* along a line from *pt1* to *pt2* to *value.*

RETURN VALUES

If successful *AVW_DrawVolumeLine()* returns *AVW_SUCCESS*. *AVW_FAIL* is returned if all points were outside the volume.

SEE ALSO

AVW_DrawImageLine(), AVW_PutPixel(), AVW_PutVoxel(), AVW_Volume, AVW_Point3

NAME

AVW_EditPointList2 – replaces a segment of a point list

SYNOPSIS

#include "AVW.h"

AVW_PointList2 *AVW_EditPointList2(orig_trace, new_segment, new_trace)
AVW_PointList2 *orig_trace, *new_segment, new_trace;

DESCRIPTION

AVW_EditPointList2() is used to edit an *AVW_PointList2* with another point list. *Orig_trace* contains a trace. *New_segment* contains another trace which is meant to replace a portion of *orig_trace*. The edited trace is returned in *new_trace*. The function determines the points in *orig_trace* which are closest to the first and last points of *new_segment*. The trace segment of *orig_trace* between these points is replaced with with the points of *new_segment*.

New_trace is provided as a method of reusing an existing *AVW_PointList2*.

RETURN VALUES

If successful *AVW_EditPointList2()* returns an *AVW_PointList2*. On failure it returns *AVW_NULL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_EditPointList2() will fail if:

BADMAL
Malloc Failed. Unable to allocate memory.

SEE ALSO

AVW_AddPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_FillPointList2(), AVW_PointList2, AVW_Point2

NAME

AVW_EnableImageFileFormat – re-enables I/O support for a previously disabled image file format

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ImageFile.h"

int AVW_EnableImageFileFormat(fileformat, properties)
char *fileformat;
int properties;
```

DESCRIPTION

AVW_EnableImageFileFormat() re-enables I/O support of a named image file format which was previously disabled by *AVW_DisableImageFileFormat()*

fileformat is the name by which the file format is identified in AVW and is the *Description* element of the *AVW_ExtendIO* structure with which the format was extend.

properties is the *Properties* element of the AVW_ExtendIO structure for this format. This is the value with which this format was originally extended by *AVW_ExtendImageIO()* and was returned by *AVW_DisableImageFileFormat()* when the format was disabled.

RETURN VALUES

AVW_EnableImageFileFormat() returns the value of the Properties element of the AVW_EntendIO structure for the format prior to the call.

SEE ALSO

AVW_ExtendExternalLibs(), AVW_DisableImageFileFormat(), AVW_CreateImageFile(), AVW_CloseImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume() AVW_ExtendIO

NAME AVW_EnableProgress – reinstates progress

SYNOPSIS #include "AVW.h"

void AVW_EnableProgress()

DESCRIPTION *AVW_EnableProgress()* is called to re-enable calls to the *AVW_ProgressFunction()*. It undoes the effect of *AVW_DisableProgress()*.

SEE ALSO *AVW_Progress(), AVW_ProgressFunction(), AVW_DisableProgress()*

NAME

AVW_ErodeImage – morphologically erodes an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ErodeImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ErodeImage() performs binary morphological erosion on *in_image* using *element* as the structuring element.

In_image does not have to be a binary valued but all nonzero pixels are treated as ones. *In_image*, *out_image*, and *element* must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

The erosion can be thought of as a shrinking of the binary objects contained in the data. In simplest terms the erosion process takes place by translating the structuring element so that its centerpoint lies on every point of the data. At each point in the data, if a nonzero pixel in the structuring element corresponds to a zero pixel in the data the point in the result data is set to zero. Otherwise the pixel is unchanged. The result data will only contain ones and zeros.

The structuring element, *element*, should always have odd dimensions, be symmetric, and be smaller than *in_image*. *AVW_CreateImage()*, *AVW_SetImage()*, and *AVW_PutPixel()* may be used to create a structuring element.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ErodeImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ErodeImage() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLIMG
: An illegal image was passed to the function.

SEE ALSO

AVW_ErodeVolume(), AVW_CreateImage(), AVW_MorphCloseImage(), AVW_DilateImage(), AVW_MorphOpenImage(), AVW_ConditionalDilateImage(), AVW_MorphMaxImage(), AVW_MorphMinImage(), AVW_UltimateErosionImage(), AVW_CreateStructuringImage(), AVW_Image

NAME

AVW_ErodeVolume – morphologically erodes a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ErodeVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ErodeVolume() performs binary morphological erosion on *in_volume* using *element* as the structuring element.

In_volume does not have to be a binary valued but all nonzero pixels are treated as ones. *In_volume, out_volume,* and *element* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

The erosion can be thought of as a shrinking of the binary objects contained in the data. In simplest terms the erosion process takes place by translating the structuring element so that its centerpoint lies on every point of the data. At each point in the data, if a nonzero voxel in the structuring element corresponds to a zero voxel in the data the point in the result data is set to zero. Otherwise the voxel is unchanged. The result data will only contain ones and zeros.

The structuring element, *element,* should always have odd dimensions, be symmetric, and be smaller than *in_volume. AVW_CreateVolume(), AVW_SetVolume(),* and *AVW_PutVoxel()* may be used to create a structuring element.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ErodeVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ErodeVolume() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLVOL
: An illegal volume was passed to the function.

SEE ALSO

AVW_ErodeImage(), AVW_MorphCloseVolume(), AVW_DilateVolume(), AVW_MorphOpenVolume(), AVW_ConditionalDilateVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume(), AVW_UltimateErosionVolume(), AVW_CreateStructuringVolume(), AVW_Volume

NAME

AVW_Error – writes the current AVW_ErrorMessage to stderr

SYNOPSIS

```
#include "AVW.h"

void AVW_Error(string)
char *string;
```

DESCRIPTION

AVW_Error() writes a formatted error message to stderr consisting of your own message (string) and the current *AVW_ErrorMessage.*

For example: if the current value of *AVW_ErrorMessage* is *Malloc Failed* , then "AVW_Error("Message");" will write *Message: Malloc Failed* to stderr.

SEE ALSO

AVW_GetErrorMessage(), AVW_GetErrorNumber(), AVW_SetError()

NAME

AVW_ExtendExternalLibs – loads and initializes shared libraries

SYNOPSIS

AVW_ExtendExternalLibs()

DESCRIPTION

AVW_ExtendExternalLibs() reads the contents of a configuration file called *EXTEND.conf* which lists the names of external shared libraries and their initialization routines. Standard locations are searched for these libraries, and if found, the libraries are loaded into the running program. The initialization routines are then called.

This function is designed to implement loading of image file extensions to AVW, to avoid the need to recompile each application if a new file format is added to the library. If the extensions are loaded into shared libraries and referenced in the the configuration file, this function will load and install them in a running program.

The configuration file EXTEND.conf is searched for in the user's app-defaults directory. If it is not found there $AVW / app-default is searched.

A file path specified by the environment variable AVW_EXTEND over rides over the previous files if they exist. If no configuration file is found, the function returns.

NOTE

AVW_ExtendExternalLibs() requires linking with the avwEXTEND library.

RETURN VALUES

None

ERRORS

If the requested libraries cannot be found, or if the initialization routine cannot be located in the library, the requested library will be skipped. If the configuration file is not found, the function returns silently.

SEE ALSO

AVW_ExtendImageFile(), EXTEND.conf

NAME

AVW_ExtendImageFile – extends IO routines to support additional formats

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ImageFile.h"

int AVW_ExtendImageFile(fl)
AVW_ExtendIO *fl;
```

DESCRIPTION

AVW_ExtendImageFile() extends the *AVW_ImageFile* library to support other data formats. In this way applications can be written using the *AVW* calls without concern for the data format of image files which are being processed. Applications needn't be rewritten to support additional data formats, merely extend the *AVW* library to support the data format. Each time *AVW_ExtendImageFile()* is invoked the external integer *avw_NumberOfSupportedFormats* is incremented. The argument passed to the *AVW_ExtendImageFile()* is an *AVW_ImageFileFunction*. The elements of this structure describe the attributes of this format, and format specific functions which correspond to and are called by *AVW_CreateImageFile(), AVW_OpenImageFile(), AVW_CloseImageFile(), AVW_ReadImageFile(), AVW_WriteImageFile(),* and *AVW_SeekImageFile().*

To extend the AVW_ImageFile routines you must set the following elements of an *AVW_ExtendIO* structure and pass the initialized structure to *AVW_ExtendImageFile.*

fl->Extension

is the ending (if any) by which the file type may be identified.

fl->Description

is the text name by which this data format is identified. This is the string which will appear for this format when AVW_ListFormats() is invoked. This is also the string used to identify what format of file is being created by AVW_CreateImageFile()

fl->MagicNumber

is the magic number (if any) by which the file type may be identified. NO_MAGIC_NUMBER is used to indicate that this file type is not identified via a magic number.

fl->Open()

is the function that AVW_OpenImageFile() will pass its arguments to and return its return value from. This function should do error checking and return a AVW_ImageFile structure. AVW_OpenImageFile() will return whatever this function returns.

fl->Seek()

is the function that AVW_SeekImageFile will pass its arguments to and return a value from. This function should do error checking and return AVW_SUCESS or AVW_FAIL. For 2D formats (various raster formats) this should be a routine that simply checks if the volume and slice arguments are both 0.

fl->Read()

is the function AVW_ReadImageFile() will pass its arguments to and return a value from. The value returned should be an AVW_Image or NULL.

fl->Write()

is the function AVW_WriteImageFile() will pass its arguments to and return a value from. The values returned should be AVW_SUCCESS or AVW_FAIL.

fl->Close()

is the function AVW_CloseImageFile() will pass its arguments to and return a value from. The values returned should be AVW_SUCCESS or AVW_FAIL.

fl->Create()

is the function that AVW_CreateImageFile will pass its arguments to and return a value from. The return value should be a pointer to an AVW_ImageFile allocated and initialized by the function.

fl->Query()

determines whether the named file is of this format. Returns TRUE or FALSE to indicate whether the name of the file passed to it is of this data format. AVW_OpenImageFile() will call this routine to see if the named file is of this data format.

fl->Properties

is a mask of attributes of this format which describes and controls which attributes are supported by the interface routines.

C Datatypes supported by this format:

AVW_SUPPORT_UNSIGNED_CHAR
AVW_SUPPORT_SIGNED_CHAR
AVW_SUPPORT_UNSIGNED_SHORT
AVW_SUPPORT_SIGNED_SHORT
AVW_SUPPORT_UNSIGNED_INT
AVW_SUPPORT_SIGNED_INT
AVW_SUPPORT_FLOAT
AVW_SUPPORT_COMPLEX
AVW_SUPPORT_COLOR

Other attributes supported by this format:

AVW_SUPPORT_2D

AVW_SUPPORT_3D
AVW_SUPPORT_READ
AVW_SUPPORT_WRITE

For example an attribute mask for a raster file type which supported only 8 bit images might look like:

```
(AVW_SUPPORT_UNSIGNED_CHAR|
 AVW_SUPPORT_2D|
 AVW_SUPPORT_READ|
 AVW_SUPPORT_WRITE)
```

The following file formats are directly supported by AVW Image File IO routines:

AnalyzeAVW
AVW_VolumeFile
AVW_Objectmap

Additional file formats are extended under the control of the configuration file EXTEND.conf. The first time an application makes a call to AVW_OpenImageFile(), the function AVW_ExtendExternalLibs() is invoked to dynamically load libraries for additional iage fil formats. Libraries are provided for these formats:

AnalyzeImage(7.5)
AnalyzeScreen
SunRaster
GE9800 (Read Only)
GESigna (Read Only)
GE Advantage (Read Only)
GESTARCAM (Read Only)
Imatron (Read Only)
SiemensCT (Read Only)
INTERFILE (Read Only)
ACRNEMA (Read Only)
PAPYRUS (Read Only)
SGIrgb
SGIbw
PPM
PGM
YUV
BMP
PIC
JPG
PICKERMRI (Read Only)
SMIS (Read Only)
CTI (Read Only)
TARGA
CTIECAT7
PostScript (Write Only)

Support for some of the formats is limited to Read Only. Files cannot be created or written in these formats.

RETURN VALUES

AVW_ExtendImageFile() returns the current number of supported formats. This is the value to which the *DataFormat* element within the *AVW_ImageFile* is set upon a succesful *AVW_OpenImageFile()*. It is also the argument which is used to direct *AVW_CreateImageFile()* the data format of the new file.

SEE ALSO

EXTEND.conf AVW_ExtendExternalLibs(), AVW_DisableImageFileFormat(), AVW_EnableImageFileFormat(), AVW_CreateImageFile(), AVW_CloseImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume() AVW_ExtendIO

NAME

AVW_ExtractControlPoints – extracts control points from a trace

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_ExtractControlPoints(trace, step, close_flag, distance, control)
AVW_PointList2 *trace;
double step;
int close_flag;
int distance;
AVW_PointList2 *control;
```

DESCRIPTION

AVW_ExtractControlPoints() extracts a set of control points from a *trace*.

Points are removed from *trace* and *AVW_MakeSpline()* is called using the *step* and *close_flag* parameters. If the output does not vary by more than *distance* pixels, from the input *trace*, the point is removed. This continues until the minimum number of control points have been generated.

Control contains the resultant list of control points.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Control is provided as a method of reusing an existing *AVW_PointList2*.

RETURN VALUES

If successful *AVW_ExtractControlPoints()* returns an *AVW_PointList2*. On failure it returns *AVW_NULL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_ExtractControlPoints() will fail if:

ILLPAR
An illegal parameter, NULL trace, was passed to the routine.

SEE ALSO

AVW_AddFPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_FillPointList2(), AVW_MakeSpline(), AVW_PointList2, AVW_Point2

NAME

AVW_ExtractGradients – extracts a surface

SYNOPSIS

```
#include       "AVW_Render.h"

AVW_Gradients *AVW_ExtractGradients(param, last_gradients)
AVW_RenderParameters *param;
AVW_Gradients *last_gradients;
```

DESCRIPTION

AVW_ExtractGradients() uses the *param->ThresholdMinimum* and *param->ThresholdMaximum* members of the *AVW_RenderParameters* structure to extract all the surface points from the *AVW_Volume* specified within the structure. Normals are also precomputed, so that when the *AVW_Gradients* structure is used as input to *AVW_RenderGradients(),* very fast interactive renderings can be generated.

Last_gradients is provided as a method of reusing an existing *AVW_Gradients.* (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ExtractGradients()* returns an *AVW_Gradients.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ExtractGradients() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_DestroyGradients(), AVW_RenderGradients(), AVW_Gradients, AVW_RenderParameters

NAME

AVW_ExtractObject – extracts an object from a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ExtractObject(in_volume, orient, slice, seedlist, thresh_max,
                    thresh_min, mask_image, out_volume)
AVW_Volume *in_volume;
int orient, slice;
AVW_PointList2 *seedlist2;
double threshold_max, threshold_min;
AVW_Image *mask_image;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ExtractObject() extracts an object from a volume using thresholding, region growing and morphological operations.

In_volume specifies the input greyscale volume which contains the object. *Orient* and *slice* specify a slice of the volume on which the object can be defined by a threshold range and one or more seed points. *Seedlist* specifies one or more seed points from which the object can be grown. *Thresh_max* and *thresh_min* specify the threshold range of the object. *Mask_image* contains a 2D mask of the object for the specified slice. *Mask_image* may be set to *NULL* if the seed points and threshold range are sufficient to define the object on the specified slice. It is only necessary to provide a *mask_image* if manual interaction (limits) are required for definition of the object.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ExtractObject()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ExtractObject() will fail if the following is true:

ILLDT
Illegal data type.

SEE ALSO

AVW_ConditionalDilateGreyVolume(), AVW_ConnectAndKeepVolume(), AVW_ErodeVolume(), AVW_GetThresholdedBoundary(), AVW_ThresholdVolume(), AVW_Image

NAME

AVW_ExtractVisibleSurface – extracts surface from rendered image

SYNOPSIS

```
#include       "AVW_Render.h"

AVW_VisibleSurface *AVW_ExtractVisibleSurface(rendered, last_surface)
AVW_RenderedImage *rendered;
AVW_VisibleSurface *last_surface;
```

DESCRIPTION

AVW_ExtractVisibleSurface() creates an *AVW_VisibleSurface* which contains only the points in the *rendered->Volume* which were part of the visible surface displayed in *rendered->Image.* Pixel values are also stored, so that when the *AVW_VisibleSurface* structure is used as input to *AVW_VisibleSurfaceRender(),* very fast interactive renderings can be generated.

Last_surface is provided as a method of reusing an existing *AVW_Surface.* (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ExtractVisibleSurface()* returns an *AVW_VisibleSurface.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ExtractVisibleSurface() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_DestroySurface(), AVW_RenderVisibleSurface(), AVW_ExtractSurface(), AVW_VisibleSurface, AVW_RenderedImage

NAME

AVW_FFT2D – performs 2D Fast Fourier Transformations

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_FFT2D(input_image, direction, output_image)
AVW_Image *input_image;
int direction;
AVW_Image *output_image;
```

DESCRIPTION

AVW_FFT2D() returns the forward or inverse 2D FFT of the *input_image*, depending on whether *direction* is *AVW_FORWARD* or *AVW_BACKWARD*. For a forward FFT, image dimensions are automatically padded to the next larger power of 2 before the transform is applied. If these (possibly padded) dimensions are X2, Y2, the returned image will be an image of data type *AVW_COMPLEX* with dimensions X2/2 + 1, Y2. For an inverse FFT, the input image must be of data type *AVW_COMPLEX* and of dimensions X2/2 + 1, Y2 where X2 and Y2 are powers of 2. The returned image will be of data type *AVW_FLOAT* and of dimensions X2, Y2.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FFT2D()* returns an *AVW_Image*. On failure it returns *NULL*, and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FFT2D() will fail if one or more of the following are true:

BDSPCT
Bad Input Spectrum. An invalid image was entered for an inverse FFT.

ILLPAR
Illegal Parameter. An invalid value was given for direction.

BADMAL
Malloc Failed. Unable to allocate memory for return image.

SEE ALSO

AVW_FFT3D(), AVW_CreateButterworthCoeffs(), AVW_CreateCircularMTF(), AVW_CreateCoeffs(), AVW_CreateGaussianCoeffs(), AVW_CreateStoksethMTF(), AVW_DeconvDivideImage(), AVW_DeconvWienerImage(), AVW_ImageOpImage(), AVW_IterDeconvImage(), AVW_NearestNeighborDeconv(), AVW_Image

NAME AVW_FFT3D – performs 3D Fast Fourier Transformations

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_FFT3D(input_volume, direction, output_volume)
AVW_Volume *input_volume;
int direction;
AVW_Volume *output_volume
```

DESCRIPTION

AVW_FFT3D() returns the forward or inverse 3D FFT of the *input_volume,* depending on whether *direction* is *AVW_FORWARD* or *AVW_BACKWARD.* For a forward FFT, volume dimensions are automatically padded to the next larger power of 2 before the transform is applied. If these (possibly padded) dimensions are X2, Y2, Z2, the returned image will be an image of data type *AVW_COMPLEX* with dimensions X2 / 2 + 1, Y2, Z2. For an inverse FFT, the input volume must be of data type *AVW_COMPLEX* and of dimensions X2 / 2 + 1, Y2, Z2 where X2, Y2, and Z2 are powers of 2. The returned volume will be of data type *AVW_FLOAT* and of dimensions X2, Y2, Z2.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FFT3D()* returns an *AVW_Volume.* On failure it returns *NULL*, and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FFT3D() will fail if one or more of the following are true:

BDSPCT
Bad Input Spectrum. An invalid image was entered for an inverse FFT.

ILLPAR
Illegal Parameter. An invalid value was given for direction.

BADMAL
Malloc Failed. Unable to allocate memory for return volume.

SEE ALSO

AVW_FFT2D(), AVW_CreateButterworthCoeffs(), AVW_CreateCoeffs(), AVW_CreateGaussianCoeffs(), AVW_CreateSphericalMTF(), AVW_DeconvDivideVolume(), AVW_DeconvWienerVolume(), AVW_VolumeOpVolume(), AVW_IterDeconvVolume(), AVW_Volume

NAME

AVW_FillHolesImage – fills holes in an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_FillHolesImage(in_image, connectivity, out_image)
AVW_Image *in_image;
int connectivity;
AVW_Image *out_image;
```

DESCRIPTION

AVW_FillHolesImage() fills in holes in an image. Holes are defined as 0 valued pixels which are entirely surrounded by pixels with a value of 1.

Connectivity may be either *AVW_4_CONNECTED* or *AVW_8_CONNECTED* and specifies the neighbors to be used to determine the connected components.

In_image has to be a binary valued, i.e. ones and zeroes. *In_image,* and *out_image* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FillHolesImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FillHolesImage() will fail if:

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_FillHolesVolume(), AVW_FindImageComponents(), AVW_Image

NAME

AVW_FillHolesVolume – fills holes in a volume

SYNOPSIS

```
#include    "AVW.h"

AVW_Volume *AVW_FillHolesVolume(in_volume, connectivity, out_volume)
AVW_Volume *in_volume;
int connectivity;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_FillHolesVolume() fills in holes in a volume. Holes are defined as 0 valued voxels which are entirely surrounded by voxels with a value of 1.

Connectivity may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED* and specifies the neighbors to be used to determine the connected components.

In_volume has to be a binary valued, i.e. ones and zeroes. *In_volume*, and *out_volume* must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FillHolesVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FillHolesVolume() will fail if:

ILLDT
: Data type is not defined or supported.

SEE ALSO

AVW_FillHolesImage(), AVW_FindVolumeComponents(), AVW_Volume

NAME

AVW_FillPointList2 – fills in gaps in an AVW_PointList2 structure

SYNOPSIS

#include "AVW.h"

AVW_PointList2 ∗AVW_FillPointList2(plist, out_plist)
AVW_PointList2 ∗plist;
AVW_PointList2 ∗out_plist;

DESCRIPTION

AVW_FillPointList2() creates an 8-connected *AVW_PointList2* by filling in any gaps in *plist* with linearly interpolated points.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks. The last point in the point list is not automatically connected to the first point.

Out_plist is provided as a method of reusing an existing *AVW_PointList2.*

RETURN VALUES

If successful *AVW_FillPointList2()* returns an *AVW_PointList2.* On failure it returns *AVW_NULL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_FillPointList2() will fail if:

BADMAL
Malloc Failed. Unable to allocate memory.

SEE ALSO

AVW_AddFPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_MakeMaskFromTrace(), AVW_PointList2, AVW_Point2

NAME

AVW_FillPointList3 – fills in gaps in an AVW_PointList3 structure

SYNOPSIS

```
#include "AVW.h"

AVW_PointList3 *AVW_FillPointList3(plist, out_plist)
AVW_PointList3 *plist;
AVW_PointList3 *out_plist;
```

DESCRIPTION

AVW_FillPointList3() creates an 26-connected *AVW_PointList3* by filling in any gaps in *plist* with linearly interpolated points.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks. The last point in the point list is not automatically connected to the first point.

Out_plist is provided as a method of reusing an existing *AVW_PointList3.*

RETURN VALUES

If successful *AVW_FillPointList3()* returns an *AVW_PointList3.* On failure it returns *AVW_NULL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_FillPointList3() will fail if:

BADMAL
Malloc Failed. Unable to allocate memory.

SEE ALSO

AVW_AddPoint3(), AVW_CreatePointList3(), AVW_DestroyPointList3(), AVW_PointList3, AVW_Point3

NAME

AVW_FindImageComponents – finds the connected regions in an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_FindImageComponents(in_image, label_flag, connectivity,
                        max_size, min_size, out_image)
AVW_Image *in_image;
int label_flag, connectivity, max_size, min_size;
AVW_Image *out_image;
```

DESCRIPTION

AVW_FindImageComponents() finds all of the unique four or eight connected regions in *in_image.*

In_image must be of the data type *AVW_UNSIGNED_CHAR*. It does not have to be binary valued, but all nonzero values are treated as ones.

The *label_flag* parameter can be set to *AVW_NO_LABEL*, *AVW_LABEL*, *AVW_SORT_AND_LABEL, and AVW_INVERSE_SORT_AND_LABEL.*

> When set to *AVW_NO_LABEL,* the returned image contains all 1's and 0's (NOTE: This option is slightly slower than AVW_LABEL, as the labeling is actually done anyway and then removed in a post-processing step.)
>
> When set to *AVW_LABEL*, each seperately connected object contains a unique label. The objects are labeled 1 thru N.
>
> When set to *AVW_SORT_AND_LABEL*, the objects are labeled as above but sorted largest to smallest.
>
> When set to *AVW_INVERSE_SORT_AND_LABEL,* the objects are labeled as above but sorted smallest to largest.

Connectivity may be either *AVW_4_CONNECTED* or *AVW_8_CONNECTED* and specifies the neighbors to be used to determine the connected components.

Max_size specifies the largest allowable component size. Components larger than this size will be deleted, or in other words set to zero.

Min_size specifies the minimum allowable component size. Components smaller than this size will be deleted, or in other words set to zero.

The returned image will be of data type *AVW_UNSIGNED_CHAR* unless the number of components found is greater than 255, in which case the data type will be *AVW_UNSIGNED_SHORT.*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FindImageComponents()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_FindVolumeComponents(), AVW_FindImageEdges(), AVW_LabelImageFromEdges(), AVW_DefineConnected(), AVW_ConnectAndDeleteImage(), AVW_ConnectAndKeepImage(), AVW_Image

NAME

AVW_FindImageEdges – finds the edges in an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_FindImageEdges(in_image, edge_flag, connectivity, out_image)
AVW_Image *in_image;
int edge_flag;
int connectivity;
AVW_Image *out_image;
```

DESCRIPTION

AVW_FindImageEdges() finds all of the nonzero pixels in *in_image* that have a zero as a four or eight connected neighbor. These edges are labeled as ones in *out_image.*

In_image must be of the data type *AVW_UNSIGNED_CHAR.*

If *edge_flag* is set to one, the the edges appear on top of the nonzero pixels that have a zero neighbor. If *edge_flag* is set to zero, the edges appear on top of zero pixels that have a nonzero neighbor.

Connectivity may be either *AVW_4_CONNECTED* or *AVW_8_CONNECTED* and specifies the neighbors to be used to determine the edges.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FindImageEdges()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_FindVolumeEdges(), AVW_FindImageComponents(), AVW_LabelImageFromEdges()
AVW_Image

NAME

AVW_FindImageMaxMin – finds the maximum and minimum values of an image

SYNOPSIS

```
#include "AVW.h"

int AVW_FindImageMaxMin(image, max_val, min_val)
AVW_Image *image;
double *max_val, *min_val;
```

DESCRIPTION

AVW_FindImageMaxMin() finds the maximum and minimum data values in *image* and returns them in *max_val* and *min_val.*

After calculation, the values are stored in the information string of the input image to allow quicker future max / min queries with the *AVW_QuickImageMaxMin* function.

RETURN VALUES

If successful, *AVW_FindImageMaxMin()* returns *AVW_SUCCESS.* On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FindImageMaxMin() will fail if:

NOTSUP
Image data type not supported.

SEE ALSO

AVW_FindVolumeMaxMin(), AVW_QuickImageMaxMin(), AVW_QuickVolumeMaxMin(), AVW_Image

NAME

AVW_FindRenderedPoint – returns volume coordinates from a rendered image

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_FindRenderedPoint(rendered, in, out)
AVW_RenderedImage *rendered;
AVW_FPoint2 *in;
AVW_FPoint3 *out;
```

DESCRIPTION

AVW_FindRenderedPoint() transforms a point from rendered space back into the input volumes space. The point *in,* is specified as an *AVW_Point2* location within the *rendered->Image.* The transformed point is returned in the *out AVW_Point3* structure.

RETURN VALUES

AVW_FindRenderedPoint() returns *AVW_SUCCESS* if *in* can be successfully transformed to a location within the input volume. It returns *AVW_FAIL* if the point transforms to a location outside the volume.

SEE ALSO

AVW_DrawRenderedLine(), AVW_DrawRenderedPoint(), AVW_RenderVolume(), AVW_RenderedImage AVW_FPoint2, AVW_FPoint3

NAME

AVW_FindRotation – adjusts render matrix to move point interactively

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_FindRotation(pt3, pt2, r_param)
AVW_Point3 *out;
AVW_Point2 *in;
AVW_RenderParameters *r_param;
```

DESCRIPTION

AVW_FindRotation() applies single degree X, Y, and Z screen relative rotations recursively to the rendering rotation matrix, *r_param->Matrix, until the 3-D point, pt3,* is rotated into the position which is closest to the 2-point, *pt2.*

This function is used to to implement interactive "Change View", where the user selects a point on a rendered image and "drags" that point to a new location, thuse rotating the rendering.

RETURN VALUES

AVW_FindRotation() returns *AVW_SUCCESS* if *in* can be successfully transformed to a location within the input volume. It returns *AVW_FAIL* if the point transforms to a location outside the volume.

SEE ALSO

AVW_FindRenderedPoint(), AVW_ExtractVisibleSurface(), AVW_RenderVisibleSurface(), AVW_RenderParameters, AVW_Point3, AVW_Point2,

NAME

AVW_FindSurfaceArea – measures surface area

SYNOPSIS

```
#include "AVW.h"

int AVW_FindSurfaceArea(rendered, mask, mask_depth, surface_voxels,
                surface_faces, planar_area, surface_area)
AVW_RenderedImage *rendered;
AVW_Image *mask;
int mask_depth;
int *surface_voxels;
int *surface_faces;
int *planar_area;
double *surface_area;
```

DESCRIPTION

AVW_FindSurfaceArea() returns a surface voxels, surface faces, planar area, and a estimated surface area.

Rendered specifies the surface to measure.

Mask is an *AVW_Image* in which all non-zero pixels indicate an area on the surface to measure. *Mask->Width* and *mask->Height* must be the same as *rendered->Width* and *rendered->Height. Mask* must have a datatype of *AVW_UNSIGNED_CHAR.*

Mask_depth is a flag, which when set to *AVW_TRUE,* constrains the surface grow to the actual minimum and maximum depths found in the *mask.*

If successful, *surface_voxels* will contain a count of each connected surface voxel which was within the *mask. Surface_faces* is a count of the exposed faces from these voxels. *Planar_area* is a count of the pixels in the *mask* which were over a surface point in the *rendered* image. Knowing the exposed surface faces for each surface voxel, triangles, rectangles and squares can be measured to estimate the value returned in *surface_area.*

RETURN VALUES

If successful *AVW_FindSurfaceArea()* returns *AVW_TRUE.* On failure it returns *AVW_FALSE* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FindSurfaceArea() will fail if one or more of the following is true:

BADMAL
: Unable to allocate sufficient memory.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_FindSurfaceDistance(), AVW_FindRenderedPoint(), AVW_RenderedImage, AVW_Image

NAME

AVW_FindSurfaceDistance − returns distance along a surface

SYNOPSIS

```
#include "AVW.h"

int AVW_FindSurfaceDistance(rendered, pt1, pt2, curved)
AVW_RenderedImage *rendered;
AVW_Point2 *pt1, *pt2;
double *curved;
```

DESCRIPTION

AVW_FindSurfaceDistance() returns distance between two point on a surface.

Rendered specifies the surface to measure.

Pt1 and *pt2* specify the two end points of the curved line to measure. Both points must be on the surface and all pixels in a straight line between then in the rendered image must be surface points.

Curved will contain the measured distance if *AVW_FindSurfaceDistance()* returns successfully. The surface distance is calculated by summing the distances between each point on the surface.

RETURN VALUES

If successful *AVW_FindSurfaceDistance()* returns *AVW_TRUE.* On failure it returns *AVW_FALSE* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FindSurfaceDistance() will fail if one or more of the following is true:

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_FindSurfaceArea(), AVW_FindRenderedPoint(), AVW_RenderedImage, AVW_Point2

NAME

AVW_FindSurfacePoints – locates all surface points

SYNOPSIS

```
#include "AVW_ObjectMap.h"

AVW_PointList3 *AVW_FindSurfacePoints(volume, object_map, thresh_max,
                    thresh_min, connect_flag)
AVW_Volume *volume;
AVW_ObjectMap *object_map;
double thresh_max, thresh_min;
int connect_flag;
```

DESCRIPTION

AVW_FindSurfacePoints() locates all points in the input *volume* which have an intensity value greater than or equal to *thresh_min* and less than or equal to *thresh_max* and have at least one connected neighbor which is not that within this range. If the *object_map* parameter is not *NULL,* in addition to checking thresholds the voxels found must belong to an object which has it's *DisplayFlag* set to *AVW_TRUE,* and at least one neighbor outside the threshold range or having it's set to *AVW_FALSE.*

Connect_flag may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED* and specifies the neighbors to be used to determine the connected components.

RETURN VALUES

If successful *AVW_FindSurfacePoints()* returns an *AVW_PointList3.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FindSurfacePoints() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_DestroyPointList3(), AVW_ExtractSurface(), AVW_ObjectMap

NAME

AVW_FindTraceCenter – finds the the center point of a trace

SYNOPSIS

```
#include "AVW.h"

int AVW_FindTraceCenter(trace, point)
AVW_PointList2 *trace;
AVW_Point2 *point;
```

DESCRIPTION

AVW_FindTraceCenter() finds the center point of a traced region defined by *trace.* The center point coordinates are returned in *point.*

RETURN VALUES

If successful, *AVW_FindTraceCenter()* returns *AVW_SUCCESS.* On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_FillPointList2(), AVW_Image, AVW_Point2, AVW_PointList2

NAME

AVW_FindTreeIndex – get a point from a list

SYNOPSIS

```
#include "AVW_Tree.h"

int AVW_FindTreeIndex(tree, point)
AVW_Tree *tree;
AVW_Point3 *point;
```

DESCRIPTION

AVW_FindTreeIndex() gets finds the index of a specified *point* from a tree structure.

Tree is an *AVW_Tree.*

RETURN VALUES

If succesful the index into the tree structure is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

SEE ALSO

AVW_AddTreeChild(), AVW_CreateTree(), AVW_DestroyTree(), AVW_LoadTree(), AVW_SaveTree(), AVW_Point3, AVW_Tree

NAME

AVW_FindTreeStart – finds a tree starting point

SYNOPSIS

```
#include "AVW_Tree.h"

int AVW_FindTreeStart(in_volume, startpos)
AVW_Volume *in_volume;
AVW_Point3 *startpos;
```

DESCRIPTION

AVW_FindTreeStart() finds the closest endpoint of a thinned skeleton to the approximate root location contained in *startpos.* The coordinates of *startpos* are then set to the location of the endpoint.

RETURN VALUES

If successful, *AVW_FindTreeStart()* returns *AVW_SUCCESS.* On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FindTreeStart() will fail if one or more of the following are true:

ILLPAR
: Point specified is outside of the image.

ILLDT
: Illegal datatype.

EMPTYI
: All pixels in the image are zero.

SEE ALSO

AVW_MakeTree(), AVW_Tree, AVW_Point3

NAME

AVW_FindVolumeComponents – finds the connected regions in a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_FindVolumeComponents(in_volume, label_flag, connectivity,
                    max_size, min_size, out_volume)
AVW_Volume *in_volume;
int label_flag, connectivity, max_size, min_size;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_FindVolumeComponents() finds all of the unique six or twenty six connected regions in *in_volume.*

In_volume must be of the data type *AVW_UNSIGNED_CHAR*. It does not have to be binary valued, but all nonzero values are treated as ones.

The *label_flag* parameter can be set to *AVW_NO_LABEL, AVW_LABEL, AVW_SORT_AND_LABEL, and AVW_INVERSE_SORT_AND_LABEL.*

> When set to *AVW_NO_LABEL,* the returned volume contains all 1's and 0's (NOTE: This option is slightly slower than AVW_LABEL, as the labeling is actually done anyway and then removed in a post-processing step.)
>
> When set to *AVW_LABEL*, each seperately connected object contains a unique label. The objects are labeled 1 thru N.
>
> When set to *AVW_SORT_AND_LABEL,* the objects are labeled as above but sorted largest to smallest.
>
> When set to *AVW_INVERSE_SORT_AND_LABEL,* the objects are labeled as above but sorted smallest to largest.

Connectivity may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED* and specifies the neighbors to be used to determine the connected components.

Max_size specifies the largest allowable component size. Components larger than this size will be deleted, or in other words set to zero.

Min_size specifies the minimum allowable component size. Components smaller than this size will be deleted, or in other words set to zero.

The returned volume will be of data type *AVW_UNSIGNED_CHAR* unless the number of components found is greater than 255, in which case the data type will be *AVW_UNSIGNED_SHORT*.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FindVolumeComponents()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_ConnectAndDeleteVolume(), AVW_ConnectAndKeepVolume(), AVW_DefinedConnected(), AVW_FindImageComponents(), AVW_FindVolumeEdges(), AVW_LabelVolumeFromEdges(), AVW_Volume

NAME

AVW_FindVolumeEdges – finds the edges in a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_FindVolumeEdges(in_volume, edge_flag, connectivity, out_volume)
AVW_Volume *in_volume;
int edge_flag;
int connectivity;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_FindVolumeEdges() finds all of the nonzero pixels in *in_volume* that have a zero as a 6 or 26 connected neighbor. These edges are labeled as ones in *out_volume.*

In_volume must be of the data type *AVW_UNSIGNED_CHAR.*

If *edge_flag* is set to one, the the edges appear on top of the nonzero voxels that have a zero neighbor. If *edge_flag* is set to zero, the edges appear on top of zero voxels that have a nonzero neighbor.

Connectivity may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED* and specifies the neighbors to be used to determine the edges.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FindVolumeEdges()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_FindImageEdges(), AVW_FindVolumeComponents(), AVW_LabelVolumeFromEdges(), AVW_Volume

NAME

AVW_FindVolumeMaxMin – finds the maximum and minimum values of a volume

SYNOPSIS

```
#include "AVW.h"

int AVW_FindVolumeMaxMin(volume, max_val, min_val)
AVW_Volume *volume;
double *max_val, *min_val;
```

DESCRIPTION

AVW_FindVolumeMaxMin() finds the maximum and minimum data values in *volume* and returns them in *max_val* and *min_val.*

After calculation, the values are stored in the information string of the input volume to allow quicker future max / min queries when the *AVW_QuickVolumeMaxMin* function is used.

RETURN VALUES

If successful, *AVW_FindVolumeMaxMin()* returns *AVW_SUCCESS*. On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FindVolumeMaxMin() will fail if:

NOTSUP
Data type not supported.

SEE ALSO

AVW_FindImageMaxMin(), AVW_QuickImageMaxMin(), AVW_QuickVolumeMaxMin(), AVW_Volume

NAME

AVW_FindVolumePoint – returns a volume coordinates position within a rendered image

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_FindVolumePoint(rendered, in, out)
AVW_RenderedImage *rendered;
AVW_FPoint3 *in;
AVW_FPoint3 *out;
```

DESCRIPTION

Provided with a position within the originating volume,

The point *in,* is specified as an *AVW_FPoint3* location within the *rendered->Volume.* The transformed point is returned in an *AVW_FPoint3.*

RETURN VALUES

AVW_FindVolumePoint() returns *AVW_SUCCESS* if *in* can be successfully transformed to a location within the input volume. It returns *AVW_FAIL* if the point transforms to a location outside the volume.

SEE ALSO

AVW_DrawRenderedLine(), AVW_DrawRenderedPoint(), AVW_FindRenderedPoint(), AVW_RenderVolume(), AVW_RenderedImage AVW_FPoint2, AVW_FPoint3

NAME

AVW_FlattenImageHistogram – distributes pixel intensities evenly

SYNOPSIS

```
#include       "AVW_Histogram.h"

AVW_Image *AVW_FlattenImageHistogram(in_image, max, min, out_image)
AVW_Image *in_image;
double max, min;
AVW_Image *out_image;
```

DESCRIPTION

AVW_FlattenImageHistogram() attempts to produce an even distribution of pixels across the grey scale range specified by *max* and *min*.

An image which has roughly equal numbers of pixels at every greyscale value will tend to exhibit maximal contrast across the entire greyscale range. Original greyscale values may be brightened, darkened, and or binned together to effect the flattening procedure.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FlattenImageHistogram()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FlattenImageHistogram() will fail if the following is true:

BADMAL
Malloc Failed. Could not allocate memory for the results.

ILLPAR
Range of max and min must be less than or equal to the range of in_image.

SEE ALSO

AVW_FlattenVolumeHistogram(), AVW_PreserveImageHistogram(), AVW_MatchImageHistogram(), AVW_Image

NAME

AVW_FlattenVolumeHistogram – distributes voxel intensities evenly

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Volume *AVW_FlattenVolumeHistogram(in_vol, max, min, out_vol)
AVW_Volume *in_vol;
double max, min;
AVW_Volume *out_vol;
```

DESCRIPTION

AVW_FlattenVolumeHistogram() attempts to produce an even distribution of voxels across the grey scale range specified by *max* and *min.*

A volume which has roughly equal numbers of voxels at every greyscale value will tend to exhibit maximal contrast across the entire greyscale range. Original greyscale values may be brightened, darkened, and or binned together to effect the flattening procedure.

Out_vol is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_vol* meet the requirements of the function. In this case the pointer to *out_vol* is returned by the function. If not reusable *out_vol* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FlattenVolumeHistogram()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FlattenVolumeHistogram() will fail if the following is true:

BADMAL
Malloc Failed. Could not allocate memory for the results.

ILLPAR
Range of max and min must be less than or equal to the range of in_vol.

SEE ALSO

AVW_FlattenImageHistogram(), AVW_PreserveVolumeHistogram(), AVW_MatchVolumeHistogram(), AVW_Volume

NAME

AVW_FlipImage – flips an image

SYNOPSIS

```
#include       "AVW.h"

AVW_Image *AVW_FlipImage(in_image, axes, out_image)
AVW_Image *in_image;
int axes;
AVW_Image *out_image;
```

DESCRIPTION

AVW_FlipImage() flips *in_image* in the specified direction. If *axes* is set to *AVW_FLIPX* the image will be flipped horizontally. If *axes* is set to *AVW_FLIPY* the image will be flipped vertically. If *axes* is set to *AVW_FLIPX | AVW_FLIPY* the image will be flipped in both directions.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FlipImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_FlipVolume(), AVW_Image

NAME

AVW_FlipVolume – flips a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_FlipVolume(in_volume, axes, out_volume)
AVW_Volume *in_volume;
int axes;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_FlipVolume() flips *in_volume* in the specified direction. If *axes* is set to *AVW_FLIPX* the volume will be flipped horizontally. If *axes* is set to *AVW_FLIPY* the volume will be flipped vertically. If *axes* is set to *AVW_FLIPZ* the order of the slices in the volume will be flipped. Any combination of these directions is also valid.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FlipVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_FlipImage(), AVW_Volume

NAME

AVW_FormatSupports – determines if image file format supports a property

SYNOPSIS

#include "AVW_ImageFile.h"

int AVW_FormatSupports(format, property)
char *format;
int property;

DESCRIPTION

AVW_FormatSupports() is used to determine if an image file format supports a property.

format is a character string which identifies the format. *property is a property or mask* Property values are defined in AVW_ImageFile.h and include:

AVW_SUPPORT_UNSIGNED_CHAR AVW_SUPPORT_SIGNED_CHAR AVW_SUPPORT_UNSIGNED_SHORT AVW_SUPPORT_SIGNED_SHORT AVW_SUPPORT_UNSIGNED_INT AVW_SUPPORT_SIGNED_INT AVW_SUPPORT_FLOAT AVW_SUPPORT_COMPLEX AVW_SUPPORT_COLOR AVW_SUPPORT_2D AVW_SUPPORT_3D AVW_SUPPORT_4D AVW_SUPPORT_READ AVW_SUPPORT_WRITE

For example to determine if "GE9800" files can be written from AVW,

```
int ret;

ret = AVW_FormatSupports("GE9800",AVW_SUPPORT_WRITE);
if(ret == AVW_SUCCESS)
        printf("Write Supported0);
else
        printf("Write not supporte0);
```

RETURN VALUES

AVW_TRUE is returned if the image file format does support the property, otherwise *AVW_FALSE* is returned.

SEE ALSO

AVW_CreateImageFile(), AVW_ListFormats(), AVW_ImageFile

NAME

AVW_Malloc – allocates system memory

SYNOPSIS

```
#include     "AVW.h"

void *AVW_Malloc(size)
unsigned int size;

void *AVW_Calloc(num, size)
unsigned int num;
unsigned int size;

void *AVW_Realloc(size, ptr)
unsigned int size;
void *ptr;

void AVW_Free(ptr)
void *ptr;
```

DESCRIPTION

These procedures provide a platform and compiler independent interface for memory allocation. Programs that need to transfer ownership of memory blocks between *AVW* and other modules should use these routines rather than the native *malloc()* and *free()* routines provided by the *C* run-time library.

AVW_Malloc returns a pointer to a size bytes suitably aligned for any use.

AVW_Calloc allocates space for an array nelem elements of *size* elsize. The space is initialized to zeros.

Tcl_Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the new block. The contents will be unchanged up to the lesser of the new and old sizes. The returned location may be different from *ptr.*

Tcl_Free makes the space referred to by *ptr* available for further allocation.

SEE ALSO

malloc(), free()

NAME

AVW_FunctionImage – applies a function to an image

SYNOPSIS

```
#include "AVW_Parse.h"

AVW_Image *AVW_FunctionImage(function_code, in_image, out_image)
int function_code;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_FunctionImage() returns an *AVW_Image* which is the result of applying a specified function to *in_image.*

The following function codes are defined in *AVW_Parse.h:*

> *AVW_F_ABS* returns the absolute value of each *in_image* pixel in the corresponding *out_image* pixel.
>
> *AVW_F_SQRT* returns the square root of each *in_image* pixel in the corresponding *out_image* pixel.
>
> *AVW_F_LOG* returns the natural log of each *in_image* pixel in the corresponding *out_image* pixel.
>
> *AVW_F_EXP* returns the exponential value of each *in_image* pixel in the corresponding *out_image* pixel.
>
> *AVW_F_COUNT* returns a counter value in each pixel of *out_image. In_image* is used to determine size only. The counter starts at 1 and increases each time *AVW_FunctionImage()* is called.
>
> *AVW_F_XPOS* returns the column number in each *out_image* pixel value. In_image is used to determine size only. The first column is assigned #1 and the last column is assigned *in_image*->*Width;*
>
> *AVW_F_YPOS* returns the row number in each *out_image* pixel value. In_image is used to determine size only. The first row is assigned #1 and the last row is assigned *in_image*->*Height;*
>
> *AVW_F_MAX* returns an *out_image* which contains the maximum value within *in_image* in each of *out_image* pixels.
>
> *AVW_F_MIN* returns an *out_image* which contains the minimum value within *in_image* in each of *out_image* pixels.
>
> *AVW_F_AVG* returns an *out_image* which contains the average value within *in_image* in each of *out_image* pixels.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FunctionImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FunctionImage() will fail if one or more of the following are true:

NOTSUP
Unknown function type, or unsupported input data type.

SEE ALSO

AVW_FunctionVolume(), AVW_ConstantOpImage(), AVW_ImageOpConstant(), AVW_ImageOpImage(), AVW_Image

NAME

AVW_FunctionVolume – applies a function to a volume

SYNOPSIS

```
#include "AVW_Parse.h"

AVW_Volume *AVW_FunctionVolume(function_code, in_volume, out_volume)
int function_code;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_FunctionVolume() returns an *AVW_Volume* which is the result of applying a specified function to *in_volume.*

AVW_FunctionVolume() calls *AVW_FunctionImage()* with each slice in the input volume(s) to produce the output volume.

The following function codes are defined in *AVW_Parse.h:*

AVW_F_ABS returns the absolute value of each *in_volume* voxel in the corresponding *out_volume* voxel.

AVW_F_SQRT returns the square root of each *in_volume* voxel in the corresponding *out_volume* voxel.

AVW_F_LOG returns the natural log of each *in_volume* voxel in the corresponding *out_volume* voxel.

AVW_F_EXP returns the exponential value of each *in_volume* voxel in the corresponding *out_volume* voxel.

AVW_F_COUNT returns a counter value in each voxel of *out_volume. In_volume* is used to determine size only. The counter starts at 1 and increases each time *AVW_Functionzvolume()* is called.

AVW_F_XPOS returns the column number in each *out_volume* voxel value. In_volume is used to determine size only. The firdt column is assigned #1 and the last column is assigned *in_volume->Width;*

AVW_F_YPOS returns the row number in each *out_volume* voxel value. In_volume is used to determine size only. The first row is assigned #1 and the last row is assigned *in_volume->Height;*

AVW_F_MAX returns an *out_volume* which contains the maximum value within *in_volume* in each of *out_volume* voxels.

AVW_F_MIN returns an *out_volume* which contains the minimum value within *in_volume* in each of *out_volume* voxels.

AVW_F_AVG returns an *out_volume* which contains the average value within *in_volume* in each of *out_volume* voxels.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_FunctionVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_FunctionVolume() will fail if one or more of the following are true:

NOTSUP
Unknown function type, or unsupported input data type.

SEE ALSO

AVW_ConstantOpVolume(), AVW_FunctionImage(), AVW_VolumeOpConstant(), AVW_VolumeOpVolume(), AVW_Volume

NAME

AVW_GetBoundaryAndDelete – finds the boundary of a thresholded region

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_GetBoundaryAndDelete(image, thresh_max, thresh_min,
                seed_point, del_value, trace)
AVW_Image *image;
double thresh_max, thresh_min;
AVW_Point2 *seed_point;
double del_value;
AVW_PointList2 *trace;
```

DESCRIPTION

AVW_GetBoundaryAndDelete() finds and returns an *AVW_PointList2*, *trace,* which contains the boundary of the region defined by the *seed_point*, *threshold_max*, and *threshold_min*. The region is grown from the *seed_point* and includes all pixels which are connected via four neighbors to it and within the threshold values. The defined region is set to *del_value* after the boundary is found.

Trace is provided as a method of reusing an existing *AVW_PointList2.*

RETURN VALUES

If successful *AVW_GetBoundaryAndDelete()* returns an *AVW_PointList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetBoundaryAndDelete() will fail if:

ILLDT
Data type is not defined or supported.

ILLIMG
An illegal image was passed to the function.

SEE ALSO

AVW_GetThresholdedBoundary(), AVW_GetMaskBoundary(), AVW_ThresholdImage(), AVW_PointList2, AVW_Point2, AVW_Image

NAME

AVW_GetClippedBoundary – returns a boundary for a connected thresholded region

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2* AVW_GetClippedBoundary ( image, thresh_max, thresh_min,
        seed_point, type, gap_size, trace)

AVW_Image *inImage;
double thresh_max;
double thresh_min;
AVW_Point2 *seed_point;
int type;
int gap_size;
AVW_PointList2* theList;
```

DESCRIPTION

AVW_GetClippedBoundary() finds and returns an *AVW_PointList2, trace,* which contains the boundary of the region defined by the *seed_point, threshold_max,* and *threshold_min.* The region is grown from the *seed_point* and includes all pixels which are connected via four neighbors to it and within the threshold values.

Type specifies whether the returned border threshold range or off the edge. Valid values are *AVW_AUTO_ON_EDGE (1)* and *AVW_AUTO_OFF_EDGE (0).*

Gap_size specifies the number of layers to be peeled away from and then added back to the thresholded region during the border detection process. Larger values will close the border across thin areas of the object.

Trace is provided as a method of reusing an existing *AVW_PointList2.*

RETURN VALUES

If successful *AVW_GetClippedBoundary()* returns an *AVW_PointList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_GetThresholdedBoundary(), AVW_GetBoundaryAndDelete(), AVW_GetMaskBoundary(), AVW_ThresholdImage(), AVW_Point2, AVW_Image

NAME

AVW_GetCurved – extracts a curved image from a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_GetCurved(volume, orientation, trace)
AVW_Volume *volume;
int orientation;
AVW_PointList2 *trace;
```

DESCRIPTION

When supplied with an *AVW_Volume, orientation,* and *trace, AVW_GetCurved()* returns an *AVW_Image* representing the curved section described. The curved section is constructed by taking the lines perpendicular to each point of the *trace* and putting them into the rows of an image. *Orientation* must be *AVW_TRANSVERSE, AVW_CORONAL,* or *AVW_SAGITTAL* which are defined in *AVW.h.*

RETURN VALUES

If successful *AVW_GetCurved()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetCurved() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Couldn't allocate memory.

ILLPAR
Illegal trace or orientation.

SEE ALSO

AVW_AddPoint2(), AVW_CreatePointList2(), AVW_GetOblique(), AVW_GetOrthogonal(), AVW_PointList2, AVW_Volume

NAME

AVW_GetErrorMessage – returns the current error number

SYNOPSIS

#include "AVW.h"

char *AVW_GetErrorMessage()

DESCRIPTION

AVW_GetErrorMessage() returns a pointer to the current error message. This pointer should not be freed or written to.

SEE ALSO

AVW_Error(), AVW_GetErrorNumber(), AVW_SetError()

NAME

AVW_GetErrorNumber – returns the current error number

SYNOPSIS

```
#include "AVW.h"
#include "AVW_Error.h"

int AVW_GetErrorNumber()
```

DESCRIPTION

AVW_GetErrorNumber() returns the current error number.

A value of zero *(NOERR)* indicates that no error has occured. See the AVW_Error.h include file for specific error values.

SEE ALSO

AVW_Error(), AVW_GetErrorMessage(), AVW_SetError()

NAME

AVW_GetFPoint2 – get a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_GetFPoint2(trace, which_point, point)
AVW_FPointList2 *trace;
int which_point;
AVW_FPoint2 *point;
```

DESCRIPTION

AVW_GetFPoint2() gets a specific point from a point list.

Trace is an *AVW_FPointList2.*

Which_point specifies the point to get from the list. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetFPoint2 will fail if the following is true:

ILLPAR
which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_GetFPoint3(), AVW_GetPoint2(), AVW_GetPoint3(), AVW_GetIPoint2(), AVW_GetIPoint3(), AVW_GetPointValue(), AVW_AddFPoint2(), AVW_CreateFPointList2(), AVW_RemoveFPoint2(), AVW_FPoint2, AVW_FPointList2

NAME

AVW_GetFPoint3 – get a point from a list

SYNOPSIS

#include "AVW.h"

```
int AVW_GetFPoint3(trace, which_point, point)
AVW_FPointList3 *trace;
int which_point;
AVW_FPoint3 *point;
```

DESCRIPTION

AVW_GetFPoint3() gets a specific point from a point list.

Trace is an *AVW_FPointList3*.

Which_point specifies the point to get from the list. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetFPoint3 will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_GetFPoint2(), AVW_GetIPoint2(), AVW_GetIPoint3(), AVW_GetPoint2(), AVW_GetPoint3(), AVW_GetPointValue(), AVW_AddFPoint3(), AVW_CreateFPointList3(), AVW_RemoveFPoint3(), AVW_FPoint3, AVW_FPointList3

NAME

AVW_GetHistogramMedianValue – returns the mode value of a histogram

SYNOPSIS

```
#include "AVW.h"

double AVW_GetHistogramMedianValue(histogram)
AVW_Histogram *histogram;
```

DESCRIPTION

AVW_GetHistogramMedianValue() returns the value of the *histogram* bin where half of the counts in the entire histogram occurs. Two passes through the histogram are made. The first pass calculates the total number of counts in the histogram. The second pass determines which bin contains the half of total count.

RETURN VALUES

If successful, *AVW_GetHistogramMedianValue()* returns the median value of the histogram. On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. To insure valid results *AVW_ErrorNumber* should be checked after calling this function.

ERRORS

AVW_GetHistogramMedianValue() will fail if one or more of the following are true:

ILLHIS
: Illegal histogram, histogram is NULL.

SEE ALSO

AVW_CreateHistogram(), AVW_GetHistogramModeValue(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_Histogram

NAME

AVW_GetHistogramModeValue – returns the mode value of a histogram

SYNOPSIS

#include "AVW.h"

double AVW_GetHistogramModeValue(histogram)
AVW_Histogram *histogram;

DESCRIPTION

AVW_GetHistogramModeValue() returns the value of the *histogram* bin that has the highest number of counts.

RETURN VALUES

If successful, *AVW_GetHistogramModeValue()* returns the mode value of the histogram. On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. To insure valid results *AVW_ErrorNumber* should be checked after calling this function.

ERRORS

AVW_GetHistogramModeValue() will fail if one or more of the following are true:

ILLHIS
Illegal histogram, histogram is NULL.

SEE ALSO

AVW_CreateHistogram(), AVW_GetHistogramMedianValue(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_Histogram

NAME

AVW_GetIPoint2 – get a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_GetIPoint2(trace, which_point, point)
AVW_IPointList2 *trace;
int which_point;
AVW_IPoint2 *point;
```

DESCRIPTION

AVW_GetIPoint2() gets a specific point from a point list.

Trace is an *AVW_IPointList2*.

Which_point specifies the point to get from the list. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetIPoint2 will fail if the following is true:

ILLPAR
: *which_value* is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_GetIPoint3(), AVW_GetFPoint2(), AVW_GetFPoint3(), AVW_GetPoint2(), AVW_GetPoint3(), AVW_GetPointValue(), AVW_AddIPoint2(), AVW_CreateIPointList2(), AVW_RemoveIPoint2(), AVW_IPoint2, AVW_IPointList2

NAME

AVW_GetIPoint3 – get a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_GetIPoint3(trace, which_point, point)
AVW_IPointList3 *trace;
int which_point;
AVW_IPoint3 *point;
```

DESCRIPTION

AVW_GetIPoint3() gets a specific point from a point list.

Trace is an *AVW_IPointList3*.

Which_point specifies the point to get from the list. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetIPoint3 will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_GetIPoint2(), AVW_GetFPoint2(), AVW_GetFPoint3(), AVW_GetPoint2(), AVW_GetPoint3(), AVW_GetPointValue(), AVW_AddIPoint3(), AVW_CreateIPointList3(), AVW_RemoveIPoint3(), AVW_IPoint3, AVW_IPointList3

NAME

AVW_GetImageChannel – extracts a specific channel from an AVW_Color image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_GetImageChannel(in_image, channel, out_image)
AVW_Image *in_image;
int channel;
AVW_Image *out_image;
```

DESCRIPTION

AVW_GetImageChannel() returns the contents of a specified *channel* within the *AVW_COLOR* image, *in_image.*

Channel can be specified as any of the following:

AVW_RED_CHANNEL **-** specifies the red channel.

AVW_GREEN_CHANNEL **-** specifies the green channel.

AVW_BLUE_CHANNEL **-** specifies the blue channel.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_GetImageChannel()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetImageChannel() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

ILLPAR
Illegal Parameter. in_image is not AVW_COLOR or channel specification is invalid.

SEE ALSO

AVW_GetVolumeChannel(), AVW_PutImageChannel(), AVW_MakeColorImage(), AVW_DestroyImage() AVW_Image()

NAME

AVW_GetImageHistogram – computes an image histogram

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Histogram *AVW_GetImageHistogram(in_image, mask_image, mask_value,
                    sumflag, histo)
AVW_Image *in_image;
AVW_Image *mask_image;
int sumflag, mask_value;
AVW_Histogram *histo;
```

DESCRIPTION

AVW_GetImageHistogram() computes the histogram, *histo,* of *in_image.* The *mask_image* can be used to specify a region of the image for which the histogram is to be computed. Only pixels from the *in_image* corresponding to pixels of the *mask_image* equal to the *mask_value* will be used in the histogram computation. If the *mask_image* is equal to *NULL* the entire *in_image* is used to compute the histogram.

Sumflag, if set to *AVW_TRUE,* specifies that the occurrences are summed into *histo. AVW_FALSE* specifies that the entire histogram is set to zero before the occurrences are counted.

Histo will contain a count of all of the pixels of each intensity within *in_image.*

Histo is provided as a method of reusing an existing *AVW_Histogram.* Reuse is possible only if the size of the provided *histo* meets the requirements of the function. In this case the pointer to *histo* is returned by the function. If not reusable *histo* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_GetImageHistogram()* returns an *AVW_Histogram.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetImageHistogram() will fail if the following is true:

BADMAL
: Malloc Failed. Could not allocate memory for the results.

ILLIMG
: Specified image was not valid.

NOTSUP
: Data type not supported.

SEE ALSO

AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_GetVolumeHistogram(), AVW_NormalizeHistogram(), AVW_VerifyHistogram(), AVW_Histogram(), AVW_Image

NAME

AVW_GetImageIntensities – returns pixel intensities in a masked image

SYNOPSIS

#include "AVW.h"

AVW_PointValueList *AVW_GetImageIntensities(image, mask, mask_val, pvlist)
AVW_Image *image;
AVW_Image *mask;
int mask_val;
AVW_PointValueList *pvlist;

DESCRIPTION

AVW_GetImageIntensities() returns the X and Y coordinates and pixel value of the pixels in a masked region. If the mask is *NULL* the values are returned for every pixel in the *image.* The values are returned in an *AVW_PointValueList.*

RETURN VALUES

If successful, *AVW_GetImageIntensities()* returns an *AVW_PointValueList.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_GetPixel(), AVW_GetVoxel(), AVW_InterpolatedPixel(), AVW_NearestNeighborPixel(), AVW_PutVoxel(), AVW_GETBLUE(), AVW_GETGREEN(), AVW_GETRED(), AVW_Image, AVW_PointValueList

NAME

AVW_GetLiklihoods – retrieves liklihood data from most recent statistics based multispectral classification

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_GetLiklihoods(interleaved)
int interleaved;
```

DESCRIPTION

AVW_GetLiklihoods() retrieves the probability and liklihood data from the last multispectral classification performed with *AVW_ClassifyImage()* or *AVW_ClassifyVolume()* when the *autotype* parameter is set to one of the statistics based classification algorithms. *AVW_GAUSSIAN_CLUSTER*, *AVW_NEURAL_NETWORK*, or *AVW_PARZEN_WINDOWS*

Once an initial multispectral classification has been performed, the data in the returned liklihood volume can be used in a process of iterative relaxation with the additional AVW functions *AVW_UpdateConfidenceClasses()* and *AVW_UpdateImageClassification()*

to re-evaluate the classification of individual pixels based on the class assignments of neighboring pixels and the relative probabilities that a pixel belongs to each class.

When *Interleaved* is 1 the returned liklihood volume is interleaved; when 0 the the returned volume is not interleaved.

RETURN VALUES

If successful *AVW_GetVolume()* returns an *AVW_Volume.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetLiklihoods() will fail if one or more of the following are true:

BADMAL
: Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_GetScatLiklihoods(), AVW_UpdateConfidenceClasses(), AVW_UpdateImageClassification()

NAME

AVW_GetMaskBoundary – builds a boundary from a mask

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_GetMaskBoundary(in_mask, mask_value, out_boundary)
AVW_Image *in_mask;
int mask_value;
AVW_PointList2 *out_boundary;
```

DESCRIPTION

AVW_GetMaskBoundary() builds an *AVW_PointList2* which contains boundary points extracted from an *AVW_Image.*

All pixels of value *mask_value* in the *in_mask* are taken as belonging to a region. *AVW_GetMaskBoundary* constructs and returns a connected boundary circumscribing this region. *in_mask must be of data type*

Out_boundary is provided as a method of reusing an existing *AVW_PointList2.* Reuse is possible only if the size and data type of the provided *out_boundary* meet the requirements of the function. In this case the pointer to *out_boundary* is returned by the function. If not reusable *out_boundary* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_GetMaskBoundary()* returns an *AVW_PointList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AddPoint2(), AVW_DestroyPointList2(), AVW_FindImageEdges(), AVW_CreatePointList2(), AVW_MakeFPointList2()

NAME

AVW_GetMaskedImage – extracts an irregular area from an image

SYNOPSIS

```
#include      "AVW.h"

AVW_Image *AVW_GetMaskedImage( in_image, mask, out_image)
AVW_Image *in_image;
AVW_Image *mask;
AVW_Image *out_image;
```

DESCRIPTION

For each non-zero pixel in *mask*, a the corresponding value is copied from *in_image* to *out_image.* Each zero pixel in *mask,* results in a background pixel in the corresponding location in The background value is determined by a call to *AVW_MinimumDataValue().*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_GetMaskedImage()* returns an *AVW_Image.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetMaskedImage() will fail if one or more of the following are true:

ILLVOL
    Not a legal input volume.

BADMAL
    Couldn't allocate enough memory.

SEE ALSO

AVW_PutMaskedImage(), AVW_DestroyImage(), AVW_Image

NAME

AVW_GetMaskedVolume – extracts an irregular region from a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_GetMaskedVolume(in_volume, mask, out_volume)
AVW_Volume *in_volume;
AVW_Volume *mask;
AVW_Volume *out_volume;
```

DESCRIPTION

For each non-zero voxel in *mask*, a the corresponding value is copied from *in_volume* to *out_volume*. Each zero voxel in *mask*, results in a background voxel in the corresponding location in The background value is determined by a call to *AVW_MinimumDataValue()*.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_GetMaskedVolume()* returns an *AVW_Volume*. On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetMaskedVolume() will fail if one or more of the following are true:

ILLVOL
Not a legal input volume.

BADMAL
Couldn't allocate enough memory.

SEE ALSO

AVW_PutMaskedVolume(), AVW_DestroyVolume(), AVW_Volume

NAME

AVW_GetNumericInfo – gets a numeric value from an AVW information string

SYNOPSIS

```
#include "AVW.h"

double AVW_GetNumericInfo(match_string, info_string)
char *match_string, *info_string;
```

DESCRIPTION

AVW_GetNumericInfo() is used to retrieve a numeric value and from an *AVW* information string.

Match_string is a zero terminated string which must match an entry in an information string exactly in order to get, update, or remove information.

Info_string can be any zero terminated string. Each entry within the information string begins with a tag followed by an equal sign (=) and then the value. A space character is used as a seperator between entries. String information is enclosed in double quotes (") to allow for spaces within strings.

Info strings are used in AVW structures to carry optional additional information about the data that may not always be present and to allow the user to extend AVW structures to carry application dependant data.

The numeric value is returned as a double if the *match_string* entry is found within *info_string*. If the *match_string* is not found in the *info_string* 0.0 is returned. To verify whether a match was actually found in the *info_string* check if the the value of *AVW_ErrorNumber* is *NOMTCH*.

```
ysize = AVW_GetNumericInfo("VoxelHeight", img->Info);
if(AVW_ErrorNumber == NOMTCH)
  fprintf(stderr, "VoxelHeight not found in info_string");
else
  printf("Y Size=%f\n", ysize);
```

RETURN VALUES

If successful *AVW_GetNumericInfo()* returns a double numeric value. On failure it returns 0.0 and sets *AVW_ErrorNumber* to a value corresponding to the cause of the failure.

ERRORS

AVW_GetNumericInfo() will fail if one or more of the following are true:

NOMTCH
  No match string was found in the info_string.

BDMTCH
  Something is wrong with the info_string.

SEE ALSO

AVW_GetStringInfo(), AVW_PutHistoryInfo(), AVW_PutNumericInfo(), AVW_PutStringInfo(), AVW_RemoveInfo(), AVW_Image, AVW_ImageFile, AVW_Volume

NAME

AVW_GetObject – extracts an object from an object map

SYNOPSIS

```
#include "AVW_ObjectMap.h"

AVW_Volume *AVW_GetObject(object_map, object, out_volume)
AVW_ObjectMap *object_map;
int object;
AVW_Volume *out_volume;
```

DESCRIPTION

When supplied with a *AVW_ObjectMap* and object number, *AVW_GetObject()* extracts and returns the desired object in an *AVW_Volume.*

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_GetObject()* returns an *AVW_Volume.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetObject() will fail if one or more of the following are true:

ILLPAR
Specified *object* not in the *object_map.*

BADMAL
Couldn't allocate enough memory.

SEE ALSO

AVW_AddObject(), AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_PutObject(), AVW_LoadObjectMap(), AVW_SaveObjectMap(), AVW_ObjectMap, AVW_Volume

NAME

AVW_GetOblique – extracts an oblique image from a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_GetOblique(volume, matrix, interpolate, out_image)
AVW_Volume *volume;
AVW_Matrix *matrix;
int interpolate;
AVW_Image *out_image;
```

DESCRIPTION

AVW_GetOblique() is used to extract an oblique plane from an *AVW_Volume.* The identity matrix yields the middle slice from the XY volume plane.

The orientation of the plane may be specified by: directly manipulating the *matrix* with *AVW_TranslateMatrix(), AVW_RotateMatrix(), AVW_ScaleMatrix(), AVW_SetIdentityMatrix(),* or *AVW_MirrorMatrix();* or setting the *matrix* with *AVW_MakeMatrixFrom3Points()* or *AVW_MakeMatrixFromAxis().*

The common oblique manuevers may be accomplished using *AVW_RotateMatrix()* and *AVW_TranslateMatrix()* as follows:

ROLL

```
tmat=AVW_RotateMatrix(tmat,angle,0.,0.,tmat);
```

PITCH

```
tmat=AVW_RotateMatrix(tmat,0.,angle.,0.,tmat);
```

YAW

```
tmat=AVW_RotateMatrix(tmat,0.,0.,angle,tmat);
```

ELEVATE

```
tmat=AVW_TranslateMatrix(tmat,0.,0.,voxels,tmat);
```

SLIDE

```
tmat=AVW_TranslateMatrix(tmat,0.,voxels,0.,tmat);
```

SLIP

```
tmat=AVW_TranslateMatrix(tmat,voxels,0.,0.,tmat);
```

Example: Pitch 45 degrees

```
/* This example shows how two matrices are used
   to get an oblique image relative to the last
   one generated */

/* matrix represents the current orientation of
   the oblique image */

/* tmat represents the desired relative manuever */

tmat = AVW_CreateMatrix(4,4);
```

```
tmat = AVW_RotateMatrix(tmat, 45.0, 0., 0., tmat);

/* Applies relative manuever to the current orientation */

matrix = AVW_MultiplyMatrix(matrix, tmat, matrix);
AVW_Destroymatrix(tmat);

oblq_img = AVW_GetOblique(vol, matrix, iflag, oblq_img);
```

Interpolate determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

Out_image is provided as a method of reusing an existing *AVW_Image.* If *out_image* is NULL and image will be created using the Width and Height of the volume. Otherwise the returned image determines is determined by the size of *out_image.* In this way it is possible to generate small oblique images centered on a specific are of interest or large enough to include all voxels from the volume in the plane.

Reuse is possible only if data type of the provided *out_image* are the same as the volume. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_GetOblique()* returns an *AVW_Image* containing the specified oblique image. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetOblique() will fail if:

ILLVOL
Not a legal input volume.

SEE ALSO

AVW_GetCurved(), AVW_GetOrthogonal(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis(), AVW_RotateMatrix(), AVW_MirrorMatrix(), AVW_ScaleMatrix(), AVW_SetIdentityMatrix(), AVW_TranslateMatrix() AVW_PutOblique(), AVW_Image, AVW_Matrix, AVW_Volume

NAME

AVW_GetOrthogonal – extracts an orthogonal image from a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_GetOrthogonal(vol, orient, slice, out_image)
AVW_Volume *vol;
int orient;
int slice;
AVW_Image *out_image;
```

DESCRIPTION

When supplied with a *AVW_Volume, orient,* and *slice* number, *AVW_GetOrthogonal()* extracts and returns the desired orthogonal *AVW_Image.*

AVW numbers volume slices from 0 to (n-1). This means that acceptable slice values for Transverse slices are 0 to (vol->Depth-1). Coronal slices number from 0 to (vol->Width-1) and Sagittal slices number from 0 to (vol->Height-1).

ORIENTATION	SLICE RANGE	OUTPUT SIZE
AVW_TRANSVERSE	0 - (vol->Depth-1)	vol->Width X vol->Height
AVW_CORONAL	0 - (vol->Height-1)	vol->Width X vol->Depth
AVW_SAGITTAL	0 - (vol->Width-1)	vol->Height X vol->Depth

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_GetOrthogonal()* returns an *AVW_Image.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetOrthogonal() will fail if one or more of the following are true:

ILLVOL
Not a legal input volume.

BADMAL
Couldn't allocate enough memory.

SEE ALSO

AVW_GetCurved(), AVW_GetOblique(), AVW_PutOrthogonal(), AVW_Image, AVW_Volume

NAME

AVW_GetOutsideEdges – builds a set of edges from a mask

SYNOPSIS

```
#include "AVW.h"

AVW_MultiList2 *AVW_GetOutsideEdges(in_mask, mask_value, connectivity, Out_boundaries)
AVW_Image *in_mask;
int mask_value;
int connectivity;
AVW_MultiList2 *Out_boundaries;
```

DESCRIPTION

AVW_GetOutsideEdges() builds an *AVW_MultiList2* which contains a set of all edges with an *AVW_Image* at a given *mask_value.*

All pixels of value *mask_value* in the *in_mask* are taken as belonging to a region. *AVW_GetOutsideEdges* constructs and returns a set of connected boundaries circumscribing this region. *in_mask must be of data type*

Out_boundaries is provided as a method of reusing an existing *AVW_MultiList2.* Reuse is possible only if the size and data type of the provided *Out_boundaries* meet the requirements of the function. In this case the pointer to *Out_boundaries* is returned by the function. If not reusable *Out_boundaries* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_GetOutsideEdges()* returns an *AVW_MultiList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_DestroyMultiList2(), AVW_CreateMultiList2(), AVW_MultiList2,

NAME

AVW_GetPixel – returns pixel intensity at a location in an image

SYNOPSIS

```
#include "AVW.h"

double AVW_GetPixel(image, point)
AVW_Image *image;
AVW_Point2 *point;
```

DESCRIPTION

AVW_GetPixel() returns the value found at the location in *image* specified by *point.*

Note that fast access to the pixel value is available by referencing the image memory directly. In large loops, direct manipulation of a pointer to the data memory will yield the fastest results.

Example 1:

```
  register unsigned char *rmem;
  rmem = image->Mem;
  value = *(rmem + image->YTable[point->Y] + point->X);
```

Example 2:

```
  register unsigned char *rmem;
  rmem = image->Mem;
  num = image->PixelsPerImage;
  while(num--)
    {
    value = *rmem;
    rmem++;
    }
```

For an *AVW_COLOR* image, the red, green and blue components of the pixel are packed into the returned value.

```
  value = red << 16 | green << 8 | blue
```

RETURN VALUES

If successful, *AVW_GetPixel()* returns the pixel value at *point* in the *AVW_Image.* On failure, it returns *0.0* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetPixel() will fail if one or more of the following are true:

ILLPAR
: Point specified is outside of the image.

ILLDT
: Illegal datatype.

SEE ALSO

AVW_GetVoxel(), AVW_PutPixel(), AVW_InterpolatedPixel(), AVW_NearestNeighborPixel(), AVW_PutVoxel(), AVW_GETBLUE(), AVW_GETGREEN(), AVW_GETRED(), AVW_Image, AVW_Point2

NAME

AVW_GetPoint2 – get a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_GetPoint2(trace, which_point, point)
AVW_PointList2 *trace;
int which_point;
AVW_Point2 *point;
```

DESCRIPTION

AVW_GetPoint2() gets a specific point from a point list.

Trace is an *AVW_PointList2.*

Which_point specifies the point to get from the list. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetPoint2 will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_GetPoint3(), AVW_GetFPoint2(), AVW_GetFPoint3(), AVW_GetIPoint2(), AVW_GetIPoint3(), AVW_GetPointValue(), AVW_AddPoint2(), AVW_CreatePointList2(), AVW_RemovePoint2(), AVW_Point2, AVW_PointList2

NAME

AVW_GetPoint3 – get a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_GetPoint3(trace, which_point, point)
AVW_FPointList3 *trace;
int which_point;
AVW_Point3 *point;
```

DESCRIPTION

AVW_GetPoint3() gets a specific point from a point list.

Trace is an *AVW_PointList3.*

Which_point specifies the point to get from the list. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetPoint3 will fail if the following is true:

ILLPAR
: *which_value* is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_GetPoint2(), AVW_GetFPoint2(), AVW_GetFPoint3(), AVW_GetIPoint2(), AVW_GetIPoint3(), AVW_GetPointValue(), AVW_AddPoint3(), AVW_CreatePointList3(), AVW_RemovePoint3(), AVW_Point3, AVW_FPointList3

NAME

AVW_GetPointValue – get a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_GetPointValue(trace, which_point, point, value)
AVW_PointValueList *trace;
int which_point;
AVW_Point2 *point;
double *value;
```

DESCRIPTION

AVW_GetPointValue() gets a specific point and value from a point list.

Trace is an *AVW_PointValueList.*

Which_point specifies the point and value to get from the list. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetPointValue will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_GetFPoint2(), AVW_GetFPoint3(), AVW_GetIPoint2(), AVW_GetIPoint3(), AVW_GetPoint2(), AVW_GetPoint3(), AVW_AddPointValue(), AVW_CreatePointValueList(), AVW_RemovePointValue(), AVW_PointValue, AVW_PointValueList

NAME

AVW_GetScatLiklihoods – retrieves liklihood data from most recent scattergram classification

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_GetScatLiklihoods(in_img1, in_img2, climg, interleaved)
AVW_Image *in_img1;
AVW_Image *in_img2;
AVW_Image *climg;
int interleaved;
```

DESCRIPTION

AVW_GetScatLiklihoods() retrieves the probability and liklihood data from the last scattergram classification performed with *AVW_ClassifyScattergram()* when the *autotype* parameter is set to one of the statistics based classification algorithms:
AVW_GAUSSIAN_CLUSTER, *AVW_NEURAL_NETWORK*, or
AVW_PARZEN_WINDOWS

When *Interleaved* is 1 the returned liklihood volume is interleaved; when 0 the the returned volume is not interleaved.

RETURN VALUES

If successful *AVW_GetVolume()* returns an *AVW_Volume*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_GetLiklihoods() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_GetLiklihoods(), AVW_UpdateConfidenceClasses(), AVW_UpdateImageClassification()

NAME

AVW_GetStringInfo – gets a string from an AVW information string

SYNOPSIS

```
#include "AVW.h"

char *AVW_GetStringInfo(match_string, info_string)
char *match_string, *info_string;
```

DESCRIPTION

AVW_GetStringInfo() is used to retrieve a text string from an *AVW* information string.

Match_string is a zero terminated string which must match an entry in an information string exactly in order to get information.

Info_string can be any zero terminated string. Each entry within the information string begins with a tag followed by an equal sign (=) and then the value. A space character is used as a seperator between entries. String information is enclosed in double quotes (") to allow for spaces within strings.

Info strings are used in AVW structures to carry optional additional information about the data that may not always be present and to allow the user to extend *AVW* structures to carry application dependant data.

The string is returned if the *match_string* entry is found within *info_string.* If the *match_string* is not found in the *info_string NULL* is returned.

Example:

```
img->Info = AVW_PutStringInfo("Name", "John Doe", img->Info);
printf("%s\n", img->Info);
if((name = AVW_GetStringInfo("Name", img->Info)) == NULL)
  fprintf(stderr, "Name not found in info_string");
  else
  {
  printf("Name=%s\n", name);
  AVW_Free(name);
  }
```

Results:

```
Name="John Doe"
Name=John Doe
```

RETURN VALUES

If successful *AVW_GetStringInfo()* returns a pointer to a character string. AVW_Free() must be called to free this string when it is no longer needed. On failure it returns *NULL* and sets *AVW_ErrorNumber* to a value corresponding to the cause of the failure.

ERRORS

AVW_GetStringInfo() will fail if one or more of the following are true:

NOMTCH
: No match string was found in the info_string.

SEE ALSO

AVW_GetNumericInfo(), AVW_PutHistoryInfo(), AVW_PutNumericInfo(), AVW_PutStringInfo(), AVW_RemoveInfo(), AVW_Image, AVW_ImageFile, AVW_Volume

NAME

AVW_GetSubImage – extracts a sub-region from an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_GetSubImage(input_image, region, out_image)
AVW_Image *input_image;
AVW_Rect2 *region;
AVW_Image *out_image;
```

DESCRIPTION

AVW_GetSubImage() extracts a subregion defined by *region* from an *input_image.*

With (0, 0) as the first pixel in *input_image, region->PointA.X, region->PointA.Y, region->PointB.X,* and *region->PointB.Y* specify the rectangular region to be extracted.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_GetSubImage()* returns an *AVW_Image.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetSubImage() will fail if one or more of the following are true:

ILLPAR
: Illegal subregion.

BADMAL
: Couldn't allocate enough memory.

SEE ALSO

AVW_GetSubImageWithIncrements(), AVW_GetSubVolume(), AVW_PadImage(), AVW_PutSubImage(), AVW_Rect2, AVW_Image

NAME

AVW_GetSubImageWithIncrements – extracts a sub-region from an image (with increments)

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_GetSubImageWithIncrements(input_image, region, xinc, yinc, out_image)
AVW_Image *input_image;
AVW_Rect2 *region
int xinc, yinc;
AVW_Image *out_image;
```

DESCRIPTION

AVW_GetSubImageWithIncrements() extracts a subregion defined by *region* from an *input_image.*

With (0, 0) as the first pixel in *input_image, region->PointA.X, region->PointA.Y, region->PointB.X,* and *region->PointB.Y* specify the rectangular region to be extracted.

Xinc and *yinc* specify that only every *xinc* th and *yinc* th pixel is extracted.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_GetSubImageWithIncrements()* returns an *AVW_Image.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetSubImageWithIncrements() will fail if one or more of the following are true:

ILLPAR
Illegal subregion.

BADMAL
Couldn't allocate enough memory.

SEE ALSO

AVW_GetSubImage(), AVW_Rect2, AVW_Image

NAME

AVW_GetSubVolume – extracts a sub-volume from a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_GetSubVolume(input_volume, region, out_volume)
AVW_Volume *input_volume;
AVW_Rect3 *region;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_GetSubVolume() extracts a subregion defined by *region* from an *input_volume.*

With (0, 0, 0) as the first voxel of *input_volume, region->PointA.X, region->PointA.Y, region->PointA.Z, region->PointB.X, region->PointB.Y,* and *region->PointB.Z* specifies the region to be extracted.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_GetSubVolume()* returns an *AVW_Volume.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetSubVolume() will fail if one or more of the following are true:

ILLPAR
Illegal subregion.

BADMAL
Couldn't allocate enough memory.

SEE ALSO

AVW_GetSubImage(), AVW_PadVolume(), AVW_PutSubVolume(), AVW_Volume, AVW_Rect3

NAME

AVW_GetThresholdedBoundary – finds the boundary of a thresholded region

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_GetThresholdedBoundary(image, thresh_max, thresh_min,
                        seed_point, trace)
AVW_Image *image;
double thresh_max, thresh_min;
AVW_Point2 *seed_point;
AVW_PointList2 *trace;
```

DESCRIPTION

AVW_GetThresholdedBoundary() finds and returns an *AVW_PointList2*, *trace*, which contains the boundary of the region defined by the *seed_point*, *threshold_max*, and *threshold_min*. The region is grown from the *seed_point* and includes all pixels which are connected via four neighbors to it and within the threshold values.

Trace is provided as a method of reusing an existing *AVW_PointList2*.

RETURN VALUES

If successful *AVW_GetThresholdedBoundary()* returns an *AVW_PointList2*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetThresholdedBoundary() will fail if:

ILLDT
Data type is not defined or supported.

ILLIMG
An illegal image was passed to the function.

SEE ALSO

AVW_GetBoundaryAndDelete(), AVW_GetMaskBoundary(), AVW_ThresholdImage(), AVW_PointList2, AVW_Point2, AVW_Image

NAME

AVW_GetVolumeChannel – extracts a specific channel from an AVW_Color volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_GetVolumeChannel(in_volume, channel, out_volume)
AVW_Volume *in_volume;
int channel;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_GetVolumeChannel() returns the contents of a specified *channel* within the *AVW_COLOR* volume, *in_volume.*

Channel can be specified as any of the following:

AVW_RED_CHANNEL - specifies the red channel.

AVW_GREEN_CHANNEL - specifies the green channel.

AVW_BLUE_CHANNEL - specifies the blue channel.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_GetVolumeChannel()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetVolumeChannel() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_GetImageChannel(), AVW_MakeColorVolume(), AVW_DestroyVolume() AVW_Volume()

NAME

AVW_GetVolumeHistogram – computes a volume histogram

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Histogram *AVW_GetVolumeHistogram(in_volume, mask_volume, mask_value, histo)
AVW_Volume *in_volume;
AVW_Volume *mask_volume;
int mask_value;
AVW_Histogram *histo;
```

DESCRIPTION

AVW_GetVolumeHistogram() computes the histogram, *histo*, of *in_volume.* The *mask_volume* may be used to specify a region of the volume for which the histogram is to be computed. Only voxels from the *in_volume* corresponding to voxels of the *mask_volume* equal to the *mask_value* will be used in the histogram computation. If the *mask_volume* is equal to *NULL*, *mask_volume* and *mask_value* are ignored and the entire *in_volume* is used to compute the histogram.

Histo will contain a count of all of the pixels of each intensity within *in_volume.*

Histo is provided as a method of reusing an existing *AVW_Histogram*. Reuse is possible only if the size of the provided *histo* meets the requirements of the function. In this case the pointer to *histo* is returned by the function. If not reusable *histo* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_GetVolumeHistogram()* returns an *AVW_Histogram.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_GetVolumeHistogram() will fail if the following is true:

BADMAL
Malloc Failed. Could not allocate memory for the results.

ILLVOL
Specified volume was not valid.

NOTSUP
Data type not supported.

SEE ALSO

AVW_GetImageHistogram(), AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_NormalizeHistogram(), AVW_VerifyHistogram(), AVW_Volume, AVW_Histogram

NAME

AVW_GetVoxel – returns the voxel intensity at a location in a volume

SYNOPSIS

```
#include "AVW.h"

double AVW_GetVoxel(volume, point)
AVW_Volume *volume;
AVW_Point3 *point;
```

DESCRIPTION

AVW_GetVoxel() returns the value found at the location in *volume* specified by *point.* Note that fast access to the voxel value is available by referencing the volume memory directly. In large loops, direct manipulation of a pointer to the data memory will yield the fastest results.

Example 1:

```
register unsigned char *rmem;
rmem = volume->Mem;
value = *(rmem + volume->ZTable[point->Z] + volume->YTable[point->Y] +
    point->X);
```

Example 2:

```
register unsigned char *rmem;
rmem = volume->Mem;
num = volume->VoxelsPerVolume;
while(num--)
  {
  value = *rmem;
  rmem++;
  }
```

For an *AVW_COLOR* image, the red, green and blue components of the voxel are packed into the returned value.

```
value = red << 16 | green << 8 | blue
```

RETURN VALUES

If successful, *AVW_GetVoxel()* returns the voxel value at *point* in the *AVW_Volume.* On failure, it returns *0.0* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. To insure valid results *AVW_ErrorNumber* should be checked after calling this function.

ERRORS

AVW_GetVoxel() will fail if one or more of the following are true:

ILLPAR
: Point specified is outside of the volume.

ILLDT
: Illegal datatype.

SEE ALSO

AVW_GetPixel(), AVW_PutPixel(), AVW_InterpolatedVoxel(), AVW_NearestNeighborVoxel(), AVW_PutVoxel(), AVW_GETBLUE(), AVW_GETGREEN(), AVW_GETRED(), AVW_Point3, AVW_Volume

NAME

AVW_HomotopicThickenImage – performs homtopic thickening on an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_HomotopicThickenImage(in_image, cond_image, iterations, out_image)
AVW_Image *in_image;
AVW_Image *cond_image;
int iterations;
AVW_Image *out_image;
```

DESCRIPTION

AVW_HomotopicThickenImage() performs 2D homotopic thickening on *in_image.* *Cond_image* is used as a conditioning image which limits the areas into which the image may be thickened. If *cond_image* is equal to *NULL* the thickening will not be constrained. The image is thickened *iterations* times or until no more changes can be made. If *iterations* is less than or equal to zero the image will be thickened until only a single pixel border seperates distinct objects or the thickened image matches *cond_image.* The thickened image is returned in *out_image.*

In_image has to be a binary valued, i.e. ones and zeroes. *In_image,* and *out_image* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_HomotopicThickenImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_HomotopicThickenImage() will fail if:

ILLDT
    Data type is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_Thin2D(), AVW_HomotopicThickenVolume(), AVW_Image

NAME

AVW_HomotopicThickenVolume – performs homtopic thickening on an volume

SYNOPSIS

```
#include    "AVW.h"

AVW_Volume *AVW_HomotopicThickenVolume(in_volume, cond_volume, iterations, out_volume)
AVW_Volume *in_volume;
AVW_Volume *cond_volume;
int iterations;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_HomotopicThickenVolume() performs 2D homotopic thickening on *in_volume.* *Cond_volume* is used as a conditioning volume which limits the areas into which the volume may be thickened. If *cond_volume* is equal to *NULL,* the thickening will not be constrained. The volume is thickened *iterations* times or until no more changes can be made. If *iterations* is less than or equal to zero the volume will be thickened until only a single pixel border seperates distinct objects or the thickened volume matches *cond_volume.* The thickened volume is returned in *out_volume.*

In_volume has to be a binary valued, i.e. ones and zeroes. *In_volume,* and *out_volume* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_HomotopicThickenVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_HomotopicThickenVolume() will fail if:

ILLDT
    Data type is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_Thin3D(), AVW_HomotopicThickenImage(), AVW_Volume

NAME

AVW_ImageOpConstant – transforms an image mathematically

SYNOPSIS

```
#include "AVW_Parse.h"

AVW_Image *AVW_ImageOpConstant(in_image, operation, value, out_image)
AVW_Image *in_image;
int operation;
double value;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ImageOpConstant() applies the *operation* and *value* to *in_image* and returns the resulting image:

output = *in_image operation value*

The following operations are defined in *AVW_Parse.h:*

AVW_OP_ADD AVW_OP_SUB AVW_OP_MUL AVW_OP_DIV AVW_OP_LT AVW_OP_GT AVW_OP_LE AVW_OP_GE AVW_OP_EQ AVW_OP_NE AVW_OP_AND AVW_OP_OR AVW_OP_MOD

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ImageOpConstant()*, returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ImageOpConstant() will fail if one or more of the following are true:

NOTSUP
: Operation is not supported.

DIVZER
: Division by zero.

BADMAL
: Memory could not be allocated for results.

ILLDT
: Data type is not defined or supported.

SEE ALSO

AVW_ConstantOpImage(), AVW_ImageOpImage(), AVW_VolumeOpConstant(), AVW_FunctionImage(), AVW_Image

NAME

AVW_ImageOpImage – transforms an image mathematically

SYNOPSIS

```
#include "AVW_Parse.h"

AVW_Image *AVW_ImageOpImage(in_image1, operation, in_image2, out_image)
AVW_Image *in_image1;
int operation;
AVW_Image *in_image2;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ImageOpImage() applies the *operation* and *in_image2* to *in_image1* and returns the resulting image:

output = *in_image1 operation in_image2*

The following operations are defined in *AVW_Parse.h:*

AVW_OP_ADD

AVW_OP_SUB

AVW_OP_MUL

AVW_OP_DIV

AVW_OP_LT

AVW_OP_GT

AVW_OP_LE

AVW_OP_GE

AVW_OP_EQ

AVW_OP_NE

AVW_OP_AND

AVW_OP_OR

AVW_OP_MOD

Out_image is provided as a method of reusing an existing *AVW_Image* or *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ImageOpImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ImageOpImage() will fail if one or more of the following are true:

NOTSUP
Operation is not supported.

DIVZER
Division by zero.

BADMAL
Memory could not be allocated for results.

ILLDT
Data type is not defined or supported.

CFLSZ
Input images conflict in size.

SEE ALSO

AVW_ConstantOpImage(), AVW_ImageOpConstant(), AVW_VolumeOpVolume(), AVW_FunctionImage(), AVW_DestroyImage(), AVW_Image

NAME AVW_ImageSampleEntropy – Calculate the entropy of a sample of voxels

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

double AVW_ImageSampleEntropy(image,points,interpolate)
AVW_Image *image;
AVW_FPointList2 *points;
int interpolate;
```

DESCRIPTION

AVW_ImageSampleEntropy() calculates the entropy of a sample of pixels defined by a list of floating-point 2-D coordinates.

Interpolate determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

RETURN VALUES

If successful returns the entropy.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ImageSampleEntropy() will fail if the following is true:

ILLIMG
Illegal Image.

ILLPAR
Interpolation type is not recognized.

SEE ALSO

AVW_ImageSampleJointEntropy(), AVW_ImageSampleNMI(), AVW_VolumeSampleEntropy().

NAME

AVW_ImageSampleJointEntropy – Calculate the joint entropy of a sample of pixels from two images

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

double AVW_ImageSampleJointEntropy(base,match,points,matrix,interpolate)
AVW_Image *base,*match;
AVW_FPointList2 *points;
AVW_Matrix *matrix;
int interpolate;
```

DESCRIPTION

AVW_ImageSampleJointEntropy() calculates the joint entropy of a sample of pixels defined by a list of floating-point 2-D coordinates. *points* defines a set of 2-D coordinates in the *match* image to be sampled. Those pixel values are paired with those from the same coordinates transformed by *matrix* in the *base* image. The joint entropy of the samples is returned

Interpolate determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

RETURN VALUES

If successful returns the joint entropy.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ImageSampleJointEntropy() will fail if the following is true:

ILLIMG
Illegal Image. Floating point, Complex, and color images are not supported

BADMAL
Unable to allocate memory for internal calculations.

ILLPAR
Interpolation type is not recognized.

SEE ALSO

AVW_VolumeSampleJointEntropy(), AVW_ImageSampleNMI(), AVW_ImageSampleEntropy().

NAME

AVW_ImageSampleNMI – Calculate the Normalized Mutual Information of a sample of pixels from two images

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

double AVW_ImageSampleNMI(base,match,points,matrix,interpolate)
AVW_Image *base,*match;
AVW_FPointList2 *points;
AVW_Matrix *matrix;
int interpolate;
```

DESCRIPTION

AVW_ImageSampleNMI() calculates the normalized mutual information of a sample of pixels defined by a list of floating-point 2-D coordinates. *points* defines a set of 2-D coordinates in the *match* image to be sampled. Those pixel values are paired with those from the same coordinates transformed by *matrix* in the *base* image. The normalized mutual information (sum of individual image entropies divided by joint entropy) of the samples is returned

Interpolate determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

RETURN VALUES

If successful returns the normalized mutual information.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ImageSampleNMI() will fail if the following is true:

ILLIMG
Illegal Image. Floating point, Complex, and color images are not supported

BADMAL
Unable to allocate memory for internal calculations.

ILLPAR
Interpolation type is not recognized.

SEE ALSO

AVW_ImageSampleEntropy(), AVW_ImageSampleJointEntropy(), AVW_VolumeSampleNMI().

NAME

AVW_InhomogeneityCorrectVolume – performs an inhomogeneity correction filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_InhomogeneityCorrectVolume(in_volume, mask_volume,
                                mask_value, window_size,
                                out_volume)
AVW_Volume *in_volume, *mask_volume;
int mask_value, window_size;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_InhomogeneityCorrectVolume() applies mean-based homomorphic filtering to the specified *in_volume* and returns the filtered volume in This routine scans through the volume in a slice by slice fashion and adjusts voxel intensity values by the following formula:

Io = Ii * global_mean / local_mean

Io is the intensity of the filtered voxel and Ii is the intensity of the input voxel. The global mean is calculated as the average voxel intensity value for all voxels corresponding to voxels equal to *mask_value* in *mask_volume.*

The *mask_volume* is used to exclude noise voxels. It must be the same size as *in volume* and be of data type *AVW_UNSIGNED_CHAR.* The *mask_volume* can be a thresholded version of *in_volume* or another spatially registered volume. *Mask_value* specifies the value of the voxels in *mask_volume* whose corresponding voxels in *in_volume* will be used in the mean calculations and eventually corrected in *out_volume.*

The algorithm uses a roving square window with length and width equal to the *window_size.* The algorithm centers the window around each voxel and calculates the mean value of all voxels in the window corresponding to voxels equal to *mask_value* in *mask_volume.* This value serves as the local mean for the intensity correction. Voxels which do not correspond to voxels equal to *mask_value* in *mask_volume* are not included in the filter calculations, nor are these voxels corrected.

This algorithm is useful for removing low-frequency grayscale gradients from images of all types. Examples of these grayscale gradients include uneven illumination in microscope images and uneven coil coverage (for example, as is seen with surface coil images) in magnetic resonance imaging.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RECCOMMENDED PARAMETERS

For magnetic resonance images:

Mask_volume: The *mask_volume* is created by thresholding an MRI volume to exclude background noise. It is recommended that this value be chosen by interactively thresholding the volume and choosing the threshold value that best separates the object of interest from the background. *Mask_value* can be set to 1 when using a thresholded volume.

Window Dimensions: The optimal *window_size* depends upon the strength and spatial frequency of the unwanted intensity variations in the image. For a strong but smoothly-varying gradient a 65x65 window is recommended. For a more subtle gradient a larger window (95x95) should be used. For more complex inhomogeneities, such as are found in surface coil images, smaller windows (45x45) are recommended.

For further information on this algorithm, see the following article:

Brinkmann, Manduca, Robb. "Optimized Homomorphic Unsharp Masking for MR Grayscale Inhomogeneity Correction" IEEE Trans Med Img, April, 1998. vol. 17(2): 161-171.

RETURN VALUES

If successful *AVW_InhomogeneityCorrectVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterImage(), AVW_RankFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_SobelFilterVolume(), AVW_ThresholdVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_InitializeMatchVoxelParams – initializes match voxel parameters

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_MatchVoxelParams *AVW_InitializeMatchVoxelParams(param)
AVW_MatchVoxelParams *param;
```

DESCRIPTION

AVW_InitializeMatchVoxelParams() creates and returns a structure required to match two specified *AVW_Volume.* This structure is passed to *AVW_MatchVoxels()* to obtain the matching transformation between the two specified volumes.

Parameters of *AVW_InitializeMatchVoxelParams():*

> *param* specifies the *AVW_MatchVoxelParams* to be created or resets. If *param* is not NULL the function resets all of the *AVW_MatchVoxelParams* values to their defaults.

The following elements of the *AVW_MatchVoxelParams* structure are set by *AVW_InitializeMatchVoxelParams()* when the structure is created or resets.

> *param->Ftol* determines the convergence tolerance for the search strategy. If the change in the cost-function is smaller than this value, the search will stop assuming this is the minimum.
>
> *param->Ptol* specifies the transformation parameters convergence tolerance. That is, if the total change in all of the 6 transformation parameters (X,Y,Z rotations and translations) is less than Ptol for a number of successive iterations the subroutine will terminate the search.
>
> *param->Interpolate* specifies whether the transformed image will be computed with bilinear (AVW_TRUE) or nearest neighbor (AVW_FALSE) interpolation.
>
> *param->Smpl1to1, param->Smpl2to1, param->Smpl4to1* and *param->Smpl8to1* Specifies sampling in the X, Y and Z directions. (i.e., if X and Y are set to 3, and Z is 1, than the calculation of the cost function will use every third voxel in the X and Y directions, and all of the slices in the Z direction.
>
> The search is done in stages, first on a volume scaled to a size of 8:1, than 4:1, 2:1 and finally 1:1. The sampling values can be specified for each one of these scaling stages (Smpl8to1, Smpl4to1, Smpl2to1 and Smpl1to1 respectively) . If scaling to a certain size would cause a the volume to become to small the stage will be skipped. It is the users responsibility to assign reasonable values for sampling. Values which cause the use of only a very small number of voxels, will lead to non-accurate results. To determine a good experimental value, define the sample in such a way that there is 30 to 50 points in each direction. This will usually lead to good results with the best possible computation time.
>
> *param->InitGuess* specifies the initial position (X,Y,Z rotation and translations) of the match volume.
>
> *param->SearchLength* defines the problem characteristic scale in X, Y, Z rotation and translation. Theses parameters limit the distance of the search algorithm.

RETURN VALUES

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_InitializeMatchVoxelParams() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

SEE ALSO

AVW_MatchVoxelsParams, AVW_DestroyMatchVoxelParams(), AVW_MatchVoxels(), AVW_Volume.

NAME

AVW_InitializeRPParam – initializes surface tiling parameters

SYNOPSIS

```
#include    "AVW_Model.h"

AVW_RPParam *AVW_InitializeRPParam(volume, last_rp_param)
AVW_Volume *volume;
AVW_RPParam *last_rp_param;
```

DESCRIPTION

AVW_InitializeRPParam() creates and returns a structure required to extract a set of ribbon contours from the specified *AVW_Volume.* This structure is passed to *AVW_SliceVolume()* to obtain the contours from the volume and to write them to disk. The *last_rp_param* structure may be specified as a starting point and only required changes will be made for the returned structure.

Parameters of *AVW_InitializeRPParam():*

volume specifies the *AVW_Volume* to use during contour extraction.

last_rp_param specifies a starting set of paramaters. This structure is most likely the results of a previous call to *AVW_InitializeRPParam().* Only parameters which require modification as the results of an *AVW_Volume* change will be modified in the structure returned. If *last_rp_param is equal to NULL a AVW_RPParam* structure will be created.

The following elements of the *AVW_RPParam* structure are set by *AVW_InitializeRPParam()* when the structure is created.

param->Format specifies the output format. Supported values include:

AVW_HPGL_SURFACE – A modified form of HPGL plotter commands designed to support rapid prototyping or stereolithography machines
AVW_POGO_SURFACE() – A compressed version of the binary SLC format. This format does not differentiate between internal and external boundaries.
AVW_SLC_SURFACE() – The standard binary SLC format.
AVW_SSD_ASCII_SURFACE() – The ASCII version of the standard Analyze SSD format.

param->SubvolumeFlag If set, will subvolume the dataset so that the object of interest is contained within a minimum enclosing volume. This may cause the data to be reoriented prior to contour extraction.

param->MaskValue specify the mask value of the object within the *AVW_Volume* whose contours are to be extracted.

param->InterpolateFlag if set, will subvolume the dataset using trilinear interpolation.

param->Orientation specifies which orthogonal axis will be used during contour extraction This parameter may be one of:
AVW_TRANSVERSE [default], AVW_CORONAL, or *AVW_SAGITTAL*

param->AngleResolution specifies the angle resolution used while determining the minimum enclosing box.

RETURN VALUES

If successful *AVW_InitializeRPParam()* returns a pointer to a *AVW_RPParam* structure. This structure contains the parameters which may be modified to produce the desired contour file. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_InitializeRPParam() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_SliceVolume(), AVW_DestroyRPParam() AVW_Volume AVW_RPParam

NAME

AVW_InitializeRenderParameters – initializes volume rendering parameters

SYNOPSIS

```
#include "AVW_Render.h"

AVW_RenderParameters *AVW_InitializeRenderParameters(volume,
        object_map, last_param)
AVW_Volume *volume;
AVW_ObjectMap *object_map;
AVW_RenderParameters *last_param;
```

DESCRIPTION

AVW_InitializeRenderParameters() creates and returns a structure required to render the specified *AVW_Volume.* This structure is passed to *AVW_RenderVolume()* to obtain 3D renderings of the volume. If an *AVW_ObjectMap* is specified it will be used in the rendering process also. The *last_param* structure may be specified as a starting point and only required changes will be made for the returned structure.

Parameters of *AVW_InitializeRenderParameters():*

Volume specifies the *AVW_Volume* to be rendered.

Object_map specifies a map of objects defined within the *AVW_Volume.* This map must be exactly the same dimensions as the *AVW_Volume* to be rendered. This map is commonly the result of a call to *AVW_LoadObjectMap()* or *AVW_CreateObjectMap().* An *AVW_ObjectMap* is not required, *NULL* may be specified to indicate that no object map is to be used.

Last_param specifies a starting set of paramaters. This structure is most likely the results of a previous call to *AVW_InitializeRenderParameters().* Only parameters which require modification as the results of an *AVW_Volume* or *AVW_ObjectMap* change will be modified in the structure returned. If *last_parm is equal to NULL a AVW_RenderParameters* structure will be created.

The following elements of the *AVW_RenderParameters* structure are set by *AVW_InitializeRenderParameters()* when the structure is created.

param->Type specifies the type of ray casting preformed. Supported values include:

AVW_DEPTH_SHADING – The value of each output pixel is a function of depth only. The depth of the first voxel found along the ray path is used to determine the brightness of that pixel. Closer voxels will appear brighter than distant voxels. This output image may be further enhanced by *AVW_ProcessZGradients().*

AVW_GRADIENT_SHADING – The gray scale gradient vector is computed using a 3D neighborhood about the surface voxel. The value projected at each output location is the dot product of the gradient vector and an independantly specified light source vector. This simulates the appearance of a reflective surface under uniform-field illumination. [default]

AVW_VOLUME_COMPOSITING – The volumetric compositing algorithm integrates the gradient-shaded value of all voxels along the ray path. The contribution of each gradient-shaded voxel value is weighted by an opacity function described in the *AVW_CompositeInfo* structure. Voxel intensity is used to determined the voxels color and opacity contribution during the ray casting.

AVW_MAX_INTENSITY_PROJECTION – The maximum voxel intensity along the ray path is used.

AVW_SUMMED_VOXEL_PROJECTION – The average of all voxels along the ray path is used.

AVW_SURFACE_PROJECTION – Once a voxel within the threshold limits is detected, the average of the next N voxels is computed.

AVW_TRANSPARENCY_SHADING – Available only when an *AVW_ObjectMap* has been specified, this type produces a 24-bit true color projections of all the surface gradients along the ray path. Opacity parameters may be specified for each object controlling the transparency.

AVW_DELETE_VOXELS – Not really a rendering type, but causes the ray casting algorithm to delete (or convert to a specified value) voxels along the ray path. See the *param->DeleteDepth* and *param->DeleteValue* parameters for more information.

param->ThresholdMinimum and *param->ThresholdMaximum* specify the range of acceptable voxel values. Voxels outside the specified range are ignored. [Defaults are the minimum and maximum of the volume.]

param->ClipLowX, *param->ClipLowY*, *param->ClipLowZ*, *param->ClipHighX*, *param->ClipHighY*, and *param->ClipHighZ* specify the sub volume of the *AVW_Volume* to render. [Default = is the entire volume]

param->ClipPlaneMinimum and *param->ClipPlaneMaximum* specify the starting and ending depths for the ray casting process.

param->ClipShading specifies the type of shading that is used when the ray casting process begins at a voxel which is within the threshold range (and object enabled). The value is only used when the render *Type* is set to *AVW_GRADIENT_SHADING*. Possible values are: *AVW_CLIP_SHADED*, *AVW_CLIP_ACTUAL*, *AVW_CLIP_REMOVE_AND_RENDER*, and *AVW_CLIP_RENDER_AS_IS*. [Default = *AVW_CLIP_SHADED]*

param->RenderWidth, *param->RenderHeight*, and *param->RenderDepth* specify the size of the rendered space. By default these values are set to the maximum of the X, Y, or Z input dimension.

param->MaximumPixelValue and *param->MinimumPixelValue* specify the range of possible output values for the reflectance renderings. Transmission renderings use the maximum and minimum values from the *AVW_Volume* as the output range. [Default = 250 and 0]

param->SurfaceThickness specifies the maximum thickness for the *AVW_SURFACE_PROJECTION* rendering type. [Default = 5]

param->SurfaceSkip specify the number of voxels to skip before summing begins for the *AVW_SURFACE_PROJECTION* rendering type. [Default = 0]

param->MIP_Weight specify the weighting options used during a *AVW_SURFACE_PROJECTION*. Options include: *AVW_NO_WEIGHTING*, *AVW_WEIGHT_BEFORE*, and *AVW_WEIGHT_AFTER*. *AVW_WEIGHT_BEFORE* indicates that before a voxel is checked to see if it is the maximum value, a weighting factor is applied. The weighting factor is determined by dividing the length of the ray left to cast, by it's total length. AVW_WEIGHT_AFTER determines the maximum voxel along the entire ray casting path and then applies the weightinh factor described above.

param->Matrix is an *AVW_Matrix* which specifies the rotation and translation transformation applied to the *AVW_Volume* during the rendering process. Scale could also be specified as part of the matrix, but it's recommended that the *ScaleX, ScaleY,* and *ScaleZ* parameters be used for Scale. [Default = identity matrix]

param->LightMatrix specifies the vector of the light source used in reflective renderings. [Default = identity matrix]

param->RenderMask is an *AVW_Image* which specifies a specific area to be re-rendered. If *NULL* is specified the entire rendering space is rendered each time *AVW_RenderVolume* is called. Only *AVW_Images* with a data type of *AVW_UNSIGNED_CHAR* are supported. This *AVW_Image* should always have dimensions equal to *param->RenderWidth* and *param->RenderedHeight. [Default = NULL]*

param->MaskValue specifies the pixel value within the *param->RenderMask* which indicates rendering should occur for this output pixel. [Default = 255]

param->DeleteDepth specifies the number of voxels along the ray path which are deleted (changed to *param->DeleteValue).* When *param->Type* is set to *AVW_DeleteVoxels* the following settings are valid:

> *AVW_DELETE_ALL_THE_WAY* – All voxels along the ray path are changed to the value to specified by *param->DeleteValue. [default]*
>
> *AVW_DELETE_SINGLE_LAYER* – Once a value within the theshold limits is detected, all voxels with values within the threshold range are reddifined as the object specified by *param->DeleteValue* until a voxel outside the threshold range is detected, at which time object definition stops.
>
> *AVW_DELETE_SINGLE_VOXEL* – The first voxel along the ray path which is within the threshold range is deleted.
>
> *AVW_DEFINE_ALL_THE_WAY* – All object map locations along the ray path are changed to the value to specified by *param->DeleteValue.*
>
> *AVW_DEFINE_SINGLE_LAYER* – Once a value within the theshold range is detected, all voxels with values within the threshold limits are redfined until a voxel outside the threshold range is detected, at which time deleting stops.
>
> *AVW_DEFINE_SINGLE_VOXEL* – The first object map location along the ray path which is within the threshold limits is redefined.
>
> *NOTE:* A 3x3x3 region surrounding a voxel is deleted / defined instead of just a single voxel, this compensates for spaces which may occur between the ray paths.

param->DeleteValue specifies the value each voxel or object map location is changed to, when the *AVW_DELETE_VOXELS* rendering type is specified. [Default = 0]

param->ScaleX, ScaleY and *param->ScaleZ* specify the scaling factors. *param->ScaleX* specifies the width of each rendered voxel. *param->ScaleY* the height and *param->ScaleZ* the depth. A value of .5 indicates the voxel width ,height, or depth is half it's normal size. A value of 2.0 indicates the voxel is twice it's normal width, height, or depth. When specifying scale factors larger than 1.0, changes to the *param->RenderWidth*, *param->RenderHeight* and *param->RenderDepth* are required to prevent segmentation violations. [Default = 1.0]

param->PerspectiveType specifies the type of rendering to be performed. *AVW_PERSPECTIVE_INT* renders the image using the voxels without interpolation, possibly resulting in "blocky" renderings. If *param->PerspectiveType* is set to *AVW_PERSPECTIVE_FLOAT*, the rendered image is generated using an on the fly intepolation rendering algorithm. If *param->PerspectiveType* is set to *AVW_PERSPECTIVE_OFF,* parallel rendering is performed. For perspective rendering, the render matrix is interperted as a viewing direction for the camera model, however the parallel rendering conventions are followed, i.e. the matrix is left handed, and the identity matrix provides a view along the positive Z axis with X increasing to the right and Y increasing in the vertical direction. Perspective rendering does not support scaling at this time. For anisotropic data, rescaling during loading is the best option.

param->EyePosition specifies the location of the camera model used to generate the rendering. This parameter is only used when *param->PerspectiveType* is set to *AVW_PERSPECTIVE_FLOAT* or *AVW_PERSPECTIVE_INT*. The *X*, *Y*, and *Z* coordinates of the *AVW_FPoint3* structure refer to the position of the camera relative to the center of the volume, thus a position of (0,0,0) in a particular volume is located at voxel (Width / 2, Height / 2, Depth / 2).

param->XFieldOfViewAngle and *param->YFieldOfViewAngle* specify the field of view (FOV) angle of the camera model used to generate the image. Increasing the FOV results in a zoom out effect, if the position remains the same. Decreasing the FOV results in a zoom in effect.

param->SpecularFactor specifies the ratio of gradient shading to specular shading. If *param->SpecularFactor* is set to 0, the rendering will be shaded entirely using gradient shading with no performance penalty. If *param->SpecularFactor* is set to .1, the rendering is shaded entirely using the specular shading model.

param->SpecularExponent specifies the degree of fall-off for specular shading. High values (about 10) result in very small specular highlights, while small values (about 1 or 2) result in diffuse highlights.

param->CompositeInfo specifies a pointer to a structure which contains compositing information. This must contain a valid pointer when the *param->Type is set to AVW_VOLUME_COMPOSITING.* See *AVW_CompositeInfo* for more information.

BackgroundColor and *BackgroundValue* are used to specify the background color or value. The rendering type, input volumes datatype, and whether an object map is loaded determines which value is used. If the output is a grayscale image, *BackgroundValue* is used. If the output is a color image, then *BackgroundColor* will be used. *Background Color* is a packed RGB value which can be produced with the *AVW_RGB* macro.

RenderMode is normally set to *AVW_RENDER_NORMAL,* but when set to *AVW_PREPARE_FOR_MOVE,* the *AVW_RenderVolume()* function will process the object specified in the *InteractiveObject* seperately and return the "visible surface" in the *AVW_RenderedImage* member called *InteractiveSurface.* This results in the input to *AVW_RenderVisibleSurface()* which produces output which can be combined using *AVW_MergeRendered()* with the rendering returned at the time the *InteractiveSurface* was produced. This entire process allows an interface to be build to interactively move and rotate objects. *RenderMode* can be set to

AVW_RERENDER_MOVED, at the completion of the object transformation to rerender any missing data.

param->Internal contains parameter which are used internally within the rendering functions. *CHANGING OF INTERNAL PARAMETERS IS DISCOURAGED!*

RETURN VALUES

If successful *AVW_InitializeRenderParameters()* returns a pointer to a *AVW_RenderParameters* structure. This structure contains the parameters which may be modified to produce the desired rendered image. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_InitializeRenderParameters() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

CFLSZ
The AVW_Volume and AVW_ObjectMap have conflicting dimensions.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_DrawRenderedPoint(), AVW_DrawRenderedLine(), AVW_FindRenderedPoint(), AVW_LoadObjectMap(), AVW_MirrorRendered(), AVW_ProcessZGradients(), AVW_SaveObjectMap() AVW_RenderVolume(), AVW_Image AVW_ObjectMap AVW_RenderParameters,

NAME

AVW_InitializeTileParameters – initializes surface tiling parameters

SYNOPSIS

```
#include "AVW_Model.h"

AVW_TileParameters *AVW_InitializeTileParameters(volume,
        object_map, last_tile_param)
AVW_Volume *volume;
AVW_ObjectMap *object_map;
AVW_TileParameters *last_tile_param;
```

DESCRIPTION

AVW_InitializeTileParameters() creates and returns a structure required to tile the specified *AVW_Volume.* This structure is passed to *AVW_TileVolume()* to obtain a tiled surface from the volume. If an *AVW_ObjectMap* is specified it will be used in the tiling process instead of the volume. The *last_tile_param* structure may be specified as a starting point and only required changes will be made for the returned structure.

Parameters of *AVW_InitializeTileParameters():*

Volume specifies the *AVW_Volume* to be rendered.

Object_map specifies a map of objects defined within the *AVW_Volume.* This map must be exactly the same dimensions as the *AVW_Volume* to be rendered. This map is commonly the result of a call to *AVW_LoadObjectMap()* or *AVW_CreateObjectMap().* An *AVW_ObjectMap* is not required, *NULL* may be specified to indicate that no object map is to be used.

Last_tile_param specifies a starting set of paramaters. This structure is most likely the results of a previous call to *AVW_InitializeTileParameters().* Only parameters which require modification as the results of an *AVW_Volume* or *AVW_ObjectMap* change will be modified in the structure returned. If *last_tile_parm is equal to NULL a AVW_TileParameters* structure will be created.

The following elements of the *AVW_TileParameters* structure are set by *AVW_InitializeTileParameters()* when the structure is created.

param->Type specifies the type of tiling preformed. Supported values include:

AVW_TILE_KOHONEN – The tiled surface will be created using a Kohonen Net to map a mesh to the object's surface. *AVW_TILE_GROW()* – The tiled surface will be created using a polygon growing algorithm. *AVW_MARCHING_CUBES()* – The tiled surface will be created using the marching cubes algorithm.

param->Checkpoint Ignored at present.

param->Mask specify the mask value of the object within the *AVW_Volume* to tile.

param->CurveOpRadius specifies the spacing between elements in the curvature detection operator [default is 1].

param->CloseSrfcFlag toggles the production of a closed (if set) or open (if unset) ended surface [default is unset].

param->KohonenMajorAxis specifies which axis is to be used as the major axis when tiling. This parameter may be one of:
AVW_XAXIS, AVW_YAXIS, or *AVW_ZAXIS [default]*

param->KohonenShapeOrient specifies the orientation of the initial network. This parameter may be one of:
AVW_TRANSVERSE [default], AVW_CORONAL, or *AVW_SAGITTAL*

param->KohonenShapeOffset specifies the initial distance between the object's surface and the network. A value less than 1 will place the initial network inside the object, while a value greater than 1 will place the network outside the object. The actual distance is determined by multipling the radii of the 2-D bounding oval by *param->KohonenShapeOffset*

param->KohonenFlag is set by oring together:
AVW_TRAIN_IN_MAJOR_AXIS and *AVW_TRAIN_WITH_WEIGHT*
These parameters will cause the network to train along the major axis in addition to the minor axis and/or to use a weighted training algorithm.

param->PolygonBudget specifies the maximum number of polygons that may be used to form the model.

param->KohonenRepetitions specifies the number of times the data is presented to the network during adaptation.

param->KohonenNeighborhood specifies how a "neighborhood" of cells within the Kohonen Network will move during adaptation. This parameter is one of:
AVW_BUBBLE_NEIGHBORHOOD, AVW_GAUSS_NEIGHBORHOOD, or
AVW_TRIANGLE_NEIGHBORHOOD

param->KohonenTopology specifies the topology of the network. Currently the only accepted topology is *AVW_RECTANGULAR_TOPOLOGY*

param->KohonenNeighborRadius specifies initial radius of a network neighborhood. If this parameter is less than 1, the initial radius will be set to be one third of the distance around a 2-D bounding oval orthogonal to the major axis

param->KohonenAlpha specifies the initial learning rate during adaptation.

param->AddNodeFreq specifies the number of adaptation steps envoked by the growing net algorithm before a new node is added.

param->MaximumAge specifies the maximal age number allowed for the growing net edges (connections). Edges with age greater than *param->MaximumAge* are removing from the growing net.

param->Eb, param->En specifies the adaptation factors for the best matching unit (bmu) node and its direct neighbors, respectively.

param->GrowingAlpha specifies the factor for decreasing the error counters of the 1st and 2nd bmus after the insertion of a new node to the growing net, respectively.

param->GrowingD specifies the factor for decreasing the error counters of all nodes after the insertion of a new node to the growing net.

RETURN VALUES

If successful *AVW_InitializeTileParameters()* returns a pointer to a *AVW_TileParameters* structure. This structure contains the parameters which may be modified to produce the desired rendered image. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_InitializeTileParameters() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_LoadObjectMap(), AVW_SaveObjectMap() AVW_TileVolume(), AVW_Image AVW_ObjectMap AVW_TileParameters

NAME

AVW_InsertFPoint2 – inserts a point in a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_InsertFPoint2(trace, index, point)
AVW_FPointList2 *trace;
int index;
AVW_FPoint2 *point;
```

DESCRIPTION

AVW_InsertFPoint2() inserts an *AVW_FPoint2, point,* to an *AVW_FPointList2, trace* before the point specified by *index*.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_InsertFPoint2()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_InsertFPoint2() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddFPoint2(), AVW_InsertFPoint3(), AVW_InsertIPoint2(), AVW_InsertIPoint3(), AVW_InsertPoint2(), AVW_InsertPoint3(), AVW_InsertPointValue(), AVW_CopyPointList2() AVW_CreateFPointList2(), AVW_DestroyFPointList2(), AVW_RemoveFPoint2(), AVW_FPointList2, AVW_FPoint2

NAME

AVW_InsertFPoint3 – adds a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_InsertFPoint3(trace, index, point)
AVW_FPointList3 *trace;
int index;
AVW_FPoint3 *point;
```

DESCRFPTION

AVW_InsertFPoint3() adds an *AVW_FPoint3*, *point* to an *AVW_FPointList3, trace* before the point specified by *index*.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_InsertFPoint2()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_InsertFPoint3() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddFPoint3(), AVW_InsertFPoint2(), AVW_InsertIPoint2(), AVW_InsertIPoint3(), AVW_InsertPoint2(), AVW_InsertPoint3(), AVW_InsertPointValue(), AVW_CopyFPointList3(), AVW_CreateFPointList3(), AVW_DestroyFPointList3(), AVW_GetFPoint3(), AVW_RemoveIpoint3(), AVW_FPointList3, AVW_FPoint3

NAME

AVW_InsertIPoint2 – inserts a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_InsertIPoint2(trace, index, point)
AVW_IPointList2 *trace;
int index;
AVW_IPoint2 *point;
```

DESCRIPTION

AVW_InsertIPoint2() inserts an *AVW_IPoint2, point* to an *AVW_IPointList2, trace* before the point specified by *index*.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_InsertIPoint2()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_InsertIPoint2() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddIPoint2(), AVW_InsertIPoint3(), AVW_InsertFPoint2(), AVW_InsertFPoint3(), AVW_InsertPoint2(), AVW_InsertPoint3(), AVW_InsertPointValue(), AVW_CopyIPointList2(), AVW_CreateIPointList2(), AVW_DestroyIPointList2(), AVW_GetIPoint2(), AVW_RemoveIpoint2(), AVW_IPointList2, AVW_IPoint2

NAME

AVW_InsertIPoint3 – inserts a point to a list structure

SYNOPSIS

```
#include    "AVW.h"

int AVW_InsertIPoint3(trace, index, point)
AVW_IPointList3 *trace;
int index;
AVW_IPoint3 *point;
```

DESCRIPTION

AVW_InsertIPoint3() inserts an *AVW_IPoint3, point,* to an *AVW_IPointList3, trace* before the point specified by *index.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_InsertIPoint3()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_InsertIPoint3() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddIPoint3(), AVW_InsertIPoint2(), AVW_InsertFPoint2(), AVW_InsertFPoint3(), AVW_InsertPoint2(), AVW_InsertPoint3(), AVW_InsertPointValue(), AVW_CopyIPointList3(), AVW_CreateIPointList3(), AVW_DestroyIPointList3(), AVW_GetIPoint3(), AVW_RemoveIPoint3(), AVW_IPointList3, AVW_IPoint3

NAME AVW_InsertPoint2 – inserts a point in a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_InsertPoint2(trace, index, point)
AVW_PointList2 *trace;
int index;
AVW_Point2 *point;
```

DESCRIPTION *AVW_InsertPoint2()* adds an *AVW_Point2, point,* to an *AVW_PointList2, trace* before the point specified by *index.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES If successful *AVW_InsertPoint2()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS *AVW_InsertPoint2()* will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO *AVW_AddPointList2(), AVW_InsertFPoint2(), AVW_InsertFPoint3(), AVW_InsertIPoint2(), AVW_InsertIPoint3(), AVW_InsertPoint3(), AVW_InsertPointValue(), AVW_AddPointList2(), AVW_CreatePointList2(), AVW_CopyPointList2(), AVW_DestroyPointList2(), AVW_EditPointList2(), AVW_FillPointList2(), AVW_GetPoint2(), AVW_RemovePoint2(), AVW_PointList2, AVW_Point2*

NAME

AVW_InsertPoint3 – inserts a point to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_InsertPoint3(trace, index, point)
AVW_PointList3 *trace;
int index;
AVW_Point3 *point;
```

DESCRIPTION

AVW_InsertPoint3() inserts an *AVW_Point3, point,* to an *AVW_PointList3, trace* before the point specified by *index.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_InsertPoint3()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_InsertPoint3() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddPoint3(), AVW_InsertFPoint2(), AVW_InsertFPoint3(), AVW_InsertIPoint2(), AVW_InsertIPoint3(), AVW_InsertPoint2(), AVW_InsertPointValue(), AVW_CopyPointList3(), AVW_CreatePointList3(), AVW_DestroyPointList3(), AVW_FillPointList3(), AVW_GetPoint3(), AVW_RemovePoint3(), AVW_PointList3, AVW_Point3

NAME

AVW_InsertPointValue – inserts a point and a value to a list structure

SYNOPSIS

```
#include "AVW.h"

int AVW_InsertPointValue(trace, index, point, value)
AVW_PointValueList *trace;
index
AVW_Point2 *point;
double value;
```

DESCRIPTION

AVW_InsertPointValue() inserts an *AVW_Point2*, *point*, and a *value* to an *AVW_PointValueList*, *trace* before the point specified by *index*.

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

RETURN VALUES

If successful *AVW_InsertPointValue()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_InsertPointValue() will fail if:

BADMAL
Malloc Failed. Unable to increase structure size.

SEE ALSO

AVW_AddPointValue(), AVW_InsertFPoint2(), AVW_InsertFPoint3(), AVW_InsertIPoint2(), AVW_InsertIPoint3(), AVW_InsertPoint2(), AVW_InsertPoint3(), AVW_CopyPointValueList(), AVW_CreatePointValueList(), AVW_DestroyPointValueList(), AVW_GetPointValue(), AVW_RemovePointValue(), AVW_PointValueList, AVW_Point2

NAME

AVW_IntensityClipImage – sets values outside range

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_IntensityClipImage(in_image, clip_max, clip_min,
        clip_maxval, clip_minval, out_image)
AVW_Image *in_image;
double clip_max, clip_min, clip_maxval, clip_minval;
AVW_Image *out_image;
```

DESCRIPTION

AVW_IntensityClipImage() sets all pixels greater than *clip_max* to *clip_maxval.* All pixels less than *clip_min* are set to *clip_minval.* All pixels within the *clip_max* and *clip_min* range are copied unchanged.

The *DataType* of the returned image will always be the same as the input *DataType.* The *clip_maxval* and *clip_minval* need not be within the *clipmax* and *clipmin* range, but can be. They can also be set to the same value, thus setting all pixels outside the range to a specific value.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_IntensityClipImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_IntensityClipImage() will fail if the following is true:

BADMAL
Couldn't allocate memory.

SEE ALSO

AVW_ConvertImage(), AVW_IntensityClipVolume(), AVW_IntensityScaleImage(), AVW_TableImage(), AVW_ThresholdImage(), AVW_Image

NAME

AVW_IntensityClipVolume – sets values outside range

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_IntensityClipVolume(in_volume, clip_max, clip_min,
        clip_maxval, clip_minval, out_volume)
AVW_Volume *in_volume;
double clip_max, clip_min, clip_maxval, clip_minval;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_IntensityClipVolume() sets all voxels greater than *clip_max* to *clip_maxval.* All voxels less than *clip_min* are set to *clip_minval.* All voxels within the *clip_max* and *clip_min* range are copied unchanged.

The *DataType* of the returned volume will always be the same as the input *DataType.* The *clip_maxval* and *clip_minval* need not be within the *clipmax* and *clipmin* range, but can be. They can also be set to the same value, thus setting all voxels outside the range to a specific value.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_IntensityClipVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_IntensityClipVolume() will fail if the following is true:

BADMAL
Couldn't allocate memory.

SEE ALSO

AVW_ConvertVolume(), AVW_IntensityClipImage(), AVW_IntensityScaleVolume(), AVW_TableVolume(), AVW_ThresholdVolume(), AVW_Volume

NAME

AVW_IntensityScaleImage – scales image intensity values

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_IntensityScaleImage(in_image, in_max, in_min,
        out_max, out_min, out_dt, out_image)
AVW_Image *in_image;
double in_max, in_min, out_max, out_min;
int out_dt;
AVW_Image *out_image;
```

DESCRIPTION

AVW_IntensityScaleImage() takes an *AVW_Image, in_image,* and maximum and minimum intensity values, *in_max* and *in_min,* and linearly scale the pixel values of the image to create the resulting image with the maximum and minmum values of *out_max* and *out_min* and of the data type *out_dt.* The scaled intensity values are calculated by the following equation:

out_val = (in_val - in_min) *(out_max - out_min) / (in_max - in_min)
    + out_min

Where in_val is the original intensity value of the *in_image* and out_val is the returned intensity in *out_image.*

out_dt may be one of the AVW image data types.

AVW_UNSIGNED_CHAR AVW_SIGNED_CHAR AVW_UNSIGNED_SHORT AVW_SIGNED_SHORT AVW_UNSIGNED_INT AVW_SIGNED_INT AVW_FLOAT

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

AVW_IntensityScaleImage() can also be used to perform image conversion.

ERRORS

AVW_ItensityScaleImage() will fail if:

NOTSUP
    Input or output image data type not supported.

RETURN VALUES

If successful *AVW_IntensityScaleImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_IntensityScaleVolume(), AVW_ObjectScaleImage(), AVW_ConstantOpImage(), AVW_ConvertImage(), AVW_ImageOpConstant(), AVW_DestroyImage(), AVW_Image

NAME

AVW_IntensityScaleVolume – scales volume intensity values

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_IntensityScaleVolume(in_volume, in_max,
        in_min, out_max, out_min, out_dt, out_volume)
AVW_Volume *in_volume;
double in_max, in_min, out_max, out_min;
int out_dt;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_IntensityScaleVolume() takes an *AVW_Volume, in_volume,* and maximum and minimum intensity values, *in_max* and *in_min*, and linearly scale the voxel values of the volume to create the resulting volume with the maximum and minmum values of *out_max* and *out_min* and of the datatype *out_dt*. The scaled intensity values are calculated by the following equation:

out_val = (in_val - in_min) *(out_max - out_min) / (in_max - in_min)
    + out_min

Where out_val (returned) and in_val (original) are the intensity values of the returned volume and *in_volume.*

out_dt may be one of the AVW image data types.

AVW_UNSIGNED_CHAR AVW_SIGNED_CHAR AVW_UNSIGNED_SHORT AVW_SIGNED_SHORT AVW_UNSIGNED_INT AVW_SIGNED_INT AVW_FLOAT

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

AVW_IntensityScaleVolume() can also be used to do volume conversion.

ERRORS

AVW_IntensityScaleVolume() will fail if:

NOTSUP
: Input or output volume data type not supported.

RETURN VALUES

If successful *AVW_IntensityScaleVolume()* returns an AVW_Volume. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_IntensityScaleImage(), AVW_ImageOpConstant(), AVW_DestroyVolume(), AVW_Volume

NAME

AVW_InterpolatedPixel – returns pixel value at a floating point location

SYNOPSIS

```
#include "AVW.h"

double AVW_InterpolatedPixel(image, point)
AVW_Image *image;
AVW_FPoint2 *point;
```

DESCRIPTION

Given a floating point location *point* within *image, AVW_InterpolatedPixel()* returns the calculated pixel value at the floating point location.

Bi-linear calculation is used to estimate the pixel value at the floating point location.

Points outside the image will return a value of 0.0.

SEE ALSO

AVW_GetPixel(), AVW_GetErrorNumner(), AVW_InterpolatedVoxel(), AVW_NearestNeighborPixel(), AVW_CubicSplineInterpolatedPixel(), AVW_SincInterpolatedPixel(), AVW_Image, AVW_FPoint2

NAME

AVW_InterpolatedVoxel – returns voxel value at a floating point location

SYNOPSIS

```
#include "AVW.h"

double AVW_InterpolatedVoxel(volume, point)
AVW_Volume *volume;
AVW_FPoint3 *point;
```

DESCRIPTION

Given a floating point location *point* within *volume, AVW_InterpolatedVoxel()* returns the calculated voxel value at the floating point location.

Tri-linear calculation is used to estimate the voxel value at the floating point location.

Points outside the volume will return a value of 0.0.

SEE ALSO

AVW_GetVoxel(), AVW_GetErrorNumner(), AVW_InterpolatedPixel(), AVW_NearestNeighborVoxel(), AVW_CubicSplineInterpolatedVoxel(), AVW_SincInterpolatedVoxel(), AVW_FPoint3, AVW_Volume

NAME

AVW_IntersectingSections – renders intersecting sections

SYNOPSIS

```
#include "AVW_Render.h"

AVW_RenderedImage *AVW_IntersectingSections(volume, x, y, z, matrix, interpolate_flag,
                                            shading_fraction, last_rendered)
AVW_Volume *volume;
int x, y, z;
AVW_Matrix *matrix;
int interpolate_flag;
double shading_fraction;
AVW_RenderedImage *last_rendered;
```

DESCRIPTION

AVW_IntersectingSections() returns an *AVW_RenderedImage* showing the intersection of a transverse, coronal and sagittal section.

Volume specifies the *AVW_Volume* the sections are extracted from.

X, y, and *z* specify the 3 space coordiate where the sections intersect. *X* specifies the which sagittal slice, *Y* specifies the coronal slice, and *Z* specifies the transverse slice.

Matrix is used to specify any rotation or scale factors.

The *interpolate_flag* specifies if tri-linear interpolations should be used when generating the image. Setting the *interpolate_flag* to *AVW_FALSE* causes *Nearest Neighbor* to be used, which is much faster, but lacks some of the quality.

Last_rendered is provided as a method of reusing an existing *AVW_RenderedImage.* Reuse is possible only if the size and data type of the provided *last_rendered* meet the requirements of the function. In this case the pointer to *last_rendered* is returned by the function. If not reusable *last_rendered* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_IntersectingSections()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_IntersectingSections() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_CubeSections(), AVW_RenderOblique(), AVW_DestroyRenderedImage(), AVW_RenderedImage

NAME

AVW_InvertImage – inverts pixel intensities

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_InvertImage(in_image, maximum, minimum, out_image)
AVW_Image *in_image;
double maximum, minimum;
AVW_Image *out_image;
```

DESCRIPTION

AVW_InvertImage() inverts the pixel intensities in *in_image* according to the following formula:

(maximum-minimum) - (pixel_value - minimum) + minimum

where *pixel_value* is the value of each pixel in the image and *maximum* and *minimum* are user supplied values.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_InvertImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. ERRORS *AVW_InvertImage()* will fail if the following is true:

NOTSUP
: Data type not supported. AVW_COMPLEX and AVW_COLOR images are not supported.

SEE ALSO

AVW_InvertVolume(), AVW_Image

NAME

AVW_InvertMatrix – inverts a transformation matrix

SYNOPSIS

#include "AVW.h"

AVW_Matrix *AVW_InvertMatrix(in_matrix, out_matrix)
AVW_Matrix *in_matrix, *out_matrix;

DESCRIPTION

AVW_InvertMatrix() returns the inverse ofthe *AVW_Matrix, in_matrix.*

Out_matrix is provided as a method of reusing an existing *AVW_Matrix. (See Memory Usage* in the *AVW Programmer's Guide.)*

SEE ALSO

AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis(), AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_SetIdentityMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_Matrix

NAME

AVW_InvertVolume – inverts voxel intensities

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_InvertVolume(in_volume, maximum, minimum, out_volume)
AVW_Volume *in_volume;
double maximum, minimum;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_InvertVolume() inverts the voxel intensities in *in_volume* according to the following formula:

(maximum-minimum) - (voxel_value - minimum) + minimum

where *voxel_value* is the value of each voxel in the volume and *maximum* and *minimum* are user supplied values.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_InvertVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. ERRORS *AVW_InvertVolume()* will fail if the following is true:

NOTSUP
: Data type not supported. AVW_COMPLEX and AVW_COLOR volumes are not supported.

SEE ALSO

AVW_InvertImage(), AVW_Volume

NAME

AVW_IsASubsetImage – determines if one image is a subset of another

SYNOPSIS

#include "AVW.h"

int AVW_IsASubsetImage(image1, image2)
AVW_Image *image1, image2;

DESCRIPTION

AVW_IsASubsetImage() is used to determine if the non-zero pixels in *image1* are a subset of the non-zero pixels in *image2. Image1* and *image2* must be of the same data type.

RETURN VALUES

AVW_TRUE is returned if *image1* is a subset of *image2. AVW_FALSE* is returned if it is not.

ERRORS

AVW_IsASubsetImage() will fail if:

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_Image

NAME

AVW_IsGrayColormap – determines if a colormap is gray scale

SYNOPSIS

```
#include "AVW.h"

int AVW_IsGrayColormap(map)
AVW_Colormap *map;
```

DESCRIPTION

AVW_IsGrayColormap() is used to determine if an *AVW_Colormap* is a gray scale map.

RETURN VALUES

AVW_TRUE is returned if all corresponding Red, Green, and Blue values are equal and if successive cells have greater values. Otherwise *AVW_FALSE* is returned.

SEE ALSO

AVW_Colormap

NAME

AVW_IsImageZero – determines if an entire image is zero

SYNOPSIS

```
#include "AVW.h"

int AVW_IsImageZero(in_image)
AVW_Image *in_image;
```

DESCRIPTION

AVW_IsImageZero() is used to determine if every pixel in an *AVW_Image* is equal to zero.

RETURN VALUES

Upon encountering the first nonzero pixel in *in_image AVW_FALSE* is returned. If the entire image is set to zero, *AVW_TRUE* is returned.

SEE ALSO

AVW_IsVolumeZero(), AVW_Image

NAME

AVW_IsVolumeZero – determines if an entire volume is zero

SYNOPSIS

```
#include "AVW.h"

int AVW_IsVolumeZero(in_volume)
AVW_Image *in_volume;
```

DESCRIPTION

AVW_IsVolumeZero() is used to determine if every voxel in an *AVW_Volume* is equal to zero.

RETURN VALUES

Upon encountering the first nonzero voxel in *in_volume AVW_FALSE* is returned. If the entire volume is set to zero, *AVW_TRUE* is returned.

SEE ALSO

AVW_IsImageZero(), AVW_Volume

NAME

AVW_IterDeconvImage – performs constrained iterative image deconvolution

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_IterDeconvImage(obs_image, transfer_func,
        update_rule, no_iter, guess_image)
AVW_Image *obs_image, *transfer_func;
int update_rule, no_iter;
AVW_Image *guess_image;
```

DESCRIPTION

AVW_IterDeconvImage() performs constrained iterative deconvolution on an image (see reference below). The actual observed image is *obs_image,* and its dimensions are expected to be powers of 2. The modulation transfer function to use in the deconvolution is given by *transfer_func,* and it must be of data type *AVW_FLOAT* or *AVW_COMPLEX* and of dimensions xnum / 2 + 1, ynum, where xnum and ynum are the dimensions of the *obs_image.* The initial (or current) guess is supplied in *guess_image,* and it must be of data type *AVW_FLOAT* and the same dimensions as the *obs_image.* If it is not, or it is *NULL,* the function allocates new memory for the returned *guess_image* and takes the observed image to be the initial guess.

This function can be called repeatedly with the output from the last call being the *guess_image* supplied to the next call in order to perform some iterations, examine the result, and then perform more iterations. The *update_rule* parameter has the acceptable values *AVW_UPDATE_VC* and *AVW_UPDATE_GOLD,* and controls which updating scheme to use (see reference). The *no_iter* parameter controls how many iterations to perform.

More detailed information about constrained iterative deconvolution may be found in the reference:

Agard, David A., *Fluorescence Microscopy in Three Dimensions* 1989, in Methods in Cell Biology, Vol. 30, Chap. 13, Academic Press.

RETURN VALUES

If successful *AVW_IterDeconvImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_IterDeconvImage() will fail if the following is true:

BADMAL
: Malloc Failed. Unable to allocate memory for return image.

ILLPAR
: Illegal Parameter. An invalid value was given for an input parameter.

ILLIMG
: Illegal Image. Input image was not a power of 2 in each dimension.

BDSPCT
: Bad Input Spectrum. An invalid spectrum was entered for the transfer function.

SEE ALSO

AVW_IterDeconvVolume() AVW_CreateStoksethMTF(), AVW_DeconvDivideImage(), AVW_DeconvWienerImage(), AVW_NearestNeighborDeconv(), AVW_Image

NAME

AVW_IterDeconvVolume – performs constrained iterative volume deconvolution

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_IterDeconvVolume(obs_volume, transfer_func,
        update_rule, no_iter, guess_volume)
AVW_Volume *obs_volume, *transfer_func;
int update_rule, no_iter;
AVW_Volume *guess_volume;
```

DESCRIPTION

AVW_IterDeconvVolume() performs constrained iterative deconvolution on a volume (see reference below). The actual observed volume is *obs_volume*, and its dimensions are expected to be powers of 2. The modulation transfer function to use in the deconvolution is given by *transfer_func*, and it must be of data type *AVW_FLOAT* or *AVW_COMPLEX* and of dimensions xnum / 2 + 1, ynum, and znum where xnum, ynum, and znum are the dimensions of the *obs_volume*. The initial (or current) guess is supplied in *guess_volume*, and it must be of data type *AVW_FLOAT* and the same dimensions as the *obs_volume*. If it is not, or it is *NULL*, the function allocates new memory for the returned *guess_volume* and takes the observed volume to be the initial guess.

AVW_IterDeconvVolume() can be called repeatedly with the output from the last call being the *guess_volume* supplied to the next call in order to perform some iterations, examine the result, and then perform more iterations. The *update_rule* parameter has the acceptable values *AVW_UPDATE_VC* and *AVW_UPDATE_GOLD*, and controls which updating scheme to use (see reference). The *no_iter* parameter controls how many iterations to perform.

More detailed information about constrained iterative deconvolution may be found in the reference:

Agard, David A., *Fluorescence Microscopy in Three Dimensions* 1989, in Methods in Cell Biology, Vol. 30, Chap. 13, Academic Press.

RETURN VALUES

If successful *AVW_IterDeconvVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

Both *AVW_IterDeconvVolume()* will fail if the following is true:

BADMAL
: Malloc Failed. Unable to allocate memory for return volume or image.

ILLPAR
: Illegal Parameter. An invalid value was given for an input parameter.

ILLVOL
: Illegal Volume. Input image was not a power of 2 in each dimension.

BDSPCT
: Bad Input Spectrum. An invalid spectrum was entered for the transfer function.

SEE ALSO

AVW_IterDeconvImage(), AVW_DeconvDivideVolume(), AVW_DeconvWienerVolume(), AVW_Volume

NAME

AVW_LabelImageFromEdges – finds connected regions in an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_LabelImageFromEdges(in_image, connectivity, out_image)
AVW_Image *in_image;
int connectivity;
AVW_Image *out_image;
```

DESCRIPTION

AVW_LabelImageFromEdges() finds all of the unique connected regions in *in_image* seperated by edges. All nonzero pixels in *in_image* are interpreted as edge points. See *AVW_FindImageEdges().* Each unique connected region is given a different value in *out_image.*

Connectivity may be either *AVW_4_CONNECTED* or *AVW_8_CONNECTED* and specifies the neighbors to be used to determine the connected components.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_LabelImageFromEdges()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_LabelVolumeFromEdges(), AVW_FindImageComponents(), AVW_FindImageEdges(), AVW_Image

NAME

AVW_LabelVolumeFromEdges – finds connected regions in a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_LabelVolumeFromEdges(in_volume, connectivity, out_volume)
AVW_Volume *in_volume;
int connectivity;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_LabelVolumeFromEdges() finds all of the unique connected regions in *in_volume* seperated by edges. All nonzero voxels in *in_volume* are interpreted as edge points. See *AVW_FindVolumeEdges()*. Each unique connected region is given a different value in *out_volume.*

Connectivity may be either *AVW_6_CONNECTED* or *AVW_26_CONNECTED* and specifies the neighbors to be used to determine the connected components.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_LabelVolumeFromEdges()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_LabelImageFromEdges(), AVW_FindVolumeComponents(), AVW_FindVolumeEdges(), AVW_Volume

NAME

AVW_ListFormats – returns a list of supported image file formats

SYNOPSIS

```
#include "AVW_ImageFile.h"

AVW_List *AVW_ListFormats (properties)
int properties;
```

DESCRIPTION

AVW_ListFormats() returns an *AVW_List* of image file formats that are currently extended to support the passed property. *Properties* is a value made by combining the following values from *AVW_ImageFile.h* with the logical OR "|" operator:

AVW_SUPPORT_UNSIGNED_CHAR AVW_SUPPORT_SIGNED_CHAR AVW_SUPPORT_UNSIGNED_SHORT AVW_SUPPORT_SIGNED_SHORT AVW_SUPPORT_UNSIGNED_INT AVW_SUPPORT_SIGNED_INT AVW_SUPPORT_FLOAT AVW_SUPPORT_COMPLEX AVW_SUPPORT_COLOR AVW_SUPPORT_2D AVW_SUPPORT_3D AVW_SUPPORT_4D AVW_SUPPORT_READ AVW_SUPPORT_WRITE

For example:

```
list = AVW_ListFormats(AVW_SUPPORT_READ);
```

gets a list of all formats supported for reading;

and

```
list = AVW_ListFormats(AVW_SUPPORT_WRITE);
```

gets a list of all formats supported for writing.

RETURN VALUES

Upon success *AVW_ListFormats()* returns an *AVW_List* in which the entries strings used to identify the image file format. For use with *AVW_CreateImageFile(). AVW_SUCCESS.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of failure.

ERRORS

AVW_ListFormats() will fail if one or more of the following are true:

BDMAL
    Malloc Failed

SEE ALSO

AVW_CreateImageFile(), AVW_DestroyList(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_DisableImageFileFormat(), AVW_EnableImageFileFormat(), AVW_ExtendIO

NAME AVW_ListInfo –lists entries in an information string

SYNOPSIS

```
#include "AVW.h"

AVW_List *AVW_ListInfo(info_string)
char *info_string;
```

DESCRIPTION *AVW_ListInfo()* creates a list of strings which correspond to each information element in an information string. Each information element is a string of the form Label=Value. Info strings are used in *AVW* structures to store additional information about the data and to allow the user to extend the structures to carry application dependant data.

RETURN VALUES If successful *AVW_ListInfo()* returns an *AVW_List*. On failure, *NULL* is returned.

SEE ALSO *AVW_DestroyList(), AVW_GetNumericInfo(), AVW_MergeInfo(), AVW_PutHistoryInfo(), AVW_PutNumericInfo(), AVW_PutStringInfo(), AVW_RemoveInfo(), AVW_Image, AVW_ImageFile, AVW_List, AVW_Volume*

NAME

AVW_LoadColormap − loads a color map

SYNOPSIS

```
#include "AVW.h"

AVW_Colormap *AVW_LoadColormap(file)
char *file;
```

DESCRIPTION

AVW_LoadColormap() reads an *AVW_Colormap* from the disk file called *file. AVW* colormap files usually end in *.lkup.* Colormap values are stored as ASCII strings representing values from 0 to 255. The red value for the first cell, if followed by the green value for the first cell, followed by the blue value for the first cell. This order is followed for each colorcell defined by the file.

RETURN VALUES

If successful *AVW_LoadColormap()* returns a pointer to an *AVW_Colormap* structure. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_LoadColormap() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

BADOPEN
Could open file for reading or writing.

BADREAD
Error occurred while reading file.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_CreateColormap(), AVW_DestroyColormap(), AVW_SaveColormap(), AVW_Colormap

NAME

AVW_LoadCompositeInfo – loads compositing information

SYNOPSIS

```
#include "AVW_CompositeInfo.h"

AVW_CompositeInfo *AVW_LoadCompositeInfo(file)
char *file;
```

DESCRIPTION

AVW_LoadCompositeInfo() reads an *AVW_CompositeInfo* from the disk file called *file*. The color and opacity at each intensity is determined by the information within the file. The composite information is used by *AVW_RenderVolume()* to assign colors and opacities to voxels the rendering type is set to *AVW_VOLUME_COMPOSITING*.

RETURN VALUES

If successful *AVW_LoadCompositeInfo()* returns a pointer to an *AVW_CompositeInfo* structure. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_LoadCompositeInfo() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

BADOPEN
Could open file for reading or writing.

BADREAD
Error occurred while reading file.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_CreateCompositeInfo(), AVW_DestroyCompositeInfo(), AVW_SaveCompositeInfo(), AVW_CompositeInfo

NAME

AVW_LoadContourSurface − writes a contour surface to a file

SYNOPSIS

```
#include "AVW_Model.h"

AVW_ContourSurface* AVW_LoadContourSurface(filename, format, srfc)
char* filename;
int format;
AVW_ContourSurface *srfc;
```

DESCRIPTION

AVW_LoadContourSurface() reads a contour surface, from the file given by *filename. format* is used to determine which input format to use. Supported output formats are: *AVW_POGO_SURFACE, AVW_SLC_SURFACE,* and *AVW_SSD_ASCII_SURFACE. srfc* is provided as a means of reusing an existing *AVW_ContourSurface.*

RETURN VALUES

AVW_LoadContourSurface() returns a *AVW_ContourSurface* if successful. On failure, *NULL* will be returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure.

ERRORS

AVW_SaveContourSurface() will fail if one of the following is true:

BDREAD
File read failed.

BDOPEN
Unable to open file.

SEE ALSO

AVW_DestroyContourSurface(), AVW_SliceVolume(), AVW_ContourSurface, AVW_RPParam

NAME

AVW_LoadObjectMap – loads an object map

SYNOPSIS

```
#include "AVW_ObjectMap.h"

AVW_ObjectMap *AVW_LoadObjectMap(file)
char *file;
```

DESCRIPTION

AVW_LoadObjectMap() reads an *AVW_ObjectMap* from the disk file called *file*. The number of objects and contents of each object is determined by information within the file. The object map is used by AVW_RenderVolume() to distinguish seperate objects.

RETURN VALUES

If successful *AVW_LoadObjectMap()* returns a pointer to an *AVW_ObjectMap* structure. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_LoadObjectMap() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

BADOPEN
Could open file for reading or writing.

BADREAD
Error occurred while reading file.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_RenderVolume(), AVW_SaveObjectMap(), AVW_ObjectMap

NAME

AVW_LoadTiledSurface – reads an ordered surface from a file

SYNOPSIS

```
#include "AVW_Model.h"

AVW_TiledSurface *AVW_LoadTiledSurface(path, format, outSrfc)
char *path;
int format;
AVW_TiledSurface *outSrfc;
```

DESCRIPTION

AVW_LoadTiledSurface() reads an ordered surface from the file given by *path. Format* specifies which geometric file format is to be read. Supported formats are: *AVW_VRIO_SURFACE, AVW_DXF_SURFACE, AVW_OBJ_SURFACE, AVW_STL_SURFACE,* and *AVW_POLY_SURFACE. OutSrfc* is provided as a means of reusing an existing Kohonen Network. If *outSrfc* is NULL, a new vector list will be allocated.

RETURN VALUES

AVW_LoadTiledSurface() returns a ordered surface if successful. On failure, *NULL* will be returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure. In the event of failure, *outSrfc* is not guaranteed to hold meaningful data or even to exist.

ERRORS

AVW_LoadTiledSurface() will fail if one of the following is true:

CORRUPT
File is corrupt or not in correct format.

BADMAL
Malloc fails.

BDREAD
File read failed.

BDOPEN
Unable to open file.

SEE ALSO

AVW_DestroyTiledSurface(), AVW_SaveTiledSurface(), AVW_TileVolume(), AVW_DrawTiledSurface(), AVW_TiledSurface

NAME

AVW_LoadTree – loads a tree structure

SYNOPSIS

#include "AVW_Tree.h"

AVW_Tree *AVW_LoadTree(file)
char *file;

DESCRIPTION

AVW_LoadTree() reads an *AVW_Tree* from the disk file called *file*.

RETURN VALUES

If successful *AVW_LoadTree()* returns a pointer to an *AVW_Tree* structure. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_LoadTree() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

BADOPEN
Could open file for reading or writing.

RDERR
Error occurred while reading file.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddTreeChild(), AVW_CreateTree(), AVW_DestroyTree(), AVW_FindTreeIndex(), AVW_SaveTree(), AVW_TreePoint, AVW_Tree

NAME

AVW_LowpassFilterImage – performs a 2D Lowpass filter

SYNOPSIS

```
#include      "AVW_Filter.h"

AVW_Image *AVW_LowpassFilterImage(in_image, extents, out_image)
AVW_Image *in_image;
int extents[2];
AVW_Image *out_image;
```

DESCRIPTION

AVW_LowpassFilterImage() performs a Lowpass filter transformation on *in_image.* In the Lowpass filter each pixel value is replaced with an average pixel value in the rectangular neighborhood specified by *extents. Extents[0]* and *extents[1],* specify the x and y sizes respectively of the filter.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_LowpassFilterImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_LowpassFilterVolume(), AVW_AHEImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_UnsharpFilterEnhanceVolume(), AVW_Image

NAME

AVW_LowpassFilterVolume – performs a 3D Lowpass filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_LowpassFilterVolume(in_volume, extents, out_volume)
AVW_Volume *in_volume;
int extents[3];
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_LowpassFilterVolume() performs a Lowpass filter transformation on *in_volume.* In the Lowpass filter each voxel value is replaced with an average voxel value in the neighborhood specified by *extents. Extents[0], extents[1],* and *extents[2]* specify the x, y, and z sizes respectively of the filter.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_LowpassFilterVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_LowpassFilterImage(), AVW_AHEVolume(), AVW_OrthoGradFilterVolume() AVW_RankFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_VSFmeanFilterVolume(), AVW_Volume

NAME

AVW_MMapSelect – selects volume from multivolume mmapped file

SYNOPSIS

```
#include "AVW_ImageFile.h"

int AVW_MMapSelect(vol, which)
AVW_Volume *vol;
int which;
```

DESCRIPTION

AVW_MMapSelect() sets the selected volume of multi-volume file which has been previously memory mapped with function *AVW_MMapVolume.*

vol is the pointer to the memory mapped volume.

which is an integer indicating the volume number to select. Volumes are numbered from 0 to the number of volumes in the file less 1.

RETURN VALUES

Upon success *AVW_MMapSelect* returns *AVW_SUCCESS.* On failure *AVW_FAIL is* returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of the error.

ERRORS

Errors may occur for the following reasons:

MMAPER
vol is not a memory-mapped volume.

ILLVOL
illegal volume number.

SEE ALSO

AVW_MMapVolume()

NAME

AVW_MMapVolume – mmaps an image file to an AVW_Volume

SYNOPSIS

```
#include "AVW_ImageFile.h"

AVW_Volume *AVW_MMapVolume(imgfile)
AVW_ImageFile *imgfile;
```

DESCRIPTION

AVW_MMapVolume() uses the *UNIX mmap()* function to associate an image file to an *AVW_Volume*. This is equivalent to *AVW_ReadVolume()* except that the reading of the *imgfile* is delayed until the data within the *AVW_Volume* is actually accessed.

RETURN VALUES

Upon success *AVW_MMapVolume()* returns an *AVW_Volume*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

Errors may occur for the following reasons:

BADMAL
Bad Malloc. Memory allocation error.

MMAPER
Unmappable file.

SEE ALSO

AVW_CreateImageFile(), AVW_ExtendImageFile(), AVW_OpenImageFile(), AVW_ReadVolume(), AVW_ReadImageFile(), AVW_MMapSelect(), mmap()

NAME

AVW_MakeColorImage – makes a color image out of 3 grayscale images

SYNOPSIS

#include "AVW.h"

AVW_Image *AVW_MakeColorImage(red_image, green_image, blue_image, out_image)
AVW_Image *red_image;
AVW_Image *green_image;
AVW_Image *blue_image;
AVW_Image *out_image;

DESCRIPTION

AVW_MakeColorImage() creates a 24-bit *AVW_Image,* from one or more *AVW_UNSIGNED_CHAR* gray scale images.

NULL may be passed for up to two of the AVW_Images.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeColorImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MakeColorImage() will fail if:

ILLDT
Illegal Datatype.

SEE ALSO

AVW_MakeColorVolume, AVW_Image

NAME

AVW_MakeColorVolume – makes a color volume out of 3 grayscale volumes

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_MakeColorVolume(red_volume, green_volume, blue_volume,
        out_volume)
AVW_Volume *red_volume;
AVW_Volume *green_volume;
AVW_Volume *blue_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_MakeColorVolume() creates a 24-bit *AVW_Volume*, from one or more *AVW_UNSIGNED_CHAR* gray scale volumes.

NULL may be passed for up to two of the AVW_Volumes.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeColorVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MakeColorVolume() will fail if:

ILLDT
Illegal Datatype.

SEE ALSO

AVW_MakeColorImage, AVW_Volume

NAME

AVW_MakeComplexImageViewable – converts a complex image to a viewable data type

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_MakeComplexImageViewable(in_image, displaytype, halfflag, out_image)
AVW_Image *in_image;
int displaytype,halfflag;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MakeComplexImageViewable() converts *in_image* to the specified *displaytype.* Acceptable values for *displaytype,* as defined in *AVW.h,* are: *AVW_SPECTRUM_LOG_PHASE, AVW_SPECTRUM_PHASE_COLOR, AVW_SPECTRUM_LOG_COLOR, AVW_SPECTRUM_LOG_MAGNITUDE, AVW_SPECTRUM_MAGNITUDE, AVW_SPECTRUM_REAL, AVW_SPECTRUM_IMAGINARY.*

The most common source of complex images is *AVW_FFT2D.* These images are of dimension (Width / 2 +1)x(Height) relative to the original imput image. If *halfflag* is 1, *AVW_MakeComplexImageViewable* returns a viewable image of dimension (Width)x(Height). No intensity scaling is attempted during the conversion process.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeComplexImageViewable()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MakeComplexImageViewable() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for output volume.

ILLDT
Unknown or unsupported input or output displaytype.

SEE ALSO

AVW_ConvertVolume(), AVW_DitherImage(), AVW_IntensityScaleImage(), AVW_MakeGrayImage() AVW_Image

NAME

AVW_MakeFPointList2 – converts a list of points to floats

SYNOPSIS

```
#include "AVW.h"
AVW_FPointList2 *AVW_MakeFPointList2(plist, fplist)
AVW_PointList2 *plist;
AVW_FPointList2 *fplist;
```

DESCRIPTION

AVW_MakeFPointList2() builds an *AVW_FPointList2* from an *AVW_PointList2*.

fplist is provided as a method of reusing an existing *AVW_FPointList2*. Reuse is possible only if the size and data type of the provided *fplist* meet the requirements of the function. In this case the pointer to *fplist* is returned by the function. If not reusable *fplist* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeFPointList2()* returns an *AVW_FPointList2*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_CreateFPointList2(), AVW_CreatePointList2(), AVW_AddFPoint2(), AVW_DestroyFPointList2(), AVW_FPointList2, AVW_PointList

NAME

AVW_MakeGrayImage – converts a color image to a gray scale image

SYNOPSIS

#include "AVW.h"

AVW_Image *AVW_MakeGrayImage(in_image, out_image)
AVW_Image *in_image;
AVW_Image *out_image;

DESCRIPTION

AVW_MakeGrayImage() converts a 24-bit *AVW_Image*, or an *AVW_UNSIGNED_CHAR* image with a colormap, to a *AVW_UNSIGNED_CHAR* gray scale image.

Conversion is done using the following formula:
out_color = red * .30 + green * .59 + blue * .11;

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeGrayImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MakeGrayImage() will fail if:

ILLDT
Illegal Datatype.

SEE ALSO

AVW_ConvertImage(), AVW_ConvertVolume(), AVW_DitherImage(), AVW_DitherVolume(), AVW_MakeGrayVolume(), AVW_Image, AVW_Colormap

NAME

AVW_MakeGrayVolume – converts a color volume to a gray scale volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_MakeGrayVolume(in_volume, out_volume)
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_MakeGrayVolume() converts a 24-bit *AVW_Volume*, or an *AVW_UNSIGNED_CHAR* volume with a colormap, to a *AVW_UNSIGNED_CHAR* gray scale volume.

Conversion is done using the following formula:

out_color = red * .30 + green * .59 + blue * .11;

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeGrayVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MakeGrayVolume() will fail if:

ILLDT
Illegal Datatype.

SEE ALSO

AVW_ConvertVolume(), AVW_DitherVolume(), AVW_MakeGrayImage(), AVW_Colormap, AVW_Volume

NAME

AVW_MakeMaskFromTrace – makes a binary mask from a trace

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_MakeMaskFromTrace(trace, point, width, height,
                under_border, out_image)
AVW_PointList2 *trace;
AVW_Point2 *point;
int width, height, under_border;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MakeMaskFromTrace() creates a binary valued mask, *out_image,* in which all pixels connected to *point* and bounded by *trace* are set to 1 and all other pixels are set to 0. The dimensions of *out_image* are set to *width* and *height* and the datatype is *AVW_UNSIGNED_CHAR.* If *under_border* is set to *AVW_TRUE,* the pixels under the *trace* are set to 1. Otherwise they are set to 0.

In previous versions of *AVW* the *trace* was required to be 8 or 4 connected. Gaps in the *trace* are now filled in with linearly interpolated points.

The mask can be used to restrict the pixels of an image to be processed or sampled.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meets the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeMaskFromTrace()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_ComputeImageCentroid(), AVW_GetMaskBoundary(), AVW_ComputeImageFractalSig(), AVW_GetImageHistogram(), AVW_ComputeMEB(), AVW_ComputeRFF(), AVW_Compute2DShapeStats(), AVW_AVW_FillPointList2(), AVW_Image, AVW_PointList2, AVW_Point2

NAME

AVW_MakeMatrixFrom3Points – defines a matrix from 3 points

SYNOPSIS

```
#include "AVW.h"

AVW_Matrix *AVW_MakeMatrixFrom3Points(p1, p2, p3, xd, yd, zd, out_matrix)
AVW_Point3 *p1,*p2,*p3;
int xd, yd, zd;
AVW_Matrix *out_matrix;
```

DESCRIPTION

AVW_MakeMatrixFrom3Points() returns an *AVW_Matrix.* The matrix, *m,* defines a plane through the three given points, *p1, p2,* and *p3.* The point equidistant from all three points is taken as the translation point. The line segment from *p1* to *p3* is taken as the horizontal axis. This matrix, when passed to *AVW_GetOblique()* allows the user to extract an oblique plane which passes through the three points: *p1, p2,* and *p3.*

xd, yd, and *zd* specify the volume Width, Height, and Depth.

Out_matrix is provided as a method of reusing an existing *AVW_Matrix. (See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeMatrixFrom3Points()* returns an *AVW_Matrix.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. If the three points are colinear, they do not define a unique plane and *AVW_FAIL* will be returned.

ERRORS

AVW_MakeMatrixFrom3Points() will fail if:

ILLPAR
Illegal parameter. The three points are colinear or not unique.

SEE ALSO

AVW_CreateMatrix(), AVW_GetOblique(), AVW_MakeMatrixFromAxis(), AVW_InitializeOblique(), AVW_RotateMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_Point3, AVW_Matrix

NAME

AVW_MakeMatrixFromAxis – defines a plane perpendicular to an axis

SYNOPSIS

```
#include "AVW.h"

AVW_Matrix *AVW_MakeMatrixFromAxis(axis, xd, yd, zd, midpoint, out_matrix)
AVW_Line3 *axis;
int xd, yd, zd;
int midpoint;
AVW_Matrix *out_matrix;
```

DESCRIPTION

AVW_MakeMatrixFromAxis() returns an *AVW_Matrix.* The matrix defines a plane perpendicular to *axis.* The *midpoint* parameter specifies whether the plane is positioned at the start, *0,* end, *1,* or midpoint, *2,* of *axis.*

xd, yd, and *zd* specify the volume Width, Height, and Depth.

This matrix, when passed to *AVW_GetOblique()* allows the user to extract an oblique plane perpendicular to an axis.

Out_matrix is provided as a method of reusing an existing *AVW_Matrix. (See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeMatrixFromAxis()* returns an *AVW_Matrix.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MakeMatrixFromAxis() will fail if:

ILLPAR
Illegal parameter. The value for midpoint is undefined.

SEE ALSO

AVW_CreateMatrix(), AVW_MakeMatrixFrom3Points(), AVW_InitializeOblique(), AVW_GetOblique(), AVW_RotateMatrix(), AVW_TranslateMatrix(), AVW_ScaleMatrix(), AVW_Matrix, AVW_Line3

NAME

AVW_MakeMonoImage – Makes a monochromatic image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_MakeMonoImage(in_image, out_image)
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MakeMonoImage() makes a monochromatic (black and white) image from any *AVW_Image* by doing two color dithering.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeMonoImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MakeMonoImage() will fail if:

ILLDT
Illegal Datatype. In_image may be of any type except AVW_Complex.

SEE ALSO

AVW_DitherVolume(), AVW_DitherImage(), AVW_ConvertImage(), AVW_MakeGrayImage(), AVW_Image

NAME

AVW_MakeSpline – fits a spline to a set of points

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_MakeSpline(control, step, close_flag, spline)
AVW_PointList2 *control;
double step;
int close_flag;
AVW_PointList2 *spline;
```

DESCRIPTION

AVW_MakeSpline() fits a smooth curve to a set of *control* points. The *step* parameter specifies the spacing of points on the smooth curve between which linearly interpolated points are filled. A value of .02 is a good default. The *close_flag* specifies whether the *control* points represent a closed (1) or open (0) trace. The resultant smoothed trace is stored in *spline.*

The list structures are managed and memory for the structures is reallocated as needed. These structures are defined in *AVW.h* and are commonly used for traces and stacks.

Spline is provided as a method of reusing an existing *AVW_PointList2.*

RETURN VALUES

If successful *AVW_MakeSpline()* returns an *AVW_PointList2.* On failure it returns *AVW_NULL* and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_MakeSpline() will fail if:

ILLPAR
    An illegal parameter, NULL control, was passed to the routine.

SEE ALSO

AVW_AddFPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_ExtractControlPoints(), AVW_FillPointList2(), AVW_PointList2, AVW_Point2

NAME

AVW_MakeTree – builds tree files from a thinned volume

SYNOPSIS

```
#include "AVW_Tree.h"

AVW_Tree *AVW_MakeTree(in_volume, start_pt, minlen, maxlen)
AVW_Volume *in_volume;
AVW_Point3 *start_pt;
int minlen, maxlen;
```

DESCRIPTION

AVW_MakeTree() traverses a thinned volume and and creates an *AVW_Tree* structure containing the coordinates of the tree. One tree will be created based on the first connected component encountered within the volume. This voxels of *in_volume* are set to zero as points are added to the tree. The user can check if there are any potential trees left in the *in_volume* by using *AVW_IsVolumeZero().* The tree coordinates will always start at an endpoint.

In_volume is the input volume must be thinned. See *AVW_Thin3D().*

Start_pt specifies the starting point of the tree. *Minlen* specifies the minimum length of an entire tree. Trees with fewer *minlen* points will be ignored. *Maxlen* specifies the maximum length of an entire tree. Trees with more than *maxlen* points will be ignored. If a zero or negative *maxlen* is specified the upper limit on the tree size will be disregarded.

RETURN VALUES

If successful *AVW_MakeTree()* returns an *AVW_Tree.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. *NULL* may also be returned if the tree found by the routine is not within the specified size limits.

ERRORS

AVW_MakeTree() will fail if:

ILLDT
: Data type is not AVW_UNSIGNED_CHAR.

ILLPT
: Illegal starting point.

BDOPEN
: Could not open file.

SEE ALSO

AVW_Thin3D(), AVW_FindTreeStart(), AVW_FillHolesImage(), AVW_Volume

NAME

AVW_MakeVolumeFromImage – makes a volume from an image

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_MakeVolumeFromImage(image, volume)
AVW_Image *image;
AVW_Volume *volume;
```

DESCRIPTION

AVW_MakeVolumeFromImage() creates a one slice *AVW_Volume* from an *AVW_Image.*

Volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *volume* meets the requirements of the function. In this case the pointer to *volume* is returned by the function. If not reuseable *volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MakeVolumeFromImage()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_CreateVolume(), AVW_Image, AVW_Volume

NAME

AVW_Malloc – allocates system memory

SYNOPSIS

```
#include "AVW.h"

void *AVW_Malloc(size)
unsigned int size;

void *AVW_Calloc(num, size)
unsigned int num;
unsigned int size;

void *AVW_Realloc(size, ptr)
unsigned int size;
void *ptr;

void AVW_Free(ptr)
void *ptr;
```

DESCRIPTION

These procedures provide a platform and compiler independent interface for memory allocation. Programs that need to transfer ownership of memory blocks between *AVW* and other modules should use these routines rather than the native *malloc()* and *free()* routines provided by the *C* run-time library.

AVW_Malloc returns a pointer to a size bytes suitably aligned for any use.

AVW_Calloc allocates space for an array nelem elements of *size* elsize. The space is initialized to zeros.

Tcl_Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the new block. The contents will be unchanged up to the lesser of the new and old sizes. The returned location may be different from *ptr.*

Tcl_Free makes the space referred to by *ptr* available for further allocation.

SEE ALSO

malloc(), free()

NAME

AVW_MaskImageToSampleFile – creates a multi-spectral sample file from a mask image

SYNOPSIS

```
#include "AVW.h"

int AVW_MaskImageToSampleFile(imgs, numimgs, maskImage, SampleFile)

AVW_Image **imgs;
int numimgs;
AVW_Image *maskImage;
char *SampleFile;
```

DESCRIPTION

AVW_MaskImageToSampleFile() generates an AVW multi-spectral sample file from a mask image and a list of corresponding spatially correlated images. This file can be used to perform image classification with *AVW_ClassifyImageFromSampleFile()* or volumes with *AVW_ClassifyVolumeFromSampleFile()*.

Classification from a Sample File makes it possible to perform multi-spectral classification of images and volumes based on standardized class definitions.

Each non-zero pixel from the *maskImage* generates an row entry in the *SampleFile* which consists of the class number (the value of that maskImage pixel) and the values of the corresponding pixels from the list of images pointed at by *Imgs*.

Imgs is a list of spatially correlated images.

Numimgs is the number of images in *Imgs*

MaskImage is an and AVW_Image of DataType AVW_UNSIGNED_CHAR and the same size as the images in *Imgs* in which pixels to be used as training samples have non-zero values.

SampleFile is the name of the text file which is created by the function.

Example Sample File contents

```
AVW Multispectral Classification Samples File
Classes=6
Bands=4
1       48.0000 27.0000 52.0000 69.0000
1       61.0000 31.0000 61.0000 85.0000
2       59.0000 44.0000 86.0000 91.0000
2       58.0000 40.0000 78.0000 88.0000
2       59.0000 42.0000 82.0000 88.0000
```

The first line of the file contains a signature identifying the contents of the file. The second line indicates how many classes are described in the file. The third row indicates the number of samples in a row.

AVW_MaskImageToSampleFile() returns an *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyImage will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLIMG
Illegal Image. The images are not all the same dimension.

BDTRSM
Bad Training Sample. The supplied training samples were unusable.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO

AVW_MaskVolumeToSampleFile(), AVW_ClassifyVolumeFromSampleFile(), AVW_ClassifyImageFromSampleFile(), AVW_Image

NAME

AVW_MaskVolumeToSampleFile – creates a multi-spectral sample file from a mask volume

SYNOPSIS

```
#include "AVW.h"

int AVW_MaskVolumeToSampleFile(vols, numvols, maskVolume, SampleFile)

AVW_Volume **vols;
int numvols;
AVW_Volume *maskVolume;
char *SampleFile;
```

DESCRIPTION

AVW_MaskVolumeToSampleFile() generates an AVW multi-spectral sample file from a mask volume and a list of corresponding spatially correlated volumes. This file can be used to perform image classification with *AVW_ClassifyImageFromSampleFile()* or volumes with *AVW_ClassifyVolumeFromSampleFile().*

Classification from a Sample File makes it possible to perform multi-spectral classification of images and volumes based on standardized class definitions.

Each non-zero voxel from the *maskVolume* generates an row entry in the *SampleFile* which consists of the class number (the value of that maskVolume voxel) and the values of the corresponding voxels from the list of volumes pointed at by *Vols.*

Vols is a list of spatially correlated volumes.

Numvols is the number of vols in *vols*

MaskVolume is an and AVW_Volume of DataType AVW_UNSIGNED_CHAR and the same size as the volumes in *vols* in which voxels to be used as training samples have non-zero values.

SampleFile is the name of the text file which is created by the function.

Example Sample File contents

```
AVW Multispectral Classification Samples File
Classes=6
Bands=4
1       48.0000 27.0000 52.0000 69.0000
1       61.0000 31.0000 61.0000 85.0000
2       59.0000 44.0000 86.0000 91.0000
2       58.0000 40.0000 78.0000 88.0000
2       59.0000 42.0000 82.0000 88.0000
```

The first line of the file contains a signature identifying the contents of the file. The second line indicates how many classes are described in the file. The third row indicates the number of samples in a row.

AVW_MaskVolumeToSampleFile() returns an *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyImage will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLIMG
Illegal Image. The images are not all the same dimension.

BDTRSM
Bad Training Sample. The supplied training samples were unusable.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

SEE ALSO

AVW_MaskImageToSampleFile(), AVW_ClassifyVolumeFromSampleFile(), AVW_ClassifyImageFromSampleFile(), AVW_Volume

NAME

AVW_MatchImageHistogram – matches the intensity distribution of an image to a histogram

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Image *AVW_MatchImageHistogram(in_image, mhisto, out_image)
AVW_Image *in_image;
AVW_Histogram *mhisto;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MatchImageHistogram() forces the histogram of *in_image* to match the grey scale distribution of *mhisto.* The results are returned in *out_image.*

Images of the same subject should have the same general distribution of grey levels, even though the parameters of a particular image (exposure, brightness, contrast) may vary widely. Histogram matching may be used to normalize the absolute greyscale values of a set of images to a selected "optimal" example.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.).*

RETURN VALUES

If successful *AVW_MatchImageHistogram()* returns an *AVW_Image* which has been matched to the provided histogram. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MatchImageHistogram() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

ILLHIS
Histogram provided was not valid.

SEE ALSO

AVW_MatchVolumeHistogram(), AVW_CreateHistogram(), AVW_FlattenImageHistogram(), AVW_GetImageHistogram(), AVW_NormalizeHistogram(), AVW_PreserveImageHistogram(), AVW_VerifyHistogram(), AVW_Histogram, AVW_Image

NAME

AVW_MatchSurfaces – generates the matrix required to register volumes

SYNOPSIS

```
#include "AVW_SurfaceMatch.h"

AVW_MatchResult *AVW_MatchSurfaces(base_volume, match_volume, free_flag, param)
AVW_Volume *base_volume, *match_volume;
int free_flag;
AVW_MatchParameters *param;
```

DESCRIPTION

AVW_MatchSurfaces() determines the geometric transformation parameters required to spatially register two volumes. The transformation parameters include: 3D translation, 3D rotation, and 3D scaling. The parameters are output in *Matrix,* which is a member of the *AVW_MatchResults* structure. Image registration can be achieved by transforming a input volume, most likely the unedited version of the *match_volume,* by the matrix. Note that both the *base_volume* and *match_volume* need to have the "VoxelWidth", "VoxelHeight" and "VoxelDepth" set in the *Info* string of their structure (See *AVW_PutNumericInfo()).*

Surfaces are extracted from both the *base_volume* and *match_volume.* Both volumes must be presegmented *AVW_UNSIGNED_CHAR AVW_Volumes.* All surface points in the *match_volume* should should have corresponding points on the surface of the *base_volume.*

Free_flag, if set to *AVW_TRUE,* allows the *base_volume* and *match_volume* to be freed after the surface has been extracted. This will cause the function to run more efficiently, as copies are avoided and less memory is used. This option should not be enabled if an iteration will be preformed. If this option is enabled, the *base_volume* and *match_volume* parameters should be set to *NULL* when the function returns.

For the *base_volume:* if the data is non-cubic, shape-based interpolation is performed. Surface points are defined as any non-zero valued voxel with a zero valued 6 connected neighbor.

For the *match_volume:* surface points are defined as any non-zero valued pixel with a zero valued 4 connected neighbor.

param->SamplePoints indicates the number of uniformly sampled points to be used to evaluate the *MeanSquareDistance* (MSD) between the two surfaces.

The matching process will search through the parameter space to determine the parameter set which minimizes the MSD. The initial best guess and search range is supplied by the *AVW_MatchParameters.* The search will start from grid points uniformly distributed within the search range. A *RotationInterval* is also specified in the *AVW_MatchParameters* structure, but the translation interval is calculated during the matching process. A pyramid multi-resolutional approach is used in the searching process. The search starts from the coarsest resolution level, and a number of starting points are rejected at each resolution level. The parameter set with the smallest (MSD) error is stored in the *AVW_MatchResults Matrix.*

> *Base_volume* and *match_volume* specify the *AVW_Volumes* to match.
>
> *Param->SamplePoints* specifies the number of surface points from the *match_volume* to be used in the matching process.
>
> If *param->Centroid* is set to *AVW_TRUE* the initial position is set by registering the centroid of the *match_volume* to the centroid of the *base_volume.* If set to *AVW_FALSE,* the *param->TranslationX* (+/- width (largest of *base_volume* or *match_volume)), param->TranslationY* (+/- height (largest of *base_volume* or

match_volume)), and *param->TranslationZ* (+ / - # slices (largest of *base_volume* or *match_volume)),* are used as the initial match position.

Param->TranslationRange (0.0 to 100.0) specifies the percent search range for the three translation parameters.

Param->RotationPrecession (0.0 to 360.0 degrees), *param->RotationNutation* (0.0 to 180.0 degrees), and *param->RotationSpin* (0.0 to 360.0 degrees) specify the initial rotation guess.

Param->RotationRange (0.0 to 100.0) specifies the percent search range for the three rotation parameters.

Param->RotationInterval (0.0 to 360.0 degrees) specifies the interval of starting points along each rotation parmater axis.

The recommended *AVW_MatchParamenters* to search the entire volume space are:

param->SamplePoints = 100

param->Centroid = *AVW_TRUE*

param->TranslationRange = 100.0

param->RotationPrecession = *param->RotationNutation* = *param->RotationSpin* = 0.0

param->RotationRange = 100.0

param->RotationInterval = 30.0

More detailed information about the surface matching algorithm may be found in the reference:

Jiang, H., K. Holton and R. Robb: Image registration of multimodality 3-D medical images by chamfer matching. Proceedings of Vision and Visualization, SPIE, San Jose, CA, Feb. 1992, pp. 649-659.

RETURN VALUES

If successful *AVW_MatchSurfaces()* returns a pointer to an *AVW_MatchResult* structure.

This structure contains:

Matrix - homogeneous 4x4 matrix used to register *match_volume* to *base_volume.* A programmer could use *AVW_InvertMatrix()* to generate the inverse and match *base_volume* to *match_volume.*

MeanSquareDistance - average distance from each match point to the closest base point. A lower value indicates a better match.

NextInput - a pointer to an *AVW_MatchParameters* structure which contains the suggested parameters used for an iterative match.

The programmer is responsible for freeing the memory associated with the *AVW_MatchResult* structure when it is no longer needed.

Example:

free(match_results->NextInput);
free(match_results);

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MatchSurfaces() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

SEE ALSO

AVW_TransformVolume(), AVW_MatchParameters, AVW_MatchResult, AVW_Matrix, AVW_Volume

NAME

AVW_MatchVolumeHistogram – matches the intensity distribution of a volume to a histogram

SYNOPSIS

#include "AVW_Histogram.h"

AVW_Volume *AVW_MatchVolumeHistogram(in_vol, mhisto, out_vol)
AVW_Volume *in_vol;
AVW_Histogram *mhisto;
AVW_Volume *out_vol;

DESCRIPTION

AVW_MatchVolumeHistogram() forces the histogram of *in_volume* to match the grey scale distribution of *mhisto.*
The results are returned in *out_volume.*

Volumes of the same subject should have the same general distribution of grey levels, even though the parameters of a particular volume (exposure, brightness, contrast) may vary widely. Histogram matching may be used to normalize the absolute greyscale values of a set of volumes to a selected "optimal" example.

Out_vol is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_vol* meet the requirements of the function. In this case the pointer to *out_vol* is returned by the function. If not reusable *out_vol* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.).*

RETURN VALUES

If successful *AVW_MatchVolumeHistogram()* returns an *AVW_Volume* which has been matched to the provided histogram. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MatchVolumeHistogram() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

ILLHIS
Histogram provided was not valid.

SEE ALSO

AVW_MatchImageHistogram(), AVW_CreateHistogram(), AVW_DestroyHistogram(), AVW_FlattenVolumeHistogram(), AVW_GetVolumeHistogram(), AVW_NormalizeHistogram(), AVW_PreserveVolumeHistogram(), AVW_VerifyHistogram(), AVW_Histogram. AVW_Volume

NAME

AVW_MatchVoxels – generates the matrix required to register volumes

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_Matrix *AVW_MatchVoxels(base_volume, match_volume, param, matrix)
AVW_Volume *base_volume, *match_volume;
AVW_MatchVoxelParams *param;
AVW_Matrix *matrix;
```

DESCRIPTION

AVW_MatchVoxels() determines the geometric transformation parameters required to spatially register the *base_volume* and *match_volume* based on the matchin parameters in *param*. The transformation parameters include: 3D translation, 3D rotation, and 3D scaling. The function return an *AVW_Matrix*, with the matching transformation.

The *AVW_MatchVoxels()* function will use the mathcing parameters as deinfed in the *param* structure.

matrix is provided as a method of re-using an existing *AVW_Matrix*.

RETURN VALUES

If successful returns a pointer to an *AVW_Matrix* structure which contains an homogeneous 4x4 matrix used to register *match_volume* to *base_volume.*

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MatchVoxels() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

SEE ALSO

AVW_TransformVolume(), AVW_MatchVoxelParams, AVW_Matrix, AVW_Volume.

NAME

AVW_MatrixAngles – determines rotation angles from a transformation matrix

SYNOPSIS

```
#include "AVW.h"

void AVW_MatrixAngles(mat, xangle, yangle, zangle)
AVW_Matrix *mat;
double *xangle;
double *yangle;
double *zangle;
```

DESCRIPTION

AVW_MatrixAngles() returns the rotation angles *xangle, yangle,* and *zangle,* from the AVW_Matrix, *mat.*

SEE ALSO

AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_InvertMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis() AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_SetIdentityMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_Matrix

NAME

AVW_MaximumDataValue – returns maximum data type value

SYNOPSIS

```
#include     "AVW.h"

double AVW_MaximumDataValue(type)
int type;
```

DESCRIPTION

AVW_MaximumDataValue() returns the maximum data value for the specified data type.

Type specifies the *AVW* data type. Acceptable values are: *AVW_UNSIGNED_CHAR*, *AVW_SIGNED_CHAR*, *AVW_UNSIGNED_SHORT*, *AVW_SIGNED_SHORT*, *AVW_UNSIGNED_INT*, *AVW_SIGNED_INT*, *AVW_FLOAT*, and *AVW_COLOR*.

RETURN VALUES

If successful *AVW_MaximumDataValue()* returns the maximum value. On failure it returns 0.0 and sets *AVW_ErrorMessage* and *AVW_ErrorNumber* to appropriate values. *AVW_ErrorNumber* should be checked after calling this function.

ERRORS

AVW_MaximumDataValue will fail if:

NOTSUP
AVW_COMPLEX data type not supported.

SEE ALSO

AVW_MinimumDataValue(), AVW_QuickImageMaxMin(), AVW_QuickVolumeMaxMin()

NAME

AVW_MedialAxisTransformVolume – creates a medial axis or surface

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_MedialAxisTransformVolume(in_volume, axis_flag, out_volume)
AVW_Volume *in_volume;
int axis_flag;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_MedialAxisTransformVolume() performs the medial axis transformations to produce a medial surface or axis.

In_volume and *out_volume* must be of the data type *AVW_UNSIGNED_CHAR*. *In_volume* does not have to be a binary valued but all nonzero pixels are treated as ones. *Axis_flag* is set to AVW_MEDIAL_AXIS (1) or AVW_MEDIAL_SURFACE (0) to specify what type of output will be generated and returned.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MedialAxisTransformVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MedialAxisTransformVolume() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

SEE ALSO

AVW_ThinVolume(), AVW_Volume

NAME

AVW_MedianFilterImage – performs a 2D median filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_MedianFilterImage(in_image, xdim, ydim, out_image)
AVW_Image *in_image;
int xdim, ydim;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MedianFilterImage() performs a median filter transformation on *in_image. Xdim* and *ydim,* specify the x and y sizes respectively of the filter. Filter sizes should be odd.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MedianFilterImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_MedianFilterVolume(), AVW_AHEImage(), AVW_LowpassFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_SobelFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_Image

NAME

AVW_MedianFilterVolume – performs a 3D median filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_MedianFilterVolume(in_volume, xdim, ydim, zdim, out_volume)
AVW_Volume *in_volume;
int xdim, ydim, zdim;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_MedianFilterVolume() performs a median filter transformation on *in_volume. Xdim, ydim,* and *zdim* specify the x, y, and z sizes respectively of the filter. Filter sizes should be odd.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MedianFilterVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_MedianFilterImage(), AVW_AHEVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_SobelFilterVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_MergeInfo – merges two AVW information strings

SYNOPSIS

```
#include "AVW.h"

char *AVW_MergeInfo(info1, info2)
char *info1;
char *info2;
```

DESCRIPTION

AVW_MergeInfo() merges the *AVW* information strings, *info1* and *info2*. If common information occurs in both strings, *info1* takes precedence. If either string is NULL, the returned string is a copy of the non-NULL string.

Info2 is freed at completion to make this function compatible with many of the *AVW* functions which reuse memory. This suggests that the same character string should be used as the return string and *info2*.

Example:

```
img->Info = AVW_MergeInfo(vol->Info, img->Info);
```

RETURN VALUES

If successful *AVW_MergeInfo()* returns a char *. This char * will be managed automatically if part of an *AVW* structure, but must be freed by the user if used in another way. On failure, *NULL* is returned.

SEE ALSO

AVW_GetNumericInfo(), AVW_GetStringInfo(), AVW_ListInfo(), AVW_PutHistoryInfo(), AVW_PutNumericInfo(), AVW_PutStringInfo(), AVW_Image, AVW_Volume, AVW_ImageFile

NAME

AVW_MergeRendered – merges two rendered images

SYNOPSIS

```
#include "AVW.h"

AVW_RenderedImage *AVW_MergeRendered(in1_rendered, factor1, in2_rendered,
                    factor2, merge_type, out_rendered)
AVW_RenderedImage *in1_rendered;
double factor1;
AVW_RenderedImage *in2_rendered;
double factor2;
int merge_type;
AVW_RenderedImage *out_rendered;
```

DESCRIPTION

AVW_MergeRendered() combines two rendered images to produce and return a third. *Factor1* and *factor2*, specify and opacity value between *1.0* (Opaque) and *0.0* (Transparent). *Factor1* is used when a voxel from *in1_rendered* is in front, *Factor2 is used when in_rendered2* is in front.

Merge_type should be set to one of the following: *AVW_MERGE_DEPTH, AVW_MERGE_AVERAGE*, or *AVW_MERGE_BRIGHTEST. AVW_MERGE_DEPTH* causes the rendererings to be merged according to the depth information stored in the *PBuffer* member of the *AVW_RenderedImage's. AVW_MERGE_AVERAGE* is used to Add and Average the renderings. *AVW_MERGE_BRIGHTEST* causes the birghtest pixel in the either rendering to be used.

Out_rendered is provided as a method of reusing an existing *AVW_RenderedImage.* Reuse is possible only if the size and data type of the provided *out_rendered* meet the requirements of the function. In this case the pointer to *out_rendered* is returned by the function. If not reuseable *out_rendered* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MergeRendered()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MergeRendered() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

SEE ALSO

AVW_RenderVolume(), AVW_RenderOblique(), AVW_CubeSections(), AVW_IntersectingSections(), AVW_DestroyRenderedImage() AVW_RenderedImage()

NAME

AVW_MinimumDataValue – returns minimum data type value

SYNOPSIS

```
#include "AVW.h"

double AVW_MinimumDataValue(type)
int type;
```

DESCRIPTION

AVW_MinimumDataValue() returns the minimum data value for the specified data type.

Type specifies the *AVW* data type. Acceptable values are: *AVW_UNSIGNED_CHAR*, *AVW_SIGNED_CHAR*, *AVW_UNSIGNED_SHORT*, *AVW_SIGNED_SHORT*, *AVW_UNSIGNED_INT*, *AVW_SIGNED_INT*, *AVW_FLOAT*, and *AVW_COLOR*

RETURN VALUES

If successful *AVW_MinimumDataValue()* returns the minimum value. On failure it returns 0.0 and sets *AVW_ErrorMessage* and *AVW_ErrorNumber* to appropriate values. *AVW_ErrorNumber* should be checked after calling this function.

ERRORS

AVW_MinimumDataValue will fail if:

NOTSUP
AVW_COMPLEX data type not supported.

SEE ALSO

AVW_MaximumDataValue(), AVW_QuickImageMaxMin(), AVW_QuickVolumeMaxMin()

NAME

AVW_MirrorMatrix – mirrors a transformation matrix

SYNOPSIS

```
#include "AVW.h"

AVW_Matrix *AVW_MirrorMatrix(in_matrix, axis, out_matrix)
AVW_Matrix *in_matrix;
int axis;
AVW_Matrix *output_matrix;
```

DESCRIPTION

AVW_MirrorMatrix() is used to mirror the *AVW_Matrix*, *mat,* on one or more axes. *Axis* should be set a sum of *AVW_XAXIS, AVW_YAXIS,* and *AVW_ZAXIS* to indicate which axis the matrix is mirrored. The order of mirroring is X, then Y, then Z.

Out_matrix is provided as a method of reusing an existing *AVW_Matrix. (See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MirrorMatrix()* returns an *AVW_Matrix.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_InvertMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis() AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_SetIdentityMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_Matrix

NAME

AVW_MirrorRendered – mirrors a rendered image

SYNOPSIS

```
#include "AVW_Render.h"

AVW_Image *AVW_MirrorRendered(rendered, direction, position, out_image)
AVW_RenderedImage *rendered;
int direction, position;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MirrorRendered() mirrors a rendered image in a specified *direction* and at a specified *position.*

Direction must be one of the following:

AVW_MIRROR_RIGHT_TO_LEFT

AVW_MIRROR_LEFT_TO_RIGHT

AVW_MIRROR_BOTTOM_TO_TOP

AVW_MIRROR_TOP_TO_BOTTOM

Position is any location from 0 to (*rendered->Image->Width - 1*).

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MirrorRendered()* returns an *AVW_Image.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MirrorRendered() will fail if one or more of the following is true:

BADMAL
Couldn't allocate enough memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_RenderVolume(), AVW_RenderedImage, AVW_Image

NAME

AVW_MorphCloseImage – morphologically closes an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_MorphCloseImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MorphCloseImage() performs binary morphological closing on *in_image* using *element* as the structuring element.

In_image does not have to be a binary valued image but all nonzero pixels are treated as ones. *In_image, out_image,* and *element* must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

The closing operation is defined as a dilation followed by an erosion with the same structuring element.

AVW_CreateStructuringImage() or the combination of: *AVW_CreateImage(), AVW_SetImage(),* and *AVW_PutPixel()* may be used to create a structuring element.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphCloseImage()* returns a processed *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphCloseImage() will fail if:

BADMAL
Malloc failed. Unable to allocate memory for results.

ILLDT
Data type is not defined or supported.

ILLIMG
An illegal image was passed to the function.

SEE ALSO

AVW_ConditionalDilateImage(), AVW_CreateImage(), AVW_CreateStructuringImage(), AVW_CreateStructuringImage(), AVW_DilateImage(), AVW_ErodeImage(), AVW_MorphCloseVolume(), AVW_MorphOpenImage(), AVW_MorphMaxImage(), AVW_MorphMinImage(), AVW_PutPixel(), AVW_PutVoxel(), AVW_SetImage(), AVW_SetVolume(), AVW_Image

NAME

AVW_MorphCloseVolume – morphologically closes a volume

SYNOPSIS

```
#include      "AVW.h"

AVW_Volume *AVW_MorphCloseVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_MorphCloseVolume() performs binary morphological closing on *in_volume* using *element* as the structuring element.

In_volume does not have to be a binary valued volume but all nonzero voxels are treated as ones. *In_volume, out_volume,* and *element* must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

The closing operation is defined as a dilation followed by an erosion with the same structuring element.

AVW_CreateSTructuringVolume() or a combination of: *AVW_CreateVolume(), AVW_SetVolume(),* and *AVW_PutVoxel()* may be used to create a structuring element.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphCloseVolume()* returns a processed *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphCloseVolume() will fail if:

BADMAL
Malloc failed. Unable to allocate memory for results.

ILLDT
Data type is not defined or supported.

ILLVOL
An illegal volume was passed to the function.

SEE ALSO

AVW_ConditionalDilateVolume(), AVW_CreateStructuringVolume(), AVW_CreateVolume(), AVW_DilateVolume(), AVW_ErodeVolume(), AVW_MorphCloseImage(), AVW_MorphOpenVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume(), AVW_PutVoxel(), AVW_SetVolume(), AVW_Volume

NAME

AVW_MorphMaxImage – performs a 2D local maximum operation

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_MorphMaxImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MorphMaxImage() performs a local maximum operation on *in_image* using *element* as the structuring element.

Element must be of data type *AVW_UNSIGNED_CHAR*.

The local maximum operation is the grey scale equivalent of a binary dilation.This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

In simplest terms the local maximum process takes place by translating the structuring element so that its centerpoint lies on every point of the data. The maximum value of the pixels in the data corresponding to the nonzero pixels in the structuring element is set at this point in the result data.

AVW_CreateStructuringImage() or a combination of: *AVW_CreateImage()*, *AVW_SetImage()*, and *AVW_PutPixel()* may be used to create a structuring element.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphMaxImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphMaxImage() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLIMG
: An illegal image was passed to the function.

SEE ALSO

AVW_ConditionalDilateImage(), AVW_CreateImage(), AVW_CreateStructuringImage(), AVW_DilateImage(), AVW_ErodeImage(), AVW_MorphCloseImage(), AVW_MorphMaxVolume(), AVW_MorphMinImage(), AVW_MorphOpenImage(), AVW_PutPixel(), AVW_SetImage(), AVW_Image

NAME

AVW_MorphMaxVolume – performs a 3D local maximum operation

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_MorphMaxVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_MorphMaxVolume() performs a local maximum operation on *in_volume* using *element* as the structuring element. *Element* must be of data type *AVW_UNSIGNED_CHAR.*

The local maximum operation is the grey scale equivalent of a binary dilation. This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

In simplest terms the local maximum process takes place by translating the structuring element so that its centerpoint lies on every point of the data. The maximum value of the voxels in the data corresponding to the nonzero voxels in the structuring element is set at this point in the result data.

AVW_CreateStructuringVolume() or a combination of: *AVW_CreateVolume(), AVW_SetVolume(),* and *AVW_PutVoxel()* may be used to create a structuring element.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphMaxVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphMaxVolume() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLVOL
: An illegal volume was passed to the function.

SEE ALSO

AVW_ConditionalDilateVolume(), AVW_CreateVolume(), AVW_DilateVolume(), AVW_ErodeVolume(), AVW_MorphCloseVolume(), AVW_MorphMaxImage(), AVW_MorphOpenVolume(), AVW_MorphMinVolume(), AVW_PutVoxel(), AVW_SetVolume(), AVW_CreateStructuringImage(), AVW_Image

NAME

AVW_MorphMinImage – performs a 2D local minimum operation

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_MorphMinImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MorphMinImage() performs a local minimum operation on *in_image* using *element* as the structuring element.

Element must be of data type *AVW_UNSIGNED_CHAR.*

The local minimum operation is the grey scale equivalent of a binary erosion.

In simplest terms the local minimum process takes place by translating the structuring element so that its centerpoint lies on every point of the data. The minimum value of the pixels in the data corresponding to the nonzero pixels in the structuring element is set in the result data.

AVW_CreateStructureingImage or a combination of: *AVW_CreateImage(), AVW_SetImage(),* and *AVW_PutPixel()* may be used to create a structuring element.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphMinImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphMinImage() will fail if:

BADMAL
Malloc failed. Unable to allocate memory for results.

ILLDT
Data type is not defined or supported.

ILLIMG
An illegal image was passed to the function.

SEE ALSO

AVW_ConditionalDilateImage(), AVW_CreateImage(), AVW_CreateStructuringImage(), AVW_DilateImage(), AVW_ErodeImage(), AVW_MorphCloseImage(), AVW_MorphMinVolume(), AVW_MorphMaxImage(), AVW_MorphOpenImage(), AVW_PutPixel(), AVW_SetImage(), AVWImage

NAME

AVW_MorphMinVolume – performs a 3D local minimum operation

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_MorphMinVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_MorphMinVolume() performs a local minimum operation on *in_volume* using *element* as the structuring element.

Element must be of data type *AVW_UNSIGNED_CHAR*.

The local minimum operation is the grey scale equivalent of a binary erosion.

In simplest terms the local minimum process takes place by translating the structuring element so that its centerpoint lies on every point of the data. The minimum value of the voxels in the data corresponding to the nonzero voxels in the structuring element is set in the result data.

AVW_CreateStructuringVolume() or a combination of *AVW_CreateVolume()*, *AVW_SetVolume()*, and *AVW_PutVoxel()* may be used to create a structuring element.

Out_volume is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphMinVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphMinVolume() will fail if:

BADMAL
Malloc failed. Unable to allocate memory for results.

ILLDT
Data type is not defined or supported.

ILLVOL
An illegal volume was passed to the function.

SEE ALSO

AVW_MorphMinImage(), AVW_ConditionalDilateVolume(), AVW_CreateStructuringVolume(), AVW_CreateVolume(), AVW_DilateVolume(), AVW_ErodeVolume(), AVW_MorphCloseVolume(), AVW_MorphOpenVolume(), AVW_MorphMaxVolume(), AVW_SetVolume(), AVW_PutVoxel(), AVW_Volume

NAME

AVW_MorphOpenImage – morphologically opens an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_MorphOpenImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_MorphOpenImage() performs binary morphological opening on *in_image* using *element* as the structuring element.

In_image does not have to be a binary valued but all nonzero pixels are treated as ones. *In_image*, *out_image*, and *element* must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_image* and *out_image out_volume* are the same.

The opening is defined as an erosion of the data followed by a dilation of the data with the same structuring element. The result data will only contain ones and zeros.

AVW_CreateImage() or a combination of *AVW_CreateImage()*, *AVW_SetImage()*, and *AVW_PutPixel()* may be used to create a structuring element.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphOpenImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphOpenImage() will fail if:

BADMAL
: Malloc failed. Unable to allocate memory for results.

ILLDT
: Data type is not defined or supported.

ILLIMG
: An illegal image was passed to the function.

SEE ALSO

AVW_MorphOpenVolume(), AVW_ConditionalDilateImage(), AVW_CreateImage(), AVW_CreateStructuringImage(), AVW_DilateImage(), AVW_ErodeImage(), AVW_MorphCloseImage(), AVW_MorphMaxImage(), AVW_MorphMinImage(), AVW_SetImage(), AVW_PutPixel(), AVW_Image

NAME

AVW_MorphOpenVolume – morphologically opens a volume

SYNOPSIS

#include "AVW.h"

AVW_Volume *AVW_MorphOpenVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;

DESCRIPTION

AVW_MorphOpenVolume() performs binary morphological opening on *in_volume* using *element* as the structuring element.

In_volume does not have to be a binary valued but all nonzero pixels are treated as ones. *In_volume, out_volume,* and *element* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

The opening is defined as an erosion of the data followed by a dilation of the data with the same structuring element. The result data will only contain ones and zeros.

AVW_CreateStructuringVolume() or a combination of *AVW_CreateVolume(), AVW_SetVolume(),* and *AVW_PutVoxel()* may be used to create a structuring element.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MorphOpenVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_MorphOpenVolume() will fail if:

BADMAL
Malloc failed. Unable to allocate memory for results.

ILLDT
Data type is not defined or supported.

ILLVOL
An illegal volume was passed to the function.

SEE ALSO

AVW_MorphOpenImage(), AVW_ConditionalDilateVolume(), AVW_CreateStructuringVolume(), AVW_CreateVolume(), AVW_DilateVolume(), AVW_ErodeVolume(), AVW_MorphCloseVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume(), AVW_SetVolume(), AVW_PutVoxel(), AVW_Volume

NAME

AVW_MultiplyMatrix – multiplies two transformation matrices

SYNOPSIS

```
#include "AVW.h"

AVW_Matrix *AVW_MultiplyMatrix(mat_in1, mat_in2, mat_out)
AVW_Matrix *mat_in1, *mat_in2, *mat_out;
```

DESCRIPTION

AVW_MultiplyMatrix() multiplies the *AVW_Matrix*, *mat_in1*, times the *AVW_Matrix*, *mat_in2*, and return the result.

Out_matrix is provided as a method of reusing an existing *AVW_Matrix*. *(See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_MultiplyMatrix()* returns an *AVW_Matrix*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_Matrix, AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_InvertMatrix(), AVW_MirrorMatrix(), AVW_RotateMatrix(), AVW_SetIdentityMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis()

NAME

AVW_NearestNeighborDeconv – performs nearest-neighbor deconvolution on an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_NearestNeighborDeconv(in_image, slice_above,
                    slice_below, in_focus_filt,
                    out_focus_filt, cons, alpha,
                    out_image)

AVW_Image *in_image, *slice_above, *slice_below;
AVW_Image *in_focus_filt, *out_focus_filt;
double cons, alpha;
AVW_Image *out_image;
```

DESCRIPTION

AVW_NearestNeighborDeconv() performs nearest neighbor deconvolution for deblurring optical section data in one focal plane by using data from the slices above and below (see reference below). The slice in the focal plane is *in_image* and the image dimensions and are expected to be powers of 2. The slices above and below respectively are *slice_above* and *slice_below* and must be of the same dimensions. The in-focus transfer function *in_focus_filt* and the one slice out-of-focus transfer function *out_focus_filt* must both be of data type *AVW_FLOAT* or *AVW_COMPLEX* and of dimensions *xnum* /2 + 1, *ynum*, where xnum and ynum are the dimensions of the image. *Cons* is a constant between 0 and 1 which weights the out-of-focus contributions and is referred to as c in the reference. *Alpha* must be a positive float number and will be ignored and the value .0001 used in its place if it is less than or equal to zero.

The function allocates new memory for the returned *out_image* if the supplied *out_image* is incompatible in size or datatype.

More detailed information about nearest neighbor deconvolution may be found in the reference:

Agard, David A., *Fluorescence Microscopy in Three Dimensions* 1989, in Methods in Cell Biology, Vol. 30, Chap. 13, Academic Press.

RETURN VALUES

If successful *AVW_NearestNeighborDeconv()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_NearestNeighborDeconv() will fail if the following is true:

BADMAL
Malloc Failed. Unable to allocate memory for return volume or image.

ILLPAR
Illegal Parameter. An invalid value was given for an input parameter.

ILLIMG
Illegal Image. Input image was not a power of 2 in each dimension.

CFLSZ
Conflicting Size. The slices above or below did not match the input image.

BDSPCT
Bad Input Spectrum. An invalid spectrum was entered for a transfer function.

SEE ALSO *AVW_CreateStoksethMTF(), AVW_DeconvDivideImage(), AVW_DeconvWienerImage(), AVW_IterDeconvImage(), AVW_Image*

NAME

AVW_NearestNeighborPixel – finds value of nearest pixel

SYNOPSIS

```
#include "AVW.h"

double AVW_NearestNeighborPixel(image, point)
AVW_Image *image;
AVW_FPoint2 *point;
```

DESCRIPTION

AVW_NearestNeighborPixel() returns the value of the pixel nearest the floating point location *point* in the *image.* 0 is returned if the location is outside the *image.*

SEE ALSO

AVW_NearestNeighborVoxel(), AVW_GetPixel(), AVW_InterpolatedPixel(), AVW_CubicSplineInterpolatedPixel(), AVW_SincInterpolatedPixel(), AVW_FPoint2, AVW_Image

NAME

AVW_NearestNeighborVoxel – finds value of nearest voxel

SYNOPSIS

```
#include "AVW.h"

double AVW_NearestNeighborVoxel(volume, point)
AVW_Volume *volume;
AVW_FPoint3 *point;
```

DESCRIPTION

AVW_NearestNeighborVoxel() returns the value of the voxel nearest the floating point location *point* in the *volume.* 0 is returned if the location is outside the *volume.*

SEE ALSO

AVW_NearestNeighborPixel(), AVW_GetVoxel(), AVW_InterpolatedVoxel(), AVW_CubicSplineInterpolatedVoxel(), AVW_SincInterpolatedVoxel(), AVW_FPoint3, AVW_Volume

NAME

AVW_NonMaxImage – suppresses pixels which are not a local maximum

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_NonMaxImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_NonMaxImage() sets all the pixels in *in_image* which are not a maximum in the local neighborhood defined by the set pixels in *element* to zero.

Element is translated so that its centerpoint lies on every point of the image. At each point in the image, if the pixel is not the maximum of all the pixels that correspond to the ones in the translated structuring element it is set to zero. Otherwise it is left unchanged.

Element must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

AVW_CreateStructuringImage() or a combination of *AVW_CreateImage(), AVW_SetImage(),* and *AVW_PutPixel()* may be used to create a structuring element

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_NonMaxImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_NonMaxImage() will fail if:

ILLDT
Data type is not defined or supported or structuring element is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_NonMaxVolume(), AVW_CreateStructuringImage(), AVW_CreateImage(), AVW_MorphMaxImage(), AVW_MorphMinImage(), AVW_SetImage(), AVW_PutPixel(), AVW_Image

NAME

AVW_NonMaxVolume – suppresses voxels which are not a local maximum

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_NonMaxVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_NonMaxVolume() sets all the voxels in *in_volume* which are not a maximum in the local neighborhood defined by the set voxels in *element* to zero.

Element is translated so that its centerpoint lies on every point of the volume. At each point in the volume, if the voxel is not the maximum of all the voxels that correspond to the ones in the translated structuring element it is set to zero. Otherwise it is left unchanged.

Element must be of the data type *AVW_UNSIGNED_CHAR*. This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

AVW_CreateStructuringVolume() or a combination of *AVW_CreateVolume(), AVW_SetVolume(),* and *AVW_PutVoxel()* may be used to create a structuring element

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_NonMaxVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_NonMaxVolume() will fail if:

ILLDT
: Data type is not defined or supported or structuring element is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_NonMaxImage(), AVW_CreateStructuringVolume(), AVW_CreateVolume(), AVW_MorphMaxVolume(), AVW_MorphMinVolume(), AVW_SetVolume(), AVW_PutVoxel(), AVW_Volume

NAME

AVW_NormalizeHistogram – normalizes a histogram

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Histogram *AVW_NormalizeHistogram(histo, norm_histo)
AVW_Histogram *histo, *norm_histo;
```

DESCRIPTION

AVW_NormalizeHistogram() normalizes the *AVW_Histogram, histo,* and places the results in *norm_histo.* The number of pixels in each bin of the histogram is divided by the total number of all pixels in the histogram.

Norm_histo is provided as a method of reusing an existing *AVW_Histogram*. Reuse is possible only if the size of the provided *norm_histo* match the size of *histo.* In this case the pointer to *norm_histo* is returned by the function. If not reusable *norm_histo* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_NormalizeHistogram()* returns an *AVW_Histogram* containing the results of the function. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_NormalizeHistogram() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

ILLHIS
Histogram provided was not valid.

SEE ALSO

AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_DestroyHistogram(), AVW_FlattenImageHistogram(), AVW_FlattenVolumeHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_MatchImageHistogram(), AVW_MatchVolumeHistogram(), AVW_PreserveImageHistogram(), AVW_PreserveVolumeHistogram(), AVW_VerifyHistogram(), AVW_Histogram

NAME

AVW_ObjectScaleImage – create colored image using object map

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ObjectMap.h"

AVW_Image *AVW_ObjectScaleImage(in_image, in_max, in_min,
                obj_image, object_map,
                enhanced_flag, out_image)
AVW_Image *in_image;
double in_max, in_min;
AVW_Image *obj_image;
AVW_ObjectMap *object_map;
int enhance_flag;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ObjectScaleImage() takes an *AVW_Image*, *in_image*, and maximum and minimum intensity values, *in_max* and *in_min*, along with an image derived from an object map, the *object_map* itself, and produces a color version of the original image using a variety of enhancements controled by *enhance_flag.*

In_image is generally an image which AVW_GetOrthogonal() or *AVW_GetOblique(). Obj_image* must be the same size as *in_image*, and is usually extracted from the *object_map->Volume* using the same function and parameters as were used to extract *in_image.*

Enhance_flag specifies the method used to combine the input images to produce the desired colored image. Methods available are:

AVW_OBJECT_COLOR – The *in_max* and *in_min* parameters are used to "scale" the input values into the shades of colors available for each object.

AVW_ENHANCED_OBJECT_COLOR – The max and min values for each object within the image are determined, and then each object is scaled to it's own max / min before being divided into shades of color.

AVW_OBJECT_COLOR_ONLY – The intensity values in the *in_image* is ignored and the color is determined using only the object information contained in *obj_image.*

AVW_OBJECT_COLOR_EDGES – The edges of objects are enhanced with their color value.

AVW_OBJECT_EDGES_ONLY – The edges of objects are returned in their color value.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ObjectScaleImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO *AVW_IntensityScaleImage(), AVW_DestroyImage(), AVW_Image*

NAME

AVW_ObjectSeparator – separates connected objects

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ObjectSeparator(in_volume, seeds, omap, max, min)
AVW_Volume *in_volume;
AVW_PointList3 *seeds;
AVW_ObjectMap *omap;
double max, min;
```

DESCRIPTION

AVW_ObjectSeparator() is used to separate connected objects. *Max* and *Min*, along with the *DisplayFlags* for each object within *omap*, is used to segment the objects. *Seeds* determine which objects should not be connected, if they are connected, layers are continually peeled away and the connection rechecked until the objects have been separated. Once separated, the layers are regrown until initial values are again part of the separated objects. Each unique object is returned as a value within the returned volume. The returned volume can be used as input to *AVW_PutMultipleObjects* along with *omap* to redfine objects within an object map.

RETURN VALUES

If successful *AVW_ObjectSeparator()* returns an *AVW_Volume.* On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ObjectSeparator() will fail if one or more of the following is true:

BADMAL
: Unable to allocate sufficient memory.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_PutMultipleObjects(), AVW_PointList3, AVW_ObjectMap, AVW_Volume

NAME

AVW_OpenImageFile – opens a supported image file

SYNOPSIS

```
#include "AVW_ImageFile.h"

AVW_ImageFile *AVW_OpenImageFile(filename, modes)
char *filename;
char *modes;
```

DESCRIPTION

AVW_OpenImageFile() opens the named image file, determines the data format of the file, assigns *AVW* image IO functions related to this data format, calls the open() function appropriate for this data format, and associates the returned pointer to an *AVW_ImageFile* structure for that file. The pointer that is returned is then used as a handle for all subsequent image IO operations with this file. The *modes* parameter determines with what read modes the file is opened. Acceptable values are "r" for read only and "r+" for read and write update.

The *AVW_OpenImageFile()* function may not be used for the creation of new image files, only for opening existing files for reading or write modification. The *AVW_CreateImageFile()* function is used for the creation of new files.

RETURN VALUES

If successful *AVW_OpenImageFile()* returns an *AVW_ImageFile.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of failure. The returned *AVW_ImageFile* contains generic information about the size and type of images in the file without particular regard to the data format of the file. Access to data format specific attributes of the file are accessible through these elements in *AVW_ImageFile:*

NativeData is a (void *) pointer to the raw data for this file. This pointer allows the format specific routines in the AVW_ImageFile library to access the raw data.

DataFormat is an integer index into the currently supported list of *AVW* supported formats.

ERRORS

AVW_OpenImageFile() will fail if one or more of the following are true:

BDFRMT
: Unknown data format.

BDOPEN
: Bad Open. Error opening the file.

NOFILE
: File Doesn't Exist

NOPERMISSION
: File Permission Error

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_DisableImageFileFormat(), AVW_EnableIMageFileFormat(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume(), AVW_ImageFile

NAME

AVW_OpenImageFileList – opens a list of supported image files

SYNOPSIS

```
#include "AVW_ImageFile.h"

AVW_ImageFile *AVW_OpenImageFileList(filelist, modes)
AVW_List *filelist;
char *modes;
```

DESCRIPTION

AVW_OpenImageFileFile() attempts to open all the files named in *filelist*. If all the files can be successfully opened as image files and have the same width, height, and datatype the filelist functions as a virtual volume file.

The pointer that is returned is then used as a handle for all subsequent image IO operations with this file. The *modes* parameter determines with what read modes the file is opened. Acceptable values are "r" for read only and "r+" for read and write update.

The *AVW_OpenImageFileFiles()* function may not be used for the creation of new image files, only for opening existing files for reading or write modification. The *AVW_CreateImageFile()* function is used for the creation of new files.

RETURN VALUES

If successful *AVW_OpenImageFile()* returns an *AVW_ImageFile.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of failure. The returned *AVW_ImageFile* contains generic information about the size and type of images in the file without particular regard to the data format of the file. Access to data format specific attributes of the file are accessible through these elements in *AVW_ImageFile:*

NativeData is a (void *) pointer to the raw data for this file. This pointer allows the format specific routines in the AVW_ImageFile library to access the raw data.

DataFormat is an integer index into the currently supported list of *AVW* supported formats.

ERRORS

AVW_OpenImageFile() will fail if one or more of the following are true:

BDFRMT
: Unknown data format.

BDOPEN
: Bad Open. Error opening the file.

NOFILE
: File Doesn’t Exist

NOPERMISSION
: File Permission Error

SEE ALSO *AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_DisableImageFileFormat(), AVW_EnableIMageFileFormat(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFileList(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume(), AVW_ImageFile*

NAME

AVW_OrthoGradFilterImage – performs a 2D Orthogonal Gradient filter

SYNOPSIS

#include "AVW_Filter.h"

AVW_Image *AVW_OrthoGradFilterImage(in_image, out_image)
AVW_Image *in_image;
AVW_Image *out_image;

DESCRIPTION

AVW_OrthoGradFilterImage() replaces each pixel in *in_image* with the maximum absolute difference between the pixel’s orthogonal neighbors.

O(x,y) = max(abs(P(x,y-1)-P(x,y+1)),abs(P(x-1,y)-P(x+1,y)))

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer’s Guide.)*

RETURN VALUES

If successful *AVW_OrthoGradFilterImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_OrthoGradFilterImage will fail if the following is true:

ILLDT
Illegal Data Type

SEE ALSO

AVW_OrthoGradFilterVolume(), AVW_AHEImage(), AVW_LowpassFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterImageEnhance(), AVW_Image

NAME

AVW_OrthoGradFilterVolume – performs a 3D Orthogonal Gradient filter

SYNOPSIS

```
#include       "AVW_Filter.h"

AVW_Volume *AVW_OrthoGradFilterVolume(in_volume, out_volume)
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_OrthoGradFilterVolume() replaces each voxel in *in_volume* with the maximum absolute difference between the voxel's orthogonal neighbors.

O(x,y,z) = max(abs(P(x,y,z-1)-P(x,y,z+1)), abs(P(x,y-1,z)-P(x,y+1,z)), abs(P(x-1,y,z)-P(x+1,y,z)))

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_OrthoGradFilterVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_OrthoGradFilterImage will fail if the following is true:

ILLDT
: Illegal Data Type

SEE ALSO

AVW_OrthoGradFilterImage(), AVW_AHEVolume(), AVW_LowpassFilterVolume(), AVW_RankFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_UnsharpFilterVolumeEnhance(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_PCAImage – transforms a group of images with principal component analysis.

SYNOPSIS

```
#include "AVW.h"

int AVW_PCAImages(images, numImages, scaleFlag)
AVW_Image **images;
int numImages;
int scaleFlag
```

DESCRIPTION

AVW_PCAImages() transforms a list of spatially correlated images using principal component analysis. Principal component analysis is a preprocessing step used to reduce the number of bands required for multispectral classification. Images are transformed and resorted in order of information content.

images is an array of pointers AVW_Images.

numImages is the number of images being passed.

scaleFlag may have two acceptable values; AVW_TRUE or AVW_FALSE. When *scaleFlag* is true the dynamic range of the output images are rescaled to the range of the first image, otherwise not.

RETURN VALUES

If successful *AVW_PCAImages()* returns an *AVW_SUCCESS*. On failure *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_PCAImages() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_PCAVolumes()

NAME

AVW_PCAVolume – transforms a group of volumes using principal component analysis.

SYNOPSIS

```
#include       "AVW.h"

int AVW_PCAVolumes(volumes, numVolumes, scaleFlag)
AVW_Volume **volumes;
int numVolumes;
int scaleFlag
```

DESCRIPTION

AVW_PCAVolumes() transforms a list of spatially correlated volumes using principal component analysis. Principal component analysis is a preprocessing step used to reduce the number of bands required for multispectral classification. Volumes are transformed and resorted in order of information content.

volumes is an array of pointers AVW_Volumes.

numVolumes is the number of images being passed.

scaleFlag may have two acceptable values; AVW_TRUE or AVW_FALSE. When *scaleFlag* is true the ynamic range of the output volumes are rescaled to the range of the first volume, otherwise not.

RETURN VALUES

If successful *AVW_PCAVolumes()* returns an *AVW_SUCCESS*. On failure *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_PCAVolumes() will fail if one or more of the following are true:

BADMAL
    Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_PCAImages()

NAME

AVW_PadImage – pads an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_PadImage(in_image, out_width, out_height, location,
        pad_type, out_image)
AVW_Image *in_image;
int out_width, out_height;
AVW_Point2 *location;
int pad_type;
AVW_Image *out_image;
```

DESCRIPTION

AVW_PadImage() pads an image, *in_image,* with zeros, the values of the edge pixels, or the maximium or minimum value for the datatype. *In_image,* is placed in a larger *out_image* at the position specified by *location.*

Out_width and *out_height* specify the width and height of the returned image. *Location* specifies the position within the returned image where *in_image* will be placed. If *NULL,* the location is calculated as the middle of the out_width and out_height. *Pad_type* specifies whether the padded region in the returned image is set to zero, *AVW_PAD_ZERO,* the values of the border pixels in the *in_image, AVW_PAD_VALUE.* the maximum value for the datatype, *AVW_PAD_MAX,* or the minimum value for the datatype.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_PadImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PadImage() will fail if the following is true:

ILLDT
: Illegal data type.

SEE ALSO

AVW_PadVolume(), AVW_PutSubImage(), AVW_Image, AVW_Point2

NAME

AVW_PadVolume – pads a volume

SYNOPSIS

#include "AVW.h"

AVW_Volume ∗AVW_PadVolume(in_volume, out_width, out_height, out_depth,
    location, pad_type, out_volume)
AVW_Volume ∗in_volume;
int out_width, out_height, out_depth;
AVW_Point3 ∗location;
int pad_type;
AVW_Volume ∗out_volume;

DESCRIPTION

AVW_PadVolume() pads a volume, *in_volume,* with zeros or the values of the edge pixels. *In_volume,* is placed in a larger *out_volume* at the position specified by *location.* The padded region is set to zero or the values of the border pixels in *in_volume.*

Out_width, out_height and *out_depth* specify the width, height and depth of the returned volume. *Location* specifies the position within the returned volume where *in_volume* will be placed. *Pad_type* specifies whether the padded region in the returned volume is set to zero, *AVW_PAD_ZERO,* or the values of the border pixels in the *in_volume,* *AVW_PAD_VALUE.*

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_PadVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PadVolume() will fail if the following is true:

ILLDT
Illegal data type.

SEE ALSO

AVW_PadImage(), AVW_PutSubVolume(), AVW_Point3, AVW_Volume

NAME

AVW_Parse – evaluates an algebraic-style formula

SYNOPSIS

```
#include "AVW_Parse.h"

AVW_Instructions *AVW_Parse(formula)
char *formula;
```

DESCRIPTION

AVW_Parse() parses the string *formula* into a series of instruction which are returned in the *AVW_Instructions* structure.

A user interface will need to fill in variable information such as filename, starting slice, slice, incrment, and number to process.

The completed *AVW_Instructions* is passed to the *AVW_DoInstructions() function.*

RETURN VALUES

If successful, *AVW_Parse()* returns *AVW_SUCCESS*. On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_Parse() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

BADOP
Unknown Instruction.

SYNTAX
Syntax Error In Formula.

TCMPLX
Formula Too Complex To Be Parsed.

SEE ALSO

AVW_DoInstructions(), AVW_DestroyInstructions(), AVW_Instructions

NAME

AVW_PreserveImageHistogram – preserve an image intensity distribution

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Image *AVW_PreserveImageHistogram(in_image, max, min,
                fill_type, out_image)
AVW_Image *in_image;
double max, min;
int fill_type;
AVW_Image *out_image;
```

DESCRIPTION

AVW_PreserveImageHistogram() maps a broad grey scale range into a narrow range in a statistically optimal manner. For example, this function may be used to map 16384 gray level bins of a 16 bit image into 256 bins of an 8 bit image. The ideal distribution would place 1/256 of the total pixels in each of the 256 bins.

In_image specifies the input and *max* and *min* specify the output range.

Fill_type specifies how the bins for equalization are filled and must be:

AVW_OVERFILL **-** maps multiple input levels until output bin overfills,

AVW_UNDERFILL **-** maps multiple input levels until just before output bin overfills,

or

AVW_CLOSEST **-** chooses closest to ideal distribution.

The results are returned in *out_image.*

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_PreserveImageHistogram()* returns an *AVW_Image* which has been optimally mapped to new range. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PreserveImageHistogram() will fail if the following is true:

BADMAL
　Could not allocate memory for the results.

ILLIMG
　Image provided was not valid.

SEE ALSO

AVW_PreserveVolumeHistogram(), AVW_FlattenImageHistogram(), AVW_MatchImageHistogram(), AVW_GetImageHistogram(), AVW_Image

NAME

AVW_PreserveVolumeHistogram – preserves a volume intensity distribution

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Volume *AVW_PreserveVolumeHistogram(in_vol, max, min,
                                        fill_type, out_volume)
AVW_Volume *in_vol;
double max, min;
int fill_type;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_PreserveVolumeHistogram() maps a broad grey scale range into a narrow range in a statistically optimal manner. For example, this function may be used to map 16384 gray level bins of a 16 bit image into 256 bins of an 8 bit image. The ideal distribution would place 1/256 of the total pixels in each of the 256 bins.

In_vol specifies the input volume.

max and *min* specify the output range.

Fill_type specifies how the bins for equalization are filled and must be:

AVW_OVERFILL - maps multiple input levels until output bin overfills,

AVW_UNDERFILL - maps multiple input levels until just before output bin overfills,

or

AVW_CLOSEST - chooses closest to ideal distribution.

The results are returned in *out_volume.*

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_PreserveVolumeHistogram()* returns an *AVW_Volume* which has been optimally mapped to new range. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PreserveVolumeHistogram() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

ILLVOL
Volume provided was not valid.

SEE ALSO

AVW_FlattenVolumeHistogram(), AVW_MatchVolumeHistogram(), AVW_GetVolumeHistogram(), AVW_Volume

NAME

AVW_ProcessRenderWedge – Process subregion of Volume / Objectmap

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_ProcessRenderWedge(param, region, flag)
AVW_RenderParameters *param;
AVW_Rect3 *region;
int flag;
```

DESCRIPTION

AVW_ProcessRenderWedge() is used to preprocess the *param->Volume* or *param->Objectmap*. Copies should be made of the preprocessed volume if your wish to restore the volume to it's preprocessed condition.

Preprocessing of the input volumes, allows wedges or slabs to be removed. Only voxels which meet the current rendering requirements are removed. In other words, only voxels which are between the current threshold maximum and minimum values and those which are of an object type that is currently enabled will be processed.

Multiple pass preprocessing, by many calls to *AVW_ProcessRenderWedge()*, can further enhance renderings.

RETURN VALUES

If successful *AVW_ProcessRenderWedge()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ProcessRenderWedge() will fail if one or more of the following is true:

BADMAL
Couldn't allocate enough memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_RenderVolume(), AVW_CopyVolume(), AVW_RenderParameters, AVW_Rect3

NAME

AVW_ProcessZGradients – enhances a depth shaded rendering

SYNOPSIS

```
#include "AVW_Render.h"

AVW_Image *AVW_ProcessZGradients(rendered, coef, out_image)
AVW_RenderedImage *rendered;
double coef;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ProcessZGradients() enhances the *rendered->Image* by highlighting the changes in intensity values between neighboring pixels. This function should only be called to enhance images produced using the *AVW_DEPTH_SHADING* renderering algorithm.

Rendered is a pointer to a *AVW_RenderedImage* structure returned by a call to *AVW_RenderVolume().*

Coef is a floating point value which specifies the relationship between depth and gradient in calculating the brightness of the output pixel. The default value of .2, indicates that the calculation is weighted 20% towards the gradient calculated and 80% towards the initial voxel brightness.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ProcessZGradients()* returns an *AVW_Image.* On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ProcessZGradients() will fail if one or more of the following is true:

BADMAL
Couldn't allocate enough memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_RenderVolume(), AVW_RenderedImage(), AVW_Image

NAME

AVW_Progress – indicates current progress

SYNOPSIS

```
#include "AVW.h"

int AVW_Progress(percent)
int percent;
```

DESCRIPTION

AVW_Progress() is called by many *AVW* functions to indicate their current progress. *Percent* is a value from zero (0) to one hundred (100), indicating the current status of the calling procedure. This function will not normally be called unless *AVW* has been extended, and the programmer wishes to report the progress of the extended routine. Use of this function should be limited to functions which would normally require many seconds to execute. Functions which report progress, should not call other functions which report progress. Nested progresses are not supported.

When *AVW_Progress()* is called, the percent value is passed to a user defined function. This function is indicated to *AVW* by the *AVW_ProgressFunction()* call.

The return value is used to indicate a interrupt status. If non-zero value (1) is returned, the progress function indicates the process should continue. If zero (0) is returned, the progress function is requesting that the function be terminated.

Example:

```
int i, precent;

for(i=0; i<32768;++i)
        {
        percent = (i*100) / 32767;
        if(!AVW_Progress(percent))
                {
                /* clean up and return */
                return(ERROR_CODE);
                }
        /* continue processing */
        }
```

RETURN VALUES

If a call to *AVW_Progress()* returns zero, the function should be terminated. A non-zero return value indicates that processing should continue.

SEE ALSO

AVW_ProgressFunction(), AVW_Counter(), AVW_RenderVolume(), AVW_ReadVolume(), AVW_WriteVolume(), AVW_LoadObjectMap(), AVW_SaveObjectMap()

NAME AVW_ProgressFunction – indicates the function which reports progress

SYNOPSIS

```
#include "AVW.h"

void AVW_ProgressFunction(int (*function)())
```

DESCRIPTION Many *AVW* routines require many seconds to execute. The status of these routines is generally not reported until the function completes. To allow such things as progress bars to be generated while the function executes, many *AVW* functions call the *AVW_Progress()* function to indicate their status. When the *AVW_Progress()* function is called, the percentage value is passed on to a user specified function. The *AVW_ProgressFunction()* is used to indicate which user supplied function is to be called.

By returning zero (0), the user supplied routine can indicate a desire to interrupt the function. A non-zero value indicates the processing should continue.

Example:

```
{
int my_function();

AVW_ProgressFunction(my_function);

db = AVW_OpenImageFile("/path/to/a/image/file", "r");

volume = AVW_ReadVolume(db, 0, volume);

* Etc... */
}

int my_function(int percent)
{
DrawMyProgressBar(percent);

if(CanelKeyPressed) return(0);

return(1);
}
```

SEE ALSO *AVW_Progress(), AVW_RenderVolume(), AVW_ReadVolume(), AVW_WriteVolume(), AVW_LoadObjectMap(), AVW_SaveObjectMap()*

NAME

AVW_PruneVolume – removes branches of a skeleton

SYNOPSIS

```
#include "AVW.h"

int AVW_PruneVolume(in_volume, minlen)
AVW_Volume *in_volume;
int minlen;
```

DESCRIPTION

AVW_PruneVolume() removes the branches of a skeleton which are shorter than *minlen* voxels.

In_volume contains the 3D skeleton to be pruned which can be created by calling *AVW_Thin3D().*

Minlen specifies the minimum length of the branches in the skeleton. Branches with fewer than *minlen* voxels will be removed.

RETURN VALUES

If successful *AVW_PruneVolume()* returns the number of voxels removed by prunning. On failure it returns *0* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PruneVolume() will fail if:

ILLDT
Data type is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_Thin3D(), AVW_MakeTree(), AVW_Volume

NAME

AVW_PutHistoryInfo – updates the History field of an AVW information string

SYNOPSIS

```
#include "AVW.h"

char *AVW_PutHistoryInfo(string, info)
char *string;
char *info;
```

DESCRIPTION

AVW_PutHistoryInfo() is used to add a string value to the *History* field of an *AVW* information string. The default *match_string* is *History* (See *AVW_PutStringInfo())* and multiple entries may be stored in this field. A *string* value of "On" must be passed in order to enable the logging of *History* for subsequent calls to this routine. "Off" disables the storing of *History*.

String indicates what to set the *History* field to within *info.*

Info can be any zero terminated string. The *History* entry within the information string begins with *History* = and then the strings, seperated by colons, which have been added with this routine.

Info strings are used in AVW structures to carry optional additional information about the data that may not always be present and to allow the user to extend AVW structures to carry application dependant data.

The logging of history information must be done by the user as *AVW* routines do not currently perform this internally.

RETURN VALUES

If successful *AVW_PutHistoryInfo()* returns a pointer to a character string containing the updated information.

SEE ALSO

AVW_GetNumericInfo(), AVW_GetStringInfo(), AVW_PutNumericInfo(), AVW_PutStringInfo(), AVW_RemoveInfo(), AVW_Image, AVW_Volume

NAME

AVW_PutImageChannel – replaces a channel of an AVW_Color image

SYNOPSIS

#include "AVW.h"

```
int AVW_PutImageChannel(image, channel, rgb_image)
AVW_Image *image;
int channel;
AVW_Image *rgb_image;
```

DESCRIPTION

AVW_PutImageChannel() inserts an AVW_UNSIGNED_CHAR image into the specified channel of an AVW_COLOR image of the same size.

image must be of DataType AVW_UNSIGNBED char and the same Width and Height as *rgb_image.*

Channel can be specified as any of the following:

AVW_RED_CHANNEL - specifies the red channel.

AVW_GREEN_CHANNEL - specifies the green channel.

AVW_BLUE_CHANNEL - specifies the blue channel.

rgb_image
must be an existing AVW_COLOR image with the same width and height as *image.*

RETURN VALUES

If successful *AVW_PutImageChannel()* returns an *AVW_SUCCESS.* On failure it returns *AVW_FAIL.* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutImageChannel() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

ILLPAR
Illegal Parameter, images are wrong data type.

CFLZ
Conflicting size, images not same width and height.

SEE ALSO

AVW_GetVolumeChannel(), AVW_MakeColorImage(), AVW_DestroyImage() AVW_Image()

NAME

AVW_PutMaskedImage – copies an irregular area from one image to another

SYNOPSIS

```
#include "AVW.h"

int AVW_PutMaskedImage(from_image, to_image, mask)
AVW_Image *from_image;
AVW_Image *to_image;
AVW_Image *mask;
```

DESCRIPTION

For each non-zero pixel in *mask,* a the corresponding value is copied from *from_image* to *to_image.* Each zero pixel in *mask*, results in the corresponding pixel in *to_image* being unchanged.

RETURN VALUES

If successful *AVW_PutMaskedImage()* returns *AVW_SUCCESS.* On failure *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are to values corresponding to the cause of the failure.

ERRORS

AVW_PutMaskedImage() will fail if one or more of the following are true:

ILLDT
  Illegal Datatype.

CFLSZ
  Conflicting size or type between the image and volume.

SEE ALSO

AVW_GetMaskedImage(), AVW_Image

NAME

AVW_PutMaskedVolume – copies an irregular region from one volume to another

SYNOPSIS

```
#include "AVW.h"

int AVW_PutMaskedVolume(from_volume, to_volume, mask)
AVW_Volume *from_volume;
AVW_Volume *to_volume;
AVW_Volume *mask;
```

DESCRIPTION

For each non-zero voxel in *mask,* a the corresponding value is copied from *from_volume* to *to_volume.* Each zero voxel in *mask,* results in the corresponding voxel in *to_volume* being unchanged.

RETURN VALUES

If successful *AVW_PutMaskedVolume()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutMaskedVolume() will fail if one or more of the following are true:

ILLDT
Illegal Datatype.

CFLSZ
Conflicting size or type between the from_volume and to_volume.

SEE ALSO

AVW_GetMaskedVolume(), AVW_Volume

NAME

AVW_PutMultipleObjects – puts multiple objects into a object_map

SYNOPSIS

```
#include "AVW.h"

int AVW_PutMultipleObjects(volume, object_map)
AVW_Volume *volume;
AVW_ObjectMap *object_map;
```

DESCRIPTION

For each value within *volume*, a new object is created and defined in *object_map*.

RETURN VALUES

If successful *AVW_PutMultipleObjects()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutMultipleObjects() will fail if one or more of the following are true:

CFLSZ
Volume is not the same dimensions as the *object_map*.

ILLDT
Volume must have a *DataType* of *AVW_UNSIGNED_CHAR*.

ILLPAR
Illegal parameter. *object* value outside of range for the *objectmap*.

SEE ALSO

AVW_DefineConnected(), AVW_GetObject(), AVW_PutObject(), AVW_RenderVolume(), AVW_SaveObjectMap(), AVW_ObjectMap, AVW_Volume

NAME

AVW_PutNumericInfo – puts a numeric value in an AVW information string

SYNOPSIS

```
#include "AVW.h"

char *AVW_PutNumericInfo(match_string, numeric_value,info_string)
char *match_string, *info_string;
double numeric_value;
```

DESCRIPTION

AVW_PutNumericInfo() is used to add or update numeric values in an *AVW* information string.

Match_string is a zero terminated string which must match an entry in an information string exactly in order to get, update, or remove information.

Info_string can be any zero terminated string. Each entry within the information string begins with a tag followed by an equal sign (=) and then the value. A space character is used as a seperator between entries. String information is enclosed in double quotes (") to allow for spaces within strings.

Numeric_value indicates what to set *match_string* to within *info_string.*

Info strings are used in AVW structures to carry optional additional information about the data that may not always be present and to allow the user to extend AVW structures to carry application dependant data.

Examples:

```
img->Info = AVW_PutNumericInfo("VoxelWidth", 1.285, img->Info);
img->Info = AVW_PutNumericInfo("VoxelHeight", 1.285, img->Info);
img->Info = AVW_PutNumericInfo("VoxelDepth", 1.670, img->Info);
img->Info = AVW_PutNumericInfo("Slice", 17., img->Info);

printf("%s\n\n", img->Info);

xsize = AVW_GetNumericInfo("VoxelWidth", img->Info);
if(AVW_ErrorNumber == NOMTCH)
    fprintf(stderr, "VoxelWidth not found in info_string");
else
    printf("X Size=%f\n", xsize);

ysize = AVW_GetNumericInfo("VoxelHeight", img->Info);
if(AVW_ErrorNumber == NOMTCH)
    fprintf(stderr, "VoxelHeight not found in info_string");
else
    printf("Y Size=%f\n", ysize);

thick = AVW_GetNumericInfo("VoxelDepth", img->Info);
if(AVW_ErrorNumber == NOMTCH)
    fprintf(stderr, "VoxelDepth not found in info_string");
else
    printf("Thickness=%f\n", thick);

slice = AVW_GetNumericInfo("Slice", img->Info);
if(AVW_ErrorNumber == NOMTCH)
    fprintf(stderr, "Slice not found in info_string");
else
    printf("Slice=%d\n", (int)slice);
```

Results:

VoxelWidth=1.285 VoxelHeight=1.285 VoxelDepth=1.67 Slice=17

X Size=1.285
Y Size=1.285
Thickness=1.67
Slice=17

RETURN VALUES

If successful, *AVW_PutNumericInfo()* returns a pointer to a character string containing the updated information. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutNumericInfo() will fail if one or more of the following are true:

BADMAL
Unable to increase information string size.

NOMTCH
An illegal or NULL match_string was passed.

BDMTCH
Something is wrong with the info_string.

SEE ALSO

AVW_GetNumericInfo(), AVW_GetStringInfo(), AVW_PutHistoryInfo(), AVW_PutStringInfo(), AVW_RemoveInfo(), AVW_Image, AVW_ImageFile, AVW_Volume

NAME

AVW_PutObject – puts an object into a object_map

SYNOPSIS

#include "AVW.h"

```
int AVW_PutObject(volume, object_map, object)
AVW_Volume *volume;
AVW_ObjectMap *object_map;
int object;
```

DESCRIPTION

For each non-zero location in *volume*, the corresponding position within *object_map* is set to the *object* value.

RETURN VALUES

If successful *AVW_PutObject()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutObject() will fail if one or more of the following are true:

CFLSZ
Volume is not the same dimensions as the *object_map*.

ILLDT
Volume must have a *DataType* of *AVW_UNSIGNED_CHAR*.

ILLPAR
Illegal parameter. *object* value outside of range for the *objectmap*.

SEE ALSO

AVW_DefineConnected(), AVW_GetObject(), AVW_RenderVolume(), AVW_SaveObjectMap(), AVW_ObjectMap, AVW_Volume

NAME

AVW_PutOblique – puts an oblique image in a volume

SYNOPSIS

#include "AVW.h"

int AVW_PutOblique(image, volume, matrix)
AVW_Image *image;
AVW_Volume *volume;
AVW_Matrix *matrix;

DESCRIPTION

AVW_PutOblique() writes an image into a volume.

image is the image being written into the volume.

matrix defines the oblique orientation.

volume receives the image.

This routine can be used to store edited oblique images back into a volume.

RETURN VALUES

If successful *AVW_PutOblique()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutOblique() will fail if:

ILLVOL
Illegal volume.

SEE ALSO

AVW_GetOblique(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis(), AVW_MirrorMatrix(), AVW_PutOrthogonal(), AVW_RotateMatrix(), AVW_ScaleMatrix(), AVW_SetIdentityMatrix(), AVW_TranslateMatrix(), AVW_Image, AVW_Matrix, AVW_Volume

NAME

AVW_PutOrthogonal – puts an image in a volume

SYNOPSIS

```
#include     "AVW.h"

int AVW_PutOrthogonal(image, volume, orient, slice)
AVW_Image *image;
AVW_Volume *volume;
int      orient;
int      slice;
```

DESCRIPTION

AVW_PutOrthogonal() writes *image* into *volume* at the position specified by *orient* and *slice*. Acceptable values for *orient* are *AVW_TRANSVERSE*, *AVW_CORONAL*, and *AVW_SAGITTAL*. AVW numbers volume slices from 0 to (n-1). This means that acceptable slice values for transverse slices is 0 to (volume->Depth-1). Coronal slices number from 0 to (volume->Width-1) and sagittal slices number from 0 to (volume->Height-1).

This routine can be used to write edited or filtered images back into a volume.

RETURN VALUES

If successful *AVW_PutOrthogonal()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutOrthogonal() will fail if one or more of the following are true:

ILLVOL
Illegal volume.

ILLIMG
Illegal image.

CFLSZ
Conflicting size or type between the image and volume.

ILLPAR
Illegal parameter. Orient or slice is not valid.

SEE ALSO

AVW_GetOrthogonal(), AVW_GetOblique(), AVW_PutOblique(), AVW_Image, AVW_Volume

NAME

AVW_PutPixel – puts a value at a location in an image

SYNOPSIS

```
#include "AVW.h"

int AVW_PutPixel(image, point, value)
AVW_Image *image;
AVW_Point2 *point;
double value;
```

DESCRIPTION

AVW_PutPixel() sets the location in *image* specified by *point* to *value.*

For *AVW_COLOR* images the red, green, and blue components need to be packed into *value.* See *AVW_MAKERGB().*

RETURN VALUES

If successful *AVW_PutPixel()* returns *AVW_SUCCESS.*

On failure *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_PutPixel() will fail if one or more of the following are true:

ILLPAR
Point specified is outside of the image.

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_PutVoxel(), AVW_GetPixel(), AVW_GetVoxel(), AVW_DrawPointList2(), AVW_DrawPointList3(), AVW_Image, AVW_Point2, AVW_MAKERGB()

NAME

AVW_PutStringInfo – puts a string in an AVW information string

SYNOPSIS

```
#include "AVW.h"

char *AVW_PutStringInfo(match_string, string_value, info_string)
char *match_string, *string_value, *info_string;
```

DESCRIPTION

AVW_PutStringInfo() is used to add or update a text string value in an *AVW* information string.

Match_string is a zero terminated string which must match an entry in an information string exactly in order to get, update, or remove information.

Info_string can be any zero terminated string. Each entry within the information string begins with each tag followed by an equal sign (=) and then the value. A space character is used as a seperator between entries. String information is enclosed in double quotes (") to allow for spaces within strings.

String_value indicates what to set *match_string* to within *info_string.*

Info strings are used in AVW structures to carry optional additional information about the data that may not always be present and to allow the user to extend AVW structures to carry application dependant data.

Example:

```
img->Info = AVW_PutStringInfo("Name", "John Doe", img->Info);

printf("%s\n\n", img->Info);

if((name = AVW_GetStringInfo("Name", img->Info)) == NULL)
  fprintf("Name not found in info_string");
else
  printf("Name=%s\n", name);

free(name);
```

Results:

```
Name="John Doe"

Name=John Doe
```

RETURN VALUES

If successful, *AVW_PutStringInfo()* returns a pointer to a character string containing the updated information. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutStringInfo() will fail if one or more of the following are true:

NOMTCH
An illegal or NULL match_string was passed.

BDMTCH
Something is wrong with the info_string.

SEE ALSO

AVW_GetNumericInfo(), AVW_GetStringInfo(), AVW_PutHistoryInfo(), AVW_PutNumericInfo(), AVW_RemoveInfo(), AVW_Image, AVW_ImageFile, AVW_Volume

NAME

AVW_PutSubImage – puts a subregion into an image

SYNOPSIS

```
#include "AVW.h"

int AVW_PutSubImage(from_image, to_image, location)
AVW_Image *from_image, *to_image;
AVW_Point2 *location;
```

DESCRIPTION

AVW_PutSubImage() puts a smaller image, *from_image*, into a larger or equal sized image, *to_image*, at the position specified by *location*. The smaller image can be a subregion that was extracted from the larger image with *AVW_GetSubImage()*, and then processed in some way.

RETURN VALUES

If successful *AVW_PutSubImage()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutSubImage() will fail if one or more of the following are true:

ILLDT
Illegal datatype. From_image and to_image are not the same data type.

ILLPAR
Illegal parameter. From_image does not fit into to_image at the specified position.

SEE ALSO

AVW_PutSubVolume(), AVW_GetOrthogonal(), AVW_GetSubImage(), AVW_PutOrthogonal(), AVW_Image, AVW_Point2

NAME

AVW_PutSubVolume – puts a subvolume into a volume

SYNOPSIS

#include "AVW.h"

int AVW_PutSubVolume(from_volume, to_volume, frontupperleft)
AVW_Volume *from_volume, *to_volume;
AVW_Point3 *location;

DESCRIPTION

AVW_PutSubVolume() puts a smaller volume, *from_volume,* into a larger volume, *to_volume,* at the position specified by *location.* The smaller volume can be a subvolume that was extracted from the larger volume with *AVW_GetSubVolume()* and then processed in some way.

RETURN VALUES

If successful, *AVW_PutSubVolume()* returns *AVW_SUCCESS.* On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutSubVolume() will fail if one or more of the following are true:

ILLDT
Illegal datatype. From_volume and to_volume do not have the same data type.

ILLPAR
Illegal parameter. From_volume does not fit in to_volume at specified position.

SEE ALSO

AVW_PutSubImage(), AVW_GetSubVolume(), AVW_PadVolume(), AVW_Point3, AVW_Volume

NAME

AVW_PutImageChannel – replaces a channel of an AVW_Color volume

SYNOPSIS

#include "AVW.h"

int AVW_PutVolumeChannel(volume, channel, rgb_image)
AVW_Volume *volume;
int channel;
AVW_Volume *rgb_volume;

DESCRIPTION

AVW_PutVolumeChannel() inserts an AVW_UNSIGNED_CHAR volume into the specified channel of an existing AVW_COLOR volume of the same size.

Volume must be of DataType AVW_UNSIGNED char and the same Width and Height as *rgb_image.*

Channel can be specified as any of the following:

AVW_RED_CHANNEL **-** specifies the red channel.

AVW_GREEN_CHANNEL **-** specifies the green channel.

AVW_BLUE_CHANNEL **-** specifies the blue channel.

rgb_volume
must be an existing AVW_COLOR volume with the same width, height, and depth as *volume.*

RETURN VALUES

If successful *AVW_PutVolumeChannel()* returns an *AVW_SUCCESS.* On failure it returns *AVW_FAIL.* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_PutVolumeChannel() will fail if:

BADMAL
Malloc Failed. Unable to allocate sufficient memory.

ILLPAR
Illegal Parameter, volumes are wrong data type

CFLZ
Conflicting size, volumes not same width, height, and depth.

SEE ALSO

AVW_GetImageChannel(), AVW_MakeColorVolume(), AVW_DestroyImage() AVW_Image()

NAME

AVW_PutVoxel – puts a value at a location in a volume

SYNOPSIS

```
#include "AVW.h"

int AVW_PutVoxel(volume, point, value)
AVW_Volume *volume;
AVW_Point3 *point;
double value;
```

DESCRIPTION

AVW_PutVoxel() sets the location in *volume* specified by *point* to *value.*

For *AVW_COLOR* volumes the red, green, and blue components need to be packed into *value.* See *AVW_MAKERGB().*

RETURN VALUES

If successful *AVW_PutVoxel()* returns *AVW_SUCCESS. AVW_FAIL* is returned. On failure *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_PutVoxel() will fail if

ILLPAR
Point specified is outside of the volume.

ILLDT
Data type is not defined or supported.

SEE ALSO

AVW_PutPixel(), AVW_GetVoxel(), AVW_DrawPointList2(), AVW_DrawPointList3(), AVW_Point3, AVW_Volume, AVW_MAKERGB()

NAME

AVW_QuadSwapImage – swaps the quadrants of an image.

SYNOPSIS

```
#include "AVW.h"

int AVW_QuadSwapImage(in_image)
AVW_Image *in_image;
```

DESCRIPTION

AVW_QuadSwapImage() swaps the quadrants of *in_image* in place. The lower left quadrant becomes the upper right and vice versa, while the lower right quadrant becomes the upper left and vice versa. If an image is convolved with a point-spread function by multiplying the Fourier spectra of the image and PSF, and then performing an inverse FFT, the result will be quad-swapped. This routine undoes the swapping in place without requiring an intermediate image.

RETURN VALUES

If successful *AVW_QuadSwapImage()* returns *AVW_SUCCESS* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_QuadSwapImage() will fail if one or more of the following are true:

BADTYPE
Function does not support AVW_COMPLEX or AVW_COLOR datatypes.

ILLDT
Unknown or unsupported input or output displaytype.

SEE ALSO

AVW_MakeComplexImageViewable(), AVW_Image

NAME

AVW_QuickImageMaxMin – finds the maximum and minimum values of an image

SYNOPSIS

```
#include "AVW.h"

int AVW_QuickImageMaxMin(image, max_val, min_val)
AVW_Image *image;
double *max_val, *min_val;
```

DESCRIPTION

AVW_QuickImageMaxMin() finds the maximum and minimum data values in *image* and returns them in *max_val* and *min_val.*

AVW_QuickImageMaxMin() checks the *image->Info* information string for *MaximumDataValue* and *MinimumDataValue.* If found, the *max_val* and *min_val* values are set using the information string values. If not found, the *max_val* and *min_val* values are calculated by calling *AVW_FindImageMaxMin().*

After calculation, the values are stored in the information string to allow quicker future max / min queries.

RETURN VALUES

If successful, *AVW_QuickImageMaxMin()* returns *AVW_SUCCESS.* On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_QuickImageMaxMin() will fail if:

NOTSUP
Image data type not supported.

SEE ALSO

AVW_GetNumericInfo(), AVW_FindImageMaxMin(), AVW_QuickVolumeMaxMin(), AVW_Image

NAME

AVW_QuickMatch – generates the matrix required to register volumes

SYNOPSIS

#include "AVW_MatchVoxels.h"

AVW_Matrix *AVW_QuickMatch(base_volume, match_volume, BaseMax, BaseMin, MatchMax,Matc
AVW_Volume *base_volume, *match_volume;
double BaseMin, BaseMax,MatchMin,MatchMax;
AVW_Matrix *matrix;

DESCRIPTION

AVW_QuickMatch() determines the geometric transformation parameters required to spatially register the *base_volume* and *match_volume*

matrix is provided as a method of re-using an existing *AVW_Matrix.*

RETURN VALUES

If successful returns a pointer to an *AVW_Matrix* structure which contains an homogeneous 4x4 matrix used to register *match_volume* to *base_volume.*

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_QuickMatch() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

SEE ALSO

AVW_TransformVolume(), AVW_Matrix, AVW_Volume.

NAME

AVW_QuickVolumeMaxMin – finds the maximum and minimum values of an volume

SYNOPSIS

```
#include "AVW.h"

int AVW_QuickVolumeMaxMin(volume, max_val, min_val)
AVW_Volume *volume;
double *max_val, *min_val;
```

DESCRIPTION

AVW_QuickVolumeMaxMin() finds the maximum and minimum data values in *volume* and returns them in *max_val* and *min_val.*

AVW_QuickVolumeMaxMin() checks the *volume->Info* information string for *MaximumDataValue* and *MinimumDataValue.* If found, the *max_val* and *min_val* values are set using the information string values. If not found, the *max_val* and *min_val* values are calculated by calling *AVW_FindVolumeMaxMin().*

After calculation, the values are stored in the information string to allow quicker future max / min queries.

RETURN VALUES

If successful, *AVW_QuickVolumeMaxMin()* returns *AVW_SUCCESS.* On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_QuickVolumeMaxMin() will fail if:

NOTSUP
Volume data type not supported.

SEE ALSO

AVW_GetNumericInfo(), AVW_FindVolumeMaxMin(), AVW_QuickImageMaxMin(), AVW_Volume

NAME

AVW_RankFilterImage – performs a 2D generic rank filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_RankFilterImage(in_image, extents, rank, out_image)
AVW_Image *in_image;
int extents[2];
int rank;
AVW_Image *out_image;
```

DESCRIPTION

AVW_RankFilterImage() performs a generic rank filter transformation on *in_image. Extents[0]*, and *extents[1]*, specify the x and y sizes respectively of the filter. Extent values should be odd.

A rank filter orders all the values found in the filter region and returns the value of the *rank'th* pixel. For a 3X3 region, a rank of 5 would be the median value. *Rank* of value 1 is the lowest value. The maximum value for *rank* is *extents[0]*extents[1].* A *rank* of value 0 effects a median filter.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RankFilterImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_RankFilterVolume(), AVW_AHEImage(), AVW_LowpassFilterImage(), AVW_OrthoGradFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_SobelFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_Image

NAME

AVW_RankFilterVolume – performs a 3D generic rank filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_RankFilterVolume(in_volume, extents, rank, out_volume)
AVW_Volume *in_volume;
int extents[3];
int rank;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_RankFilterVolume() performs a generic rank filter transformation on *in_volume*. *Extents[0]*, *extents[1]*, and *extents[2]* specify the x, y, and z sizes respectively of the filter. Extent values should be odd.

A rank filter orders all the values found in the filter region and returns the value of the *rank'th* voxel. For a 3X3X3 region, a rank of 14 would be the median value. *Rank* of value 1 is the lowest value. The maximum value for *rank* is *extents[0]*extents[1]*extents[2]*. A *rank* of value 0 effects a median filter.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RankFilterVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_RankFilterImage(), AVW_AHEVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_SobelFilterVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_ReadHistogram – reads a histogram file

SYNOPSIS

```
#include "AVW_Histogram.h"

AVW_Histogram *AVW_ReadHistogram(filename, histo)
char *filename;
AVW_Histogram *histo;
```

DESCRIPTION

AVW_ReadHistogram() reads a histogram from the specified file. If the passed *histo* parameter is *NULL*, *AVW_ReadHistogram()* will allocate a histogram of the appropriate size as indicated in the histogram file. If the passed *histo* is not compatible with the requirements of the histogram file, the *histo* is destroyed and a new *histo* will be allocated.

RETURN VALUES

Upon success *AVW_ReadHistogram()* returns an *AVW_Histogram.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

Errors may occur for the following reasons:

BDOPEN
Bad Open. Failure opening file.

ILLHIS
Illegal Histogram. Could not create histogram.

SEE ALSO

AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_DestroyHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_NormalizeHistogram(), AVW_VerifyHistogram(), AVW_WriteHistogram(),

NAME

AVW_ReadImageFile – reads an image from an image file

SYNOPSIS

```
#include    "AVW_ImageFile.h"

AVW_Image *AVW_ReadImageFile (imgfile, image)
AVW_ImageFile *imgfile;
AVW_Image    *image;
```

DESCRIPTION

AVW_ReadImageFile() reads the next image from the *AVW_ImageFile.* At the conclusion of the read the file is positioned to read the following image. If the passed *image* parameter is *NULL*, *AVW_ReadImageFile()* will allocate an image of the same size and type as indicated in the *imgfile.* If the passed *image* is not compatible with the requirements of *imgfile*, the *image* is destroyed and a new *image* is allocated.

RETURN VALUES

Upon success *AVW_ReadImageFile()* returns an *AVW_Image.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

Errors may occur for the following reasons:

BADMAL
Bad Malloc. Memory allocation error.

BDREAD
Read error.

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendExternalLibs(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadVolume(), AVW_SeekImageFile() AVW_WriteImageFile(), AVW_WriteVolume(), AVW_ImageFile, AVW_Image

NAME

AVW_ReadVolume – reads a volume from an image file

SYNOPSIS

```
#include    "AVW_ImageFile.h"

AVW_Volume *AVW_ReadVolume (imgfile, volnum, vol)
AVW_ImageFile *imgfile;
int          volnum;
AVW_Volume    *vol;
```

DESCRIPTION

AVW_ReadVolume() reads the specified volume from the *AVW_ImageFile.* If the passed *vol* parameter is *NULL, AVW_ReadVolume()* will allocate a volume of the same size and type as indicated in the imgfile. If the passed *vol* is not compatible with the requirements of *imgfile,* the *vol* is destroyed and a new *vol* will be allocated.

Volnum is the volume number to be read. Volumes are numbered from 0 to NumVols-1. Thus if the first volume from a file is to be read, specify 0.

For image files whose data format does not support 3D or 4D the single image will be returned as a volume with Depth equal to 1.

RETURN VALUES

Upon success *AVW_ReadVolume()* returns an *AVW_Volume.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

Errors may occur for the following reasons:

BADMAL
: Bad Malloc. Memory allocation error.

BDREAD
: Read error.

BDVLNM
: Bad volume number.

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendExternalLibs(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteSubVolumeDescription(), AVW_WriteVolume(), AVW_ImageFile, AVW_Volume

NAME

AVW_Malloc – allocates system memory

SYNOPSIS

```
#include "AVW.h"

void *AVW_Malloc(size)
unsigned int size;

void *AVW_Calloc(num, size)
unsigned int num;
unsigned int size;

void *AVW_Realloc(size, ptr)
unsigned int size;
void *ptr;

void AVW_Free(ptr)
void *ptr;
```

DESCRIPTION

These procedures provide a platform and compiler independent interface for memory allocation. Programs that need to transfer ownership of memory blocks between *AVW* and other modules should use these routines rather than the native *malloc()* and *free()* routines provided by the *C* run-time library.

AVW_Malloc returns a pointer to a size bytes suitably aligned for any use.

AVW_Calloc allocates space for an array nelem elements of *size* elsize. The space is initialized to zeros.

Tcl_Realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the new block. The contents will be unchanged up to the lesser of the new and old sizes. The returned location may be different from *ptr*.

Tcl_Free makes the space referred to by *ptr* available for further allocation.

SEE ALSO

malloc(), free()

NAME

AVW_ReassignObject – deletes an object from an object map

SYNOPSIS

```
#include "AVW_ObjectMap.h"

int AVW_ReassignObject(object_map, from, to)
AVW_ObjectMap *object_map;
int from;
int to;
```

DESCRIPTION

AVW_ReassignObject() changes all the voxels currently assigned to the *from* object, so that they are assigned to the *to* object.

RETURN VALUES

If successful *AVW_ReassignObject()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ReassignObject() will fail if the following is true:

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_DestroyObjectMap(), AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_GetObject(), AVW_PutObject() AVW_LoadObjectMap(), AVW_SaveObjectMap(), AVW_RemoveUnusedObjects(), AVW_RenderVolume(), AVW_ObjectMap, AVW_Object

NAME

AVW_ReduceColors – reduces colors used in an image

SYNOPSIS

```
#include    "AVW.h"

int AVW_ReduceColors(image, method, ncolors)
AVW_Image *image;
int method;
int ncolors;
```

DESCRIPTION

AVW_ReduceColors() reduces the number of colors used in an *AVW_Image* by changing the *image* and *image->Colormap.*

Method may be one of or a combination of *AVW_DUPLICATE_COLORMAP_ENTRIES*, *AVW_UNUSED_COLORMAP_ENTRIES*, and *AVW_LEAST_USED_COLOR*

> When set to *AVW_DUPLICATE_COLORMAP_ENTRIES*, the colormap is searched for duplicate entries and if duplicates are found they are removed.
>
> When set to *AVW_UNUSED_COLORMAP_ENTRIES*, the colormap is searched for unused entries and if found they are removed.
>
> When set to *AVW_LEAST_USED_COLOR*, the colormap is searched and the least used entry is removed, this process is repeated until the total colors is reduced to ncolors or less.

RETURN VALUES

AVW_ReduceColors() returns the number of colormap entries that were removed. 0 will be returned if *image* did not have an associated colormap or the data type was not *AVW_UNSIGNED_CHAR.*

SEE ALSO

AVW_CopyColormap(), AVW_CreateColormap(), AVW_DestroyColormap(), AVW_IsGrayColormap(), AVW_LoadColormap(), AVW_SaveColormap(), AVW_Colormap, AVW_Image

NAME

AVW_RegenerateObjectMap – loads an object map

SYNOPSIS

```
#include "AVW_ObjectMap.h"

AVW_ObjectMap *AVW_RegenerateObjectMap(omap, vol)
AVW_ObjectMap *omap;
AVW_Volume *vol;
```

DESCRIPTION

AVW_RegenerateObjectMap() regenerates a new volume and object map by applying the transformations specified for each object.

RETURN VALUES

If successful *AVW_RegenerateObjectMap()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RegenerateObjectMap() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_LoadObjectMap(), AVW_RenderVolume(), AVW_SaveObjectMap(), AVW_ObjectMap

NAME

AVW_RegisterProgramName – initialize AVW

SYNOPSIS

```
#include "AVW.h"

int AVW_RegisterProgramName(name)
char *name;
```

DESCRIPTION

AVW_RegisterProgramName() should be the first *AVW* function called. Results can be checked to determine if a license is available. *Name* specifies a unique name for program. *Name* is no longer used during the validation process.

RETURN VALUES

AVW_RegisterProgramName() returns *AVW_SUCCESS* for valid users and systems. *AVW_FAIL* is returned for invalid systems and / or users and a descriptive message is written to stderr and / or stdout.

NAME

AVW_RemoveFPoint2 – removes a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_RemoveFPoint2(trace, which_point)
AVW_FPointList2 *trace;
int which_point;
```

DESCRIPTION

AVW_RemoveFPoint2() removes an AVW_FPoint2 from an AVW_FPointList2.

Trace is an *AVW_FPointList2.*

Which_point specifies the point number of the trace that is to be removed. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_RemoveFPoint2 will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_AddFPoint2(), AVW_CopyFPointList2(), AVW_CreateFPointList2(), AVW_DestroyFPointList2(), AVW_GetFPoint2(), AVW_MakeFPointList2(), AVW_RemoveFPoint3(), AVW_RemoveIPoint2(), AVW_RemoveIPoint3(), AVW_RemovePoint2(), AVW_RemovePoint3(), AVW_RemovePointValue(), AVW_TransformFPoint2(), AVW_FPoint2, AVW_FPointList2

NAME

AVW_RemoveFPoint3 – removes a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_RemoveFPoint3(trace, which_point)
AVW_FPointList3 *trace;
int which_point;
```

DESCRIPTION

AVW_RemoveFPoint3() removes an AVW_FPoint3 from an AVW_FPointList3.

Trace is an *AVW_FPointList3.*

Which_point specifies the point number of the trace that is to be removed. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_RemoveFPoint3 will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_AddFPoint3(), AVW_CopyFPointList3(), AVW_CreateFPointList3(), AVW_DestroyFPointList3(), AVW_GetFPoint3(), AVW_RemoveFPoint2(), AVW_RemoveIPoint2(), AVW_RemoveIPoint3(), AVW_RemovePoint2(), AVW_RemovePoint3(), AVW_RemovePointValue(), AVW_TransformFPoint3(), AVW_FPoint3, AVW_FPointList3

NAME

AVW_RemoveIPoint2 – removes a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_RemoveIPoint2(trace, which_point)
AVW_IPointList2 *trace;
int which_point;
```

DESCRIPTION

AVW_RemoveIPoint2() removes an AVW_IPoint2 from an AVW_IPointList2.

Trace is an *AVW_IPointList2.*

Which_point specifies the point number of the trace that is to be removed. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_RemoveIPoint2 will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_AddIPoint2(), AVW_CopyIPointList2(), AVW_CreateIPointList2(), AVW_DestroyIPointList2(), AVW_GetIPoint2(), AVW_RemoveFPoint2(), AVW_RemoveFPoint3(), AVW_RemoveIPoint3(), AVW_RemovePoint2(), AVW_RemovePoint3(), AVW_RemovePointValue(), AVW_TransformIPoint2(), AVW_IPoint2, AVW_IPointList2

NAME

AVW_RemoveIPoint3 – removes a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_RemoveIPoint3(trace, which_point)
AVW_IPointList3 *trace;
int which_point;
```

DESCRIPTION

AVW_RemoveIPoint3() removes an *AVW_IPoint3* from an *AVW_IPointList3.*

Trace is an *AVW_IPointList3.*

Which_point specifies the point number of the trace that is to be removed. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_RemoveIPoint3 will fail if the following is true:

ILLPAR
which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_AddIPoint3(), AVW_CopyIPoint3list(), AVW_CreateIPointList3(), AVW_DestroyIPointList3(), AVW_GetIPoint3(), AVW_RemoveFPoint2(), AVW_RemoveFPoint3(), AVW_RemoveIPoint2(), AVW_RemovePoint2(), AVW_RemovePoint3(), AVW_RemovePointValue(), AVW_TransformIpoint3(), AVW_IPoint3, AVW_IPointList3

NAME

AVW_RemoveInfo – removes a value from an information string

SYNOPSIS

#include "AVW.h"

char * AVW_RemoveInfo(match_string, info_string)
char *match_string, *info_string;

DESCRIPTION

AVW_RemoveInfo() removes an entry from an information string.

Match_string is a zero terminated string which must match an entry in an information string exactly in order to get, update, or remove information.

Info_string can be any zero terminated string. Each entry within the information string begins with a tag followed by an equal sign (=) and then the value. A space character is used as a seperator between entries. String information is enclosed in double quotes (") to allow for spaces within strings.

Info strings are used in AVW structures to carry optional additional information about the data that may not always be present and to allow the user to extend AVW structures to carry application dependant data.

RETURN VALUES

If successful, *AVW_RemoveInfo()* returns a pointer to a character string with the *match_string* removed. On failure it returns the unchanged *info_string* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RemoveInfo() will fail if the following is true:

NOMTCH
No match string was found in the info_string.

SEE ALSO

AVW_GetNumericInfo(), AVW_GetStringInfo(), AVW_ListInfo(), AVW_MergeInfo(), AVW_PutHistoryInfo(), AVW_PutNumericInfo(), AVW_PutStringInfo(), AVW_Image, AVW_Volume

NAME

AVW_RemovePoint2 – removes a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_RemovePoint2(trace, which_point)
AVW_PointList2 *trace;
int which_point;
```

DESCRIPTION

AVW_RemovePoint2() removes an AVW_Point2 from an AVW_PointList2.

Trace is an *AVW_PointList2.*

Which_point specifies the point number of the trace that is to be removed. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_RemovePoint2 will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_AddPoint2(), AVW_ClipPointList2(), AVW_ClosestInPointList2(), AVW_CopyPointList2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_DrawPointList2(), AVW_EditPointList2(), AVW_FillPointList2(), AVW_GetPoint2(), AVW_RemoveFPoint3(), AVW_RemoveFPoint2(), AVW_RemoveIPoint3(), AVW_RemoveIPoint2(), AVW_RemovePoint3(), AVW_RemovePointValue(), AVW_ScalePointList2(), AVW_ShiftPointList2(), AVW_TransformPoint2(), AVW_TranslatePointlist2(), AVW_Point2, AVW_PointList2

NAME

AVW_RemovePoint3 – removes a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_RemovePoint3(trace, which_point)
AVW_PointList3 *trace;
int which_point;
```

DESCRIPTION

AVW_RemovePoint3() removes an AVW_Point3 from an AVW_PointList3.

Trace is an *AVW_PointList3.*

Which_point specifies the point number of the trace that is to be removed. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_RemovePoint3 will fail if the following is true:

ILLPAR
: *which_value* is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_AddPoint3(), AVW_CopyPointList3(), AVW_CreatePointList3(), AVW_DestroyPointList3(), AVW_DrawPointList3(), AVW_FillPointList3(), AVW_GetPoint3(), AVW_RemoveFPoint2(), AVW_RemoveFPoint3(), AVW_RemoveIPoint2(), AVW_RemoveIPoint3(), AVW_RemovePoint2(), AVW_RemovePointValue(), AVW_TransformPoint3(), AVW_Point3, AVW_PointList3

NAME

AVW_RemovePointValue – removes a point from a list

SYNOPSIS

```
#include "AVW.h"

int AVW_RemovePointValue(trace, which_point)
AVW_PointValueList *trace;
int which_point;
```

DESCRIPTION

AVW_RemovePointValue() removes an AVW_PointValue from an AVW_PointValueList.

Trace is an *AVW_PointValueList.*

Which_point specifies the point number of the trace that is to be removed. Acceptable values range from 0 (the first point in the list) to trace->NumberOfPoints-1 (the last point).

RETURN VALUES

If succesful *AVW_SUCCESS* is returned, otherwise *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_RemovePointValue will fail if the following is true:

ILLPAR

which_value is less than 0 or greater or equal to trace->NumberOfPoints.

SEE ALSO

AVW_AddPointValue(), AVW_CopyPointValueList(), AVW_CreatePointValueList(), AVW_DestroyPointValueList(), AVW_GetPointValue(), AVW_RemoveFPoint2(), AVW_RemoveFPoint3(), AVW_RemoveIPoint2(), AVW_RemoveIPoint3(), AVW_RemovePoint2(), AVW_RemovePoint3(), AVW_PointValue, AVW_PointValueList

NAME AVW_RemoveUnrenderable – unsets unrenderable pixels

SYNOPSIS

```
#include "AVW.h"

int AVW_RemoveUnrenderable(image, threshmax, threshmin, oimage, omap)
AVW_Image *image;
double threshmax, threshmin;
AVW_Image *oimage;
AVW_ObjectMap *omap;
```

DESCRIPTION

AVW_RemoveUnrenderable() processes the *AVW_Image image,* removing all those pixels which do not meet the current rendering requirements.

Image is an *AVW_Image* extracted directly from a volume, most likely from *AVW_GetOrthogonal(), AVW_GetOblique(),* or *AVW_GetCurved.*

Threshmax and *threshmin* indicate which range of pixels to keep.

Oimage is an optional *AVW_Image.* It should have been extracted directly from the *Volume* member of an *AVW_Objectmap,* using the same function as was used to extract *image* above. If this parameter is non-NULL, all voxels in *image* which are members of disabled objects, will be removed.

Omap is an AVW_Objectmap which must be provided if *oimage* is non-NULL.

RETURN VALUES

If successful *AVW_RemoveUnrenderable()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RemoveUnrenderable() will fail if:

ILLPAR
Illegal parameter.

ILLDT
Illegal data type.

SEE ALSO

AVW_GetOrthogonal(), AVW_GetOblique(), AVW_GetCurved(), AVW_Image, AVW_ObjectMap

NAME

AVW_RemoveUnusedObjects – deletes unused object(s) from an object map

SYNOPSIS

```
#include "AVW_ObjectMap.h"

int AVW_RemoveUnusedObjects(object_map)
AVW_ObjectMap *object_map;
```

DESCRIPTION

AVW_RemoveUnusedObjects() removes all the unused objects within *object_map*. An unused object can be any object from 0 to *object_map->NumberOfObjects* which does not have at least one corresponding voxel in *object_map->Volume;*

RETURN VALUES

If successful *AVW_RemoveUnusedObjects()* returns the number of objects removed. On failure it returns -1 and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RemoveUnusedObjects() will fail if the following is true:

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_CreateObjectMap(), AVW_GetObject(), AVW_PutObject() AVW_LoadObjectMap(), AVW_SaveObjectMap(), AVW_RenderVolume(), AVW_ObjectMap, AVW_Object

NAME

AVW_RenderGradients – quick render from gradient information

SYNOPSIS

```
#include "AVW_Render.h"

AVW_RenderedImage *AVW_RenderGradients(gradients, param, last_rendered)
AVW_Gradients *gradients;
AVW_RenderParameters *param;
AVW_RenderedImage *last_rendered;
```

DESCRIPTION

AVW_RenderGradients() returns an *AVW_RenderedImage* from *gradients* and *param.* Only a very limited portion of the *param* structure is utilitized. *Param->RenderType* is required to be *AVW_GRADIENT_SHADING,* and no *param->RenderMask* may be specified. The only parameters used from the *param* structure are: *RenderWidth, RenderHeight, RenderDepth,* and most importantly, *Matrix.*

This function was designed to return very interactive renderings provided that the limited parameter set used is adequate.

RETURN VALUES

If successful *AVW_RenderGradients()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RenderGradients() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_DestroyGradients(), AVW_ExtractGradients(), AVW_ProcessZGradients(), AVW_RenderOblique(), AVW_RenderVisibleSurface(), AVW_RenderVolume(), AVW_RenderedImage, AVW_RenderedParameters, AVW_Gradients

NAME

AVW_RenderOblique – renders an oblique image

SYNOPSIS

```
#include "AVW_Render.h"

AVW_RenderedImage *AVW_RenderOblique(volume, ow, oh, omat, matrix, interpolate_flag, shadin
AVW_Volume *volume;
int ow, oh;
AVW_Matrix *omat;
AVW_Matrix *matrix;
int interpolate_flag;
double shading_fraction;
AVW_RenderedImage *last_rendered;
```

DESCRIPTION

AVW_RenderOblique() returns an *AVW_RenderedImage* showing any plane from any view point.

Volume specifies the *AVW_Volume* the plane is extracted from.

Ow and *oh* specify the dimension of the extracted plane.

Omat is an *AVW_Matrix* which describes a plane. See *AVW_GetOblique()* for more information.

Matrix is used to specify any rotation or scale factors.

The *interpolate_flag* specifies if tri-linear interpolations should be used when generating the image. Setting the *interpolate_flag* to *AVW_FALSE* causes *Nearest Neighbor* to be used, which is much faster, but lacks some of the quality.

Last_rendered is provided as a method of reusing an existing *AVW_RenderedImage.* Reuse is possible only if the size and data type of the provided *last_rendered* meet the requirements of the function. In this case the pointer to *last_rendered* is returned by the function. If not reusable *last_rendered* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RenderOblique()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RenderOblique() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_CubeSections(), AVW_IntersectingSections(), AVW_DestroyRenderedImage(), AVW_MergeRendered(), AVW_RenderedImage

NAME

AVW_RenderSections – renders intersecting sections

SYNOPSIS

```
#include "AVW_Render.h"

AVW_RenderedImage *AVW_RenderSections(volume, list, mat, interpolate, shading_fraction, render
AVW_Volume *volume;                          /* description of volume */
short *list;
AVW_Matrix *mat;
int interpolate;
double shading_fraction;
AVW_RenderedImage *rendered;
```

DESCRIPTION

AVW_RenderSections() returns an *AVW_RenderedImage* showing one or more transverse, coronal and sagittal sections.

Volume specifies the *AVW_Volume* the sections are extracted from.

List is a zero terminated array of orientation and slice pairs.

Mat is used to specify any rotation or scale factors.

The *interpolate_flag* specifies if tri-linear interpolations should be used when generating the image. Setting the *interpolate_flag* to *AVW_FALSE* causes *Nearest Neighbor* to be used, which is much faster, but lacks some of the quality.

Last_rendered is provided as a method of reusing an existing *AVW_RenderedImage.* Reuse is possible only if the size and data type of the provided *last_rendered* meet the requirements of the function. In this case the pointer to *last_rendered* is returned by the function. If not reusable *last_rendered* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RenderSections()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RenderSections() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_IntersectingSections(), AVW_CubeSections(), AVW_RenderOblique(), AVW_DestroyRenderedImage(), AVW_RenderedImage

NAME

AVW_RenderVisibleSurface – renders a visible surface

SYNOPSIS

```
#include "AVW_Render.h"

int AVW_RenderVisibleSurface(surface, param, last_rendered)
AVW_VisibleSurface *surface;
AVW_RenderParameters *param;
AVW_RenderedImage *last_rendered;
```

DESCRIPTION

AVW_RenderVisibleSurface() quickly re-renders the *surface* points at new locations as specified by *param.* The results are placed in *last_rendered,* which is the structure which resulted in a previous call to *AVW_RenderVolume()* or *AVW_RenderVisibleSurface().*

Only the *Matrix* portion of the *AVW_RenderParameters* structure will have a significant effect on the output.

This function was designed to return very interactive renderings provided that the limited parameter set used is adequate.

RETURN VALUES

If successful *AVW_RenderVisibleSurface()* returns an *AVW_RenderedImage.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RenderVisibleSurface() will fail if:

BADMAL
: Malloc Failed. Unable to increase structure size.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_DestroyVisibleSurface(), AVW_ExtractVisibleSurface(), AVW_FindSurfaceArea(), AVW_FindSurfaceDistance(), AVW_FindSurfacePoints(), AVW_RenderGradients(), AVW_RenderOblique(), AVW_RenderVolume(), AVW_RenderedImage, AVW_RenderedParameters, AVW_VisibleSurface

NAME AVW_RenderVolume – renders a volume by ray casting

SYNOPSIS

```
#include "AVW_Render.h"

AVW_RenderedImage *AVW_RenderVolume(param, last_rendered)
AVW_RenderParameters *param;
AVW_RenderedImage *last_rendered;
```

DESCRIPTION

AVW_RenderVolume() renders a volume by ray casting based on the parameters set in *param*. Elements of the *AVW_RenderParameters* structure returned by *AVW_InitializeRenderParameters()* and used by *AVW_RenderVolume() are:*

param->Type specifies the type of ray casting preformed. Supported values include:

AVW_DEPTH_SHADING – The value of each output pixel is a function of depth only. The depth of the first voxel found along the ray path is used to determine the brightness of that pixel. Closer voxels will appear brighter than distant voxels. This output image may be further enhanced by *AVW_ProcessZGradients()*.

AVW_GRADIENT_SHADING – The gray scale gradient vector is computed using a 3D neighborhood about the surface voxel. The value projected at each output location is the dot product of the gradient vector and an independantly specified light source vector. This simulates the appearance of a reflective surface under uniform-field illumination.

AVW_VOLUME_COMPOSITING – The volumetric compositing algorithm integrates the gradient-shaded value of all voxels along the ray path. The contribution of each gradient-shaded voxel value is weighted by an opacity function described in the *AVW_CompositeInfo* structure. Voxel intensity is used to determined the voxels color and opacity contribution during the ray casting.

AVW_MAX_INTENSITY_PROJECTION – The maximum voxel intensity along the ray path is used.

AVW_SUMMED_VOXEL_PROJECTION – The average of all voxels along the ray path is used.

AVW_SURFACE_PROJECTION – Once a voxel within the threshold limits is detected, the average of the next N voxels is computed.

AVW_TRANSPARENCY_SHADING – Available only when an *AVW_ObjectMap* has been specified, this type produces a 24-bit true color projections of all the surface gradients along the ray path. Opacity parameters may be specified for each object controlling the transparency.

AVW_DELETE_VOXELS – Not really a rendering type, but causes the ray casting algorithm to delete (or convert to a specified value) voxels along the ray path. See the *param->DeleteDepth* and *param->DeleteValue* parameters for more information.

param->ThresholdMinimum and *param->ThresholdMaximum* specify the range of acceptable voxel values. Voxels outside the specified range are ignored.

param->ClipLowX, param->ClipLowY, param->ClipLowZ, param->ClipHighX, param->ClipHighY, and *param->ClipHighZ* specify the portion of the *AVW_Volume* to render.

param->ClipPlaneMinimum and *param->ClipPlaneMaximum* specify the starting and ending depths for the ray casting process.

param->ClipShading specifies the type of shading that is used when the ray casting process begins at a voxel which is within the threshold range (and object enabled). The value is only used when the render *Type* is set to *AVW_GRADIENT_SHADING*. Possible values are:

AVW_CLIP_SHADED – A value based on only depth is used to create the shaded pixel.

AVW_CLIP_ACTUAL – The grayscale value of the intersected voxel is used.

AVW_CLIP_REMOVE_AND_RENDER – The voxel along the raypath, just before the intersected voxel, is temporarily set to the input volume minimum intensity value and the gradient shading is calculated.

AVW_CLIP_RENDER_AS_IS – The gradient shading is calculated for the voxel without any attempt to correct for the fact that the shading will most likely be unpredictable because no surface will be detected..

param->RenderWidth, param->RenderHeight, and *param->RenderDepth* specify the size of the rendered space. By default these values are set to the maximum of the X, Y, or Z input dimension.

param->MaximumPixelValue and *param->MinimumPixelValue* specify the range of possible output values for the reflectance renderings. Transmission renderings use the maximum and minimum values with the *AVW_Volume* as the output range.

param->SurfaceThickness specifies the maximum thickness for the *AVW_SURFACE_PROJECTION* rendering type.

param->SurfaceSkip specify the number of voxels to skip before summing begins for the *AVW_SURFACE_PROJECTION* rendering type.

param->MIP_Weight specify the weighting options used during a *AVW_SURFACE_PROJECTION*. Options include: *AVW_NO_WEIGHTING, AVW_WEIGHT_BEFORE*, and *AVW_WEIGHT_AFTER. AVW_WEIGHT_BEFORE* indicates that before a voxel is checked to see if it is the maximum value, a weighting factor is applied. The weighting factor is determined by dividing the length of the ray left to cast, by it’s total length. AVW_WEIGHT_AFTER determines the maximum voxel along the entire ray casting path and then applies the weightinh factor described above.

param->Matrix is an *AVW_Matrix* which specifies the rotation and translation transformation applied to the *AVW_Volume* during the rendering process. Scale could also be specified as part of the matrix, but it’s recommended that the *ScaleX, ScaleY*, and *ScaleZ* parameters be used for Scale.

param->LightMatrix specifies the vector of the light source used in reflective renderings.

param->RenderMask is an *AVW_Image* which specifies a specific area to be rerendered. *NULL* is the default, specifing the entire rendering space is rendered each time *AVW_RenderVolume* is called. Only *AVW_Images* with a data type of *AVW_UNSIGNED_CHAR* are supported. This *AVW_Image* should always have dimensions equal to *param->RenderWidth* and *param->RenderedHeight*.

param->MaskValue specifies the pixel value within the *param->RenderMask* which indicates rendering should occur for this output pixel.

param->DeleteDepth specifies the number of voxels along the ray path which are deleted (changed to *param->DeleteValue).* The following settings are valid when the *AVW_DELETE_VOXELS* rendering type is specified:

> *AVW_DELETE_ALL_THE_WAY* – All voxels along the ray path are changed to the value to specified by *param->DeleteValue.*
>
> *AVW_DELETE_SINGLE_LAYER* – Once a value within the theshold limits is detected, all voxels with values within the threshold range are deleted until a voxel outside the threshold range is detected, at which time deleting stops.
>
> *AVW_DELETE_SINGLE_VOXEL* – The first voxel along the ray path which is within the threshold range is deleted.
>
> *AVW_DEFINE_ALL_THE_WAY* – All object map locations along the ray path are changed to the value to specified by *param->DeleteValue.*
>
> *AVW_DEFINE_SINGLE_LAYER* – Once a value within the theshold range is detected, all voxels with values within the threshold limits are redfined until a voxel outside the threshold range is detected, at which time deleting stops.
>
> *AVW_DEFINE_SINGLE_VOXEL* – The first object map location along the ray path which is within the threshold limits is redefined.
>
> *NOTE:* A 3x3x3 region surrounding a voxel is deleted instead of just a single voxel, this compensates for spaces which may occur between the ray paths.

param->DeleteValue specifies the value each voxel is changed to, when the *AVW_DELETE_VOXELS* rendering type is specified.

param->ScaleX, *param->ScaleY*, and *param->ScaleZ* specify the scale factors to be applied to the inputs during the rendering process.

param->PerspectiveType specifies the type of rendering to be performed. *AVW_PERSPECTIVE_INT* renders the image using the voxels without interpolation, possibly resulting in "blocky" renderings. If *param->PerspectiveType* is set to *AVW_PERSPECTIVE_FLOAT*, the rendered image is generated using an on the fly intepolation rendering algorithm. If *param->PerspectiveType* is set to *AVW_PERSPECTIVE_OFF*, parallel rendering is performed. For perspective rendering, the render matrix is interperted as a viewing direction for the camera model, however the parallel rendering conventions are followed, i.e. the matrix is left handed, and the identity matrix provides a view along the positive Z axis with X increasing to the right and Y increasing in the vertical direction. Perspective rendering does not support scaling at this time. For anisotropic data, rescaling during loading is the best option.

param->EyePosition specifies the location of the camera model used to generate the rendering. This parameter is only used when *param->PerspectiveType* is set to *AVW_PERSPECTIVE_FLOAT* or *AVW_PERSPECTIVE_INT*. The *X*, *Y*, and *Z* coordinates of the *AVW_FPoint3* structure refer to the position of the camera relative to the center of the volume, thus a position of (0,0,0) in a particular volume is located at voxel (Width / 2, Height / 2, Depth / 2).

param->XFieldOfViewAngle and *param->YFieldOfViewAngle* specify the field of view (FOV) angle of the camera model used to generate the image. Increasing the FOV results in a zoom out effect, if the position remains the same. Decreasing the FOV results in a zoom in effect.

param->SpecularFactor specifies the ratio of gradient shading to specular shading. If *param->SpecularFactor* is set to 0, the rendering will be shaded entirely using gradient shading with no performance penalty. If *param->SpecularFactor is set to .1, the*

param->SpecularExponent specifies the degree of fall-off for specular shading. High values (about 10) result in very small specular highlights, while small values (about 1 or 2) result in diffuse highlights.

param->CompositeInfo specifies a pointer to a structure which contains compositing information. This must contain a valid pointer when the *param->Type is set to AVW_VOLUME_COMPOSITING*. See *AVW_CompositeInfo* for more information.

param->BackgroundColor and *param->BackgroundValue* are used to specify the background color or value. The rendering type, input volumes datatype, and whether an object map is loaded determines which value is used. If the output is a grayscale image, *param->BackgroundValue* is used. If the output is a color image, then *param->BackgroundColor* will be used. *Background Color* is a packed RGB value which can be produced with the *AVW_RGB* macro.

BackgroundColor and *BackgroundValue* are used to specify the background color or value. The rendering type, input volumes datatype, and whether an object map is loaded determines which value is used. If the output is a grayscale image, *BackgroundValue* is used. If the output is a color image, then *BackgroundColor* will be used. *Background Color* is a packed RGB value which can be produced with the *AVW_RGB* macro.

RenderMode is normally set to *AVW_RENDER_NORMAL*, but when set to *AVW_PREPARE_FOR_MOVE*, the *AVW_RenderVolume()* function will process the object specified in the *InteractiveObject* seperately and return the "visible surface" in the *AVW_RenderedImage* member called *InteractiveSurface*. This results in the input to *AVW_RenderVisibleSurface()* which produces output which can be combined using *AVW_MergeRendered()* with the rendering returned at the time the *InteractiveSurface* was produced. This entire process allows an interface to be build to interactively move and rotate objects. *RenderMode* can be set to *AVW_RERENDER_MOVED*, at the completion of the object transformation to rerender any missing data.

param->RenderMode is normally set to *AVW_RENDER_NORMAL*, but when set to *AVW_PREPARE_FOR_MOVE*, the *AVW_RenderVolume()* function will process the object specified in the *param->InteractiveObject* seperately and return the "visible surface" in the *AVW_RenderedImage* member called *rendered->InteractiveSurface*. This results in the input to *AVW_RenderVisibleSurface()* which produces output which can be combined using *AVW_MergeRendered()* with the rendering returned at the time the *rendered->InteractiveSurface* was produced. This entire process allows an interface to be build to interactively move and rotate objects. *param->RenderMode* can be set to *AVW_RERENDER_MOVED*, at the completion of the object transformation to rerender any missing data.

param->Internal contains parameter which are used internally within the rendering functions. *CHANGING OF INTERNAL PARAMETERS IS DISCOURAGED!*

Elements of the *AVW_RenderedImage* structure returned by *AVW_RenderVolume() are:*

> *rendered->Width, rendered->Height,* and *rendered->Depth* contains the size of the output rendered space.
>
> *rendered->MaximumPixelValue* and *rendered->MinimumPixelValue* contain the range of the values within the output data.
>
> *rendered->Image* contains the output image.
>
> *rendered->ZBuffer* contains an internal buffer indicating the position of all invisible faces.
>
> *rendered->PBuffer* contains a buffer which indicates the depth of each voxel.
>
> *rendered->Volume* contains a pointer to the *AVW_Volume* used to generate the rendered image.
>
> *rendered->ObjectMap* contains a pointer to the *AVW_ObjectMap* used to generate the rendered image.
>
> *rendered->Matrix* contains the *AVW_Matrix* necessary to convert 3D points into the rendered space.
>
> *rendered->InverseMatrix* contains the *AVW_Matrix* necessary to convert points in the rendered space back into the *AVW_Volume* space.

rendered->PerspectiveType indicates if one of the perspective types was used to generate this rendered image.

rendered->EyePosition indicates the eye position of the last perspective rendering.

rendered->XFieldOfViewAngle and rendered->YFieldOfViewAngle indicate the field of view for the last perspective rendering.

rendered->MergedMap is normally *NULL, but after a call to AVW_MergeRendered()* it is filled with information about where pixels came from and how they were generated.

rendered->InteractiveSurface is normally *NULL*. It's filled only when *AVW_RenderVolume()* is used with the *param->RenderMode* set to *AVW_PREPARE_FOR_MOVE* or *AVW_RERENDER_MOVED*.

Last_rendered is provided as a method of reusing an existing *AVW_RenderedImage*. Reuse is possible only if the size and data type of the provided *last_rendered* meet the requirements of the function. In this case the pointer to *last_rendered* is returned by the function. If not reusable *last_rendered* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RenderVolume()* returns a pointer to a *AVW_RenderedImage* structure. This structure contains the output of the rendering process. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RenderVolume() will fail if one or more of the following is true:

BADMAL
Unable to allocate sufficient memory.

CFLSZ
The AVW_Volume and AVW_ObjectMap have conflicting dimensions.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_DestroyRenderedImage(), AVW_InitializeRenderParameters(), AVW_RenderGradients(), AVW_RenderOblique(), AVW_RenderVisibleSurface(), AVW_RenderedImage, AVW_RenderParameters, AVW_CompositeInfo

NAME

AVW_RenderableVolume – thresholds a volume

SYNOPSIS

```
#include "AVW.h"
#include "AVW_ObjectMap.h"

AVW_Volume *AVW_RenderableVolume(in_volume, omap, tmax, tmin, out_volume)
AVW_Volume *in_volume;
AVW_ObjectMap *omap;
double tmax, tmin;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_RenderableVolume() processes an input volume and object map, returning a volume which contains *1s* for all voxels in enabled objects and within the threshold range defined by *tmax* and *tmin*. *0s* are returned for all other voxels.

The returned *AVW_Volume* is of the data type *AVW_UNSIGNED_CHAR*.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RenderableVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RenderableVolume() will fail if the following is true:

ILLDT
Illegal data type.

SEE ALSO

AVW_ThresholdVolume(), AVW_Objectmap, AVW_Volume

NAME

AVW_ResetIntensityStats – resets intensity statistics

SYNOPSIS

#include "AVW_Measure.h"

```
int AVW_ResetIntensityStats( stats)
AVW_IntensityStats *stats;
```

DESCRIPTION

AVW_ResetIntensityStats() sets all of the members of *stats* to zero.

RETURN VALUES

If successful *AVW_ComputeImageIntensityStats()* returns *AVW_SUCCESS*.

SEE ALSO

AVW_ComputeVolumeIntensityStats() AVW_ComputeImageIntensityStats(), AVW_IntensityStats

NAME AVW_ResizeImage – resizes an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ResizeImage(in_image, width, height, interpolate, out_image)
AVW_Image *in_image;
int width;
int height;
int interpolate;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ResizeImage() generates an *AVW_Image*, *out_image*, of the size specified by the *width* and *height* parameters.

Interpolate determines the method of interpolation to use. Choose from:

AVW_RESIZE_NN_ID - Input driven nearest neighbor

AVW_RESIZE_LINEAR_ID - Input driven tri-linear

AVW_RESIZE_NN_OD - Output driven nearest neighbor

AVW_RESIZE_LINEAR_OD - Output driven tri-linear

AVW_RESIZE_CUBIC_SPLINE_INTERPOLATE - Cubic Spline

AVW_RESIZE_WINDOWED_SINC_INTERPOLATE - Windowed Sinc

AVW_ResizeImage() uses highly optimized code when the specified width and height values are one of the following scale factors: 25%, 33%, 50%, 100%, 200%, 300%, and 400%. Both *width* and *height* must have the same scale factor for the optimized code to be used. The speed advantages vary depending on *in_image* and *out_image* sizes but 10 to 20 times faster results can be expected.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ResizeImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ResizeImage() will fail if one or more of the following are true:

ILLDT
Input data type not supported by this function.

BADMAL
Unable to allocate sufficient memory to complete request.

SEE ALSO

AVW_CropImage(), AVW_GetSubImage(), AVW_GetSubImageWithIncrements(), AVW_PadImage(), AVW_PutSubImage(), AVW_ResizeVolume(), AVW_TransformImage(), AVW_Image

NAME

AVW_ResizeSubImage – resizes an image

SYNOPSIS

```
#include "AVW.h"

int AVW_ResizeSubImage(in_image, in_rect2, interpolate_type, out_image, out_rect2)
AVW_Image *in_image;
AVW_Rect2 *in_rect2;
int interpolate_type;
AVW_Image *out_image;
AVW_Rect2 *out_rect2;
```

DESCRIPTION

AVW_ResizeSubImage() resizes a rectanglular area specified in *in_rect2* with *in_image.* The output size and position within *out_image* is specified in *Interpolate_type specifies the type of interpolation*

AVW_RESIZE_NN_ID specifies a nearest neighbor input driven algorithm.

AVW_RESIZE_LINEAR_ID specifies a linear input driven algorithm.

AVW_RESIZE_NN_OD specifies a nearest neighbor output driven algorithm.

AVW_RESIZE_LINEAR_OD specifies a linear output driven algorithm.

AVW_RESIZE_CUBIC_SPLINE_INTERPOLATE specifies a cubic spline algorithm.

AVW_RESIZE_WINDOWED_SINC_INTERPOLATE specifies a windowed sinc algorithm.

ERRORS

AVW_ResizeSubImage() will fail if one or more of the following are true:

ILLDT
Input data type not supported by this function.

BADMAL
Unable to allocate sufficient memory to complete request.

SEE ALSO

AVW_GetSubImage(), AVW_PutSubImage(), AVW_ResizeImage(), AVW_Image

NAME

AVW_ResizeSubVolume – resizes a volume

SYNOPSIS

#include "AVW.h"

int AVW_ResizeSubVolume(in_volume, in_rect3, interpolate_type, out_volume, out_rect3)
AVW_Volume *in_volume;
AVW_Rect3 *in_rect3;
int interpolate_type;
AVW_Volume *out_volume;
AVW_Rect3 *out_rect3;

DESCRIPTION

AVW_ResizeSubVolume() resizes a rectanglular area specified in *in_rect2* with *in_image.* The output size and position within *out_image* is specified in *Interpolate_type specifies the type of interpolation*

AVW_RESIZE_NN_ID specifies a nearest neighbor input driven algorithm.

AVW_RESIZE_LINEAR_ID specifies a linear input driven algorithm.

AVW_RESIZE_NN_OD specifies a nearest neighbor output driven algorithm.

AVW_RESIZE_LINEAR_OD specifies a linear output driven algorithm.

AVW_RESIZE_CUBIC_SPLINE_INTERPOLATE specifies a cubic spline algorithm.

AVW_RESIZE_WINDOWED_SINC_INTERPOLATE specifies a windowed sinc algorithm.

RETURN VALUES

If successful *AVW_ResizeSubVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ResizeSubVolume() will fail if one or more of the following are true:

ILLDT
Input data type not supported by this function.

BADMAL
Unable to allocate sufficient memory to complete request.

SEE ALSO

AVW_ResizeVolume(), AVW_Volume

NAME

AVW_ResizeVolume – resizes a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ResizeVolume(in_volume, out_width, out_height, out_slices,
                interpolate, out_volume)
AVW_Volume *in_volume;
int out_width, out_height, out_slices;
int interpolate;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ResizeVolume() performs trilinear interpolation to resize a volume to a specified size.

Interpolate determines the method of interpolation to use. Choose from:

AVW_RESIZE_NN_ID **-** Input driven nearest neighbor

AVW_RESIZE_LINEAR_ID **-** Input driven tri-linear

AVW_RESIZE_NN_OD **-** Output driven nearest neighbor

AVW_RESIZE_LINEAR_OD **-** Output driven tri-linear

AVW_RESIZE_CUBIC_SPLINE_INTERPOLATE **-** Cubic Spline

AVW_RESIZE_WINDOWED_SINC_INTERPOLATE **-** Windowed Sinc

In_volume is the input volume.

Out_width indicates the desired output width.

Out_height indicates the desired output height.

Out_slices indicates the desired number of output slices.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_ResizeVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ResizeVolume() will fail if one or more of the following are true:

ILLPAR
Illegal parameter.

BADMAL
Unable to allocate enough memory.

ILLDT
Unsupported data type.

SEE ALSO *AVW_GetSubVolume(), AVW_PadVolume(), AVW_PutSubVolume(), AVW_ResizeImage(), AVW_ResizeVolumeSliceBySlice(), AVW_ResizeVolumeUsingShapeInt(), AVW_TransformVolume(), AVW_Volume*

NAME

AVW_ResizeVolumeSliceBySlice – resizes a volume slice by slice

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ResizeVolumeSliceBySlice(in_image, in_count, in_slices,
                out_width, out_height, out_slices,
                out_count, interpolate, out_image)
AVW_Image *in_image;
int in_count, in_slices;
int out_width, out_height, out_slices;
int *out_count;
int interpolate;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ResizeVolumeSliceBySlice() performs a resize operation one slice at a time. This function is called one or more times with each slice in an input volume. It may or may not return an output slice, depending on whether enough input data was available to calculate the resized output slice.

Interpolate determines the method of interpolation to use. Choose from:

AVW_RESIZE_NN_ID **-** Input driven nearest neighbor

AVW_RESIZE_LINEAR_ID **-** Input driven tri-linear

AVW_RESIZE_NN_OD **-** Output driven nearest neighbor

AVW_RESIZE_LINEAR_OD **-** Output driven tri-linear

AVW_RESIZE_CUBIC_SPLINE_INTERPOLATE **-** Cubic Spline

AVW_RESIZE_WINDOWED_SINC_INTERPOLATE **-** Windowed Sinc

In_image is the input slice.

In_count indicates which slice from the input volume is being passed in. Legal values are 1 thru *in_slices.*

In_slices is the total number of slices in the input volume.

Out_width indicates the desired output width.

Out_height indicates the desired output height.

Out_slices indicates the desired number of output slices.

Out_count indicates the output slice position within the output volume. A value of 0 (zero) indicates no output slice was being returned. A positive value indicates a output slice was returned and *AVW_ResizeVolumeSliceBySlice* should be called again with *in_count* set to a value of 0 (zero) and all other parameters the same, until a 0 (zero) *out_count* value is returned.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful, *AVW_ResizeVolumeSliceBySlice()* returns an *AVW_Image*. This returned image is only used if *out_count* is a positive non-zero value. On failure, it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ResizeVolumeSliceBySlice() will fail if one or more of the following are true:

ILLPAR
Illegal parameter.

BADMAL
Unable to allocate enough memory.

ILLDT
Unsupported data type.

SEE ALSO

AVW_GetSubVolume(), AVW_PadVolume(), AVW_PutSubVolume(), AVW_ResizeImage(), AVW_ResizeVolume(), AVW_ResizeVolumeUsingShapeInt(), AVW_Image, AVW_Volume

NAME

AVW_ResizeVolumeUsingShapeInt – performs binary shape based interpolation

SYNOPSIS

#include "AVW.h"

AVW_Volume *AVW_ResizeVolumeUsingShapeInt(in_volume, out_volume)
AVW_Volume *in_volume;
AVW_Volume *out_volume;

DESCRIPTION

AVW_ResizeVolumeUsingShapeInt() creates a binary isotropic volume from *in_volume* by doing shape based interpolation. The *VoxelSize* and *SliceThickness* strings and floating point values must be stored in the *Info* string of the *AVW_Volume* structure for *in_volume*. The *VoxelSize* must be less than the *SliceThickness*. *AVW_ResizeVolumeUsingShapeInt()* only works on *AVW_UNSIGNED_CHAR* data type.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meets the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ResizeVolumeUsingShapeInt()* returns a binary interpolated *AVW_Volume*. On failure it returns *NULL* and sets the *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ResizeVolumeUsingShapeInt() will fail if one or more of the following are true:

CFLSZ
Conflicting input / output size or type.

ILLDT
Illegal data type.

NOMATCH
Info string is missing required values.

SEE ALSO

AVW_GetSubVolume(), AVW_PadVolume(), AVW_PutSubVolume(), AVW_ResizeImage(), AVW_ResizeVolume(), AVW_ResizeVolumeSliceBySlice(), AVW_PutNumericInfo(), AVW_Volume

NAME

AVW_ReverseBits – reverses bits

SYNOPSIS

```
#include      <stdio.h>
#include      "AVW.h"

AVW_ReverseBits(in, out, bytes)
void *in, *out;
int bytes;
```

DESCRIPTION

ReverseBits() reverses the bit order of *in* over a specified number of *bytes* and returns the results in *out.*

SEE ALSO

AVW_QuadSwapImage(), AVW_READSWAP(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapLong(), AVW_SwapShort(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME

AVW_Rotate90Image – quick 90 degree rotation of an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_Rotate90Image(in_image, direction, out_image)
AVW_Image *in_image;
int direction;
AVW_Image *out_image;
```

DESCRIPTION

AVW_Rotate90Image() rotates *in_image* by 90 degrees in the direction indicated by *direction*

If *direction* is zero, positive, or *AVW_COUNTERCLOCKWISE*, the image will be rotated positive 90 degrees (counterclockwise). If *direction* is less than 0 or *AVW_CLOCKWISE* , the image will be rotated negative 90 degrees (clockwise).

The returned image will have width and height equal to *in_image* height and width, respectively.

RETURN VALUES

If successful *AVW_Rotate90Image()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_Rotate90Image() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

ILLDT
Datatype is unsupported.

SEE ALSO

AVW_FlipImage(), AVW_ShiftImage(), AVW_TransformImage(), AVW_Image

NAME

AVW_RotateMatrix – rotates a transformation matrix

SYNOPSIS

```
#include "AVW.h"

AVW_Matrix *AVW_RotateMatrix(in_matrix, xangle, yangle, zangle, out_matrix)
AVW_Matrix *in_matrix;
double xangle;
double yangle;
double zangle;
AVW_Matrix *out_matrix;
```

DESCRIPTION

AVW_RotateMatrix() applies the rotation specified by *xangle, yangle,* and *zangle* (specified in degress) to the *AVW_Matrix, mat.* The rotation is applied to the X axis first, then the Y axis, and finally the Z axis. *Out_matrix* is provided as a method of reusing an existing *AVW_Matrix. (See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RotateMatrix()* returns an *AVW_Matrix.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_InvertMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis() AVW_MatrixAngles() AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_SetIdentityMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_Matrix

NAME

AVW_RotatePointList2 – rotates the points in a point list

SYNOPSIS

```
#include "AVW.h"

int AVW_RotatePointList2(ptlist, angle)
AVW_PointList2 *ptlist;
double angle;
```

DESCRIPTION

AVW_RotatePointList2() applies the rotation specified by *angle* in degrees to the *AVW_PointList2*, *ptlist.*

RETURN VALUES

If successful *AVW_RotatePointList2()* returns *AVW_SUCCESS.* On failure *AVW_FAIL* is returned.

SEE ALSO

AVW_AddPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_ScalePointList2(), AVW_TranslatePointList2(), AVW_PointList2

NAME

AVW_RoundImage – round values in a float image to nearest integers

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_RoundImage(in_image, datatype, out_image)
AVW_Image *in_image;
int datatype;
AVW_Image *out_image;
```

DESCRIPTION

AVW_RoundImage() converts *in_image*, which must have a AVW_FLOAT DataType, to the specified *datatype.* All values are rounded to the nearest integer. Acceptable values for *datatype,* as defined in *AVW.h,* are: *AVW_UNSIGNED_CHAR, AVW_SIGNED_CHAR, AVW_UNSIGNED_SHORT, AVW_SIGNED_SHORT, AVW_UNSIGNED_INT, AVW_SIGNED_INT, AVW_FLOAT, AVW_COMPLEX, AVW_COLOR.*

If the volume is converted to a data type which can hold less information, or information in a different range, the data is clipped at the maximum and minimum values possible for the new data type. No intensity scaling is attempted during the conversion process.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RoundImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RoundImage() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for output volume.

ILLDT
Unknown or unsupported input or output datatype.

SEE ALSO

AVW_RoundVolume(), AVW_ConvertImage(), AVW_Image

NAME

AVW_RoundVolume – round values in a float volume to nearest integers

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_RoundVolume(in_volume, datatype, out_volume)
AVW_Volume *in_volume;
int datatype;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_RoundVolume() converts *in_volume* which must have a AVW_FLOAT DataType, to the specified *datatype.* All values are rounded to the nearest integer. Acceptable values for *datatype,* as defined in *AVW.h,* are: *AVW_UNSIGNED_CHAR, AVW_SIGNED_CHAR, AVW_UNSIGNED_SHORT, AVW_SIGNED_SHORT, AVW_UNSIGNED_INT, AVW_SIGNED_INT, AVW_FLOAT, AVW_COMPLEX, AVW_COLOR.*

If the volume is convert to a data type which can hold less information, or information in a different range, the data is clipped at the maximum and minimum values possible for the new data type. No intensity scaling is attempted during the conversion process.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_RoundVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_RoundVolume() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for output volume.

ILLDT
Unknown or unsupported input or output datatype.

SEE ALSO

AVW_RoundImage(), AVW_ConvertVolume(), AVW_Volume

NAME

AVW_SaveColormap – saves a color map

SYNOPSIS

```
#include "AVW.h"

int AVW_SaveColormap(file, colormap)
char *filename;
AVW_Colormap *colormap;
```

DESCRIPTION

AVW_SaveColormap() saves the contents of the *colormap* to the disk file specified by *filename.*

AVW colormap files usually end in *.lkup.* Colormap values are stored as ASCII strings representing values from 0 to 255. The red value for the first cell, if followed by the green value for the first cell, followed by the blue value for the first cell. This order is followed for each colorcell defined in the file.

RETURN VALUES

If successful *AVW_SaveColormap()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SaveColormap() will fail if one or more of the following is true:

BADOPEN
: Could open file for reading or writing.

BADWRITE
: Error occurred while writing file.

ILLPAR
: Illegal parameter(s).

SEE ALSO

AVW_LoadColormap(), AVW_CreateColormap(), AVW_DestroyColormap(), AVW_Colormap

NAME

AVW_SaveCompositeInfo – saves compositing information

SYNOPSIS

```
#include "AVW_CompositeInfo.h"

int AVW_SaveCompositeInfo(file, cinfo)
char *filename;
AVW_CompositeInfo *cinfo;
```

DESCRIPTION

AVW_SaveCompositeInfo() saves the contents of *cinfo* to the disk file specified by *filename.*

RETURN VALUES

If successful *AVW_SaveCompositeInfo()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SaveCompositeInfo() will fail if one or more of the following is true:

BADOPEN
Could open file for reading or writing.

BADWRITE
Error occurred while writing file.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_CreateCompositeInfo(), AVW_DestroyCompositeInfo(), AVW_LoadCompositeInfo(), AVW_CompositeInfo

NAME

AVW_SaveContourSurface – writes a contour surface to a file

SYNOPSIS

```
#include "AVW_Model.h"

int AVW_SaveContourSurface(srfc, path, rp_param)
AVW_ContourSurface *srfc;
char *path;
AVW_RPParam *rp_param;
```

DESCRIPTION

AVW_SaveContourSurface() writes the contour surface, *srfc,* to the file given by *path. rp_param* is used to determine which output format is to be used and how blank slices are to be managed. Supported output formats are: *AVW_HPGL_SURFACE, AVW_POGO_SURFACE, AVW_SLC_SURFACE,* and *AVW_SSD_ASCII_SURFACE.*

RETURN VALUES

AVW_SaveContourSurface() returns *AVW_SUCCESS* if successful. On failure, *AVW_FAIL* will be returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure.

ERRORS

AVW_SaveContourSurface() will fail if one of the following is true:

BDWRTE
    File write failed.

BDOPEN
    Unable to open file.

SEE ALSO

AVW_DestroyContourSurface(), AVW_SliceVolume(), AVW_ContourSurface, AVW_RPParam

NAME

AVW_SaveObjectMap – saves an object map

SYNOPSIS

#include "AVW_ObjectMap.h"

int AVW_SaveObjectMap(file, object_map)
char *filename;
AVW_ObjectMap *object_map;

DESCRIPTION

AVW_SaveObjectMap() saves the contents of the *object_map* to the disk file specified by *filename.*

RETURN VALUES

If successful *AVW_SaveObjectMap()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SaveObjectMap() will fail if one or more of the following is true:

BADOPEN
Could open file for reading or writing.

BADWRITE
Error occurred while writing file.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddObject(), AVW_ComputeObjectStats(), AVW_CopyObjectMap(), AVW_CreateObjectMap(), AVW_DeleteObject(), AVW_DestroyObjectMap(), AVW_GetObject(), AVW_LoadObjectMap(), AVW_ObjectScaleImage(), AVW_PutObject(), AVW_RenderVolume(), AVW_ObjectMap

NAME

AVW_SaveTiledSurface – writes an ordered surface to a file

SYNOPSIS

```
#include "AVW_Model.h"

int AVW_SaveTiledSurface(srfc, path, format)
AVW_TiledSurface *srfc;
char *path;
int format;
```

DESCRIPTION

AVW_SaveTiledSurface() writes the ordered surface in *srfc* to the file given by *path. Format* is used to determine which output format is to be used. Supported output formats are: *AVW_VRIO_SURFACE, AVW_INVENTOR_SURFACE, AVW_VRML_SURFACE, AVW_DXF_SURFACE, AVW_OBJ_SURFACE, AVW_POLY_SURFACE,* and *AVW_STL_SURFACE.*

RETURN VALUES

AVW_SaveTiledSurface() returns *AVW_SUCCESS* if successful. On failure, *AVW_FAIL* will be returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* will be set to values corresponding to the cause of the failure.

ERRORS

AVW_SaveTiledSurface() will fail if one of the following is true:

BDWRTE
    File write failed.

BDOPEN
    Unable to open file.

SEE ALSO

AVW_DestroyTiledSurface(), AVW_LoadTiledSurface(), AVW_DrawTiledSurface(), AVW_TileVolume(), AVW_VolumeToSLC(), AVW_TiledSurface

NAME

AVW_SaveTree – saves a tree

SYNOPSIS

#include "AVW_Tree.h"

int AVW_SaveTree(tree, file)
AVW_Tree *tree;
char *filename;

DESCRIPTION

AVW_SaveTree() saves the contents of the *tree* to the disk file specified by *filename.*

RETURN VALUES

If successful *AVW_SaveTree()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SaveTree() will fail if one or more of the following is true:

BADOPEN
Could open file for reading or writing.

BADWRITE
Error occurred while writing file.

ILLPAR
Illegal parameter(s).

SEE ALSO

AVW_AddTreeChild(), AVW_CreateTree(), AVW_DestroyTree(), AVW_FindTreeIndex(), AVW_FindTreeStart(), AVW_LoadTree(), AVW_MakeTree(), AVW_PruneTree(), AVW_TreeAnalysis(), AVW_TreePoint, AVW_Tree

NAME

AVW_ScaleMatrix – scales a transformation matrix

SYNOPSIS

```
#include "AVW.h"

AVW_Matrix *AVW_ScaleMatrix(in_matrix, xfactor, yfactor, zfactor, out_matrix)
AVW_Matrix *in_matrix;
double xfactor, yfactor, zfactor;
AVW_Matrix *out_matrix;
```

DESCRIPTION

AVW_ScaleMatrix() applies the scaling specified by *xfactor*, *yfactor*, and *zfactor* to the *AVW_Matrix, in_matrix.*

Out_matrix is provided as a method of reusing an existing *AVW_Matrix. (See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ScaleMatrix()* returns an *AVW_Matrix.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_InvertMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis() AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_SetIdentityMatrix(), AVW_TranslateMatrix(), AVW_Matrix

NAME

AVW_ScalePointList2 – scales the points in a point list

SYNOPSIS

```
#include "AVW.h"

int AVW_ScalePointList2(ptlist, xscale, yscale)
AVW_PointList2 *ptlist;
double xscale, yscale;
```

DESCRIPTION

AVW_ScalePointList2() applies the scaling specified by *xscale* and *yscale*, to the *AVW_PointList2*, *ptlist*.

RETURN VALUES

If successful *AVW_ScalePointList2()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned.

SEE ALSO

AVW_AddPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_RotatePointList2(), AVW_TranslatePointList2(), AVW_PointList2

NAME

AVW_Scattergram – creates a scattergram image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_Scattergram(img1, img2, mask, scattergram)
AVW_Image *img1, *img2, *mask, *scattergram;
```

DESCRIPTION

AVW_Scattergram() creates a 2 dimensional scattergram from a pair of compatible images and an optional training mask image which indicates the positions of known classes. Compatible images are those with the same width and height; they must also be of type *AVW_UNSIGNED_CHAR*.

A scattergram created with a training *mask* image can be further classified with *AVW_ClassifyScattergram().*

If the *mask* image is NULL the returned scattergram will be of type *AVW_UNSIGNED_INT*. If the *mask* image is the same size as the input images and has known pixel classes indicated with class numbers the returned scattergram will be of type *AVW_UNSIGNED_CHAR* with known pixels set to the class number and all others zero.

The scattergram is a representation of the frequency and occurrences of pairs of values from *img1* and *img2* The values 0 to 255 are represented on the vertical and horizontal axes of *img1* and *img2* respectively. Values at point x,y in the scattergram are a count of pixels with a value of x from *img2* and a value of y from *img1.*

A masked scattergram image is used for fast voxel classification with *AVW_ClassifyFromScattergram().*

If *scattergram* is an *AVW_Image* 256 pixels square and of data type *AVW_UNSIGNED_INT* it will be reused. Otherwise *scattergram* will be destroyed (if necessary) and / or created.

RETURN VALUES

If successful *AVW_Scattergram()* returns an *AVW_Image.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_Scattergram() will fail if one or more of the following are true:

ILLDT
Illegal data type given in type.

ILLIMG
Illegal Image; img1 and img2 are incompatible.

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_ClassifyFromScattergram(), AVW_ClassifyScattergram(), AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_Image

NAME

AVW_ScattergramFromImages – creates a scattergram image from images

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ScattergramFromImages(img1, img2, mask, scattergram)
AVW_Image *img1, *img2, *mask, *scattergram;
```

DESCRIPTION

AVW_ScattergramFromImages() creates a 2 dimensional scattergram from a pair of compatible images and an optional training mask image which indicates the positions of known classes. Compatible images are those with the same width and height; they must also be of type *AVW_UNSIGNED_CHAR*.

A scattergram created with a training *mask* image can be further classified with *AVW_ClassifyScattergram()*.

If the *mask* image is NULL the returned scattergram will be of type *AVW_UNSIGNED_INT*. If the *mask* image is the same size as the input images and has known pixel classes indicated with class numbers the returned scattergram will be of type *AVW_UNSIGNED_CHAR* with known pixels set to the class number and all others zero.

The scattergram is a representation of the frequency and occurrences of pairs of values from *img1* and *img2* The values 0 to 255 are represented on the vertical and horizontal axes of *img1* and *img2* respectively. Values at point x,y in the scattergram are a count of pixels with a value of x from *img2* and a value of y from *img1*.

A masked scattergram image is used for fast voxel classification with *AVW_ClassifyFromScattergram()*.

If *scattergram* is an *AVW_Image* 256 pixels square and of data type *AVW_UNSIGNED_INT* it will be reused. Otherwise *scattergram* will be destroyed (if necessary) and / or created.

RETURN VALUES

If successful *AVW_Scattergram()* returns an *AVW_Image*. On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_Scattergram() will fail if one or more of the following are true:

ILLDT
Illegal data type given in type.

ILLIMG
Illegal Image; img1 and img2 are incompatible.

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_ScattergramFromVolumes(), AVW_ClassifyFromScattergram(), AVW_ClassifyScattergram(), AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_Image

NAME

AVW_ScattergramFromVolumes – creates a scattergram image from volumes

SYNOPSIS

#include "AVW.h"

AVW_Image *AVW_ScattergramFromVolumes(vol1, vol2, mask, scattergram)
AVW_Volume *vol1, *vol2, *mask;
AVW_Image *scattergram;

DESCRIPTION

AVW_ScattergramFromVolumes() creates a 2 dimensional scattergram from a pair of compatible volumes and an optional training mask volume which indicates the positions of known classes. Compatible volumes are those with the same Width, Height, and Depth; they must also be of type *AVW_UNSIGNED_CHAR.*

A scattergram created with a training *mask* volume can be further classified with *AVW_ClassifyScattergram().*

If the *mask* volume is NULL the returned scattergram will be of type *AVW_UNSIGNED_INT*. If the *mask* volume is the same size as the input volumes and has known pixel classes indicated with class numbers the returned scattergram will be of type *AVW_UNSIGNED_CHAR* with known pixels set to the class number and all others zero.

The scattergram is a representation of the frequency and occurrences of pairs of values from *vol1* and *vol2* The values 0 to 255 are represented on the vertical and horizontal axes of *vol1* and *vol2* respectively. Values at point x,y in the scattergram are a count of pixels with a value of x from *vol2* and a value of y from *vol1.*

A masked scattergram image is used for fast voxel classification with *AVW_ClassifyFromScattergram().*

If *scattergram* is an *AVW_Image* 256 pixels square and of data type *AVW_UNSIGNED_INT* it will be reused. Otherwise *scattergram* will be destroyed (if necessary) and / or created.

RETURN VALUES

If successful *AVW_Scattergram()* returns an *AVW_Image.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_ScattergramFromVolumes() will fail if one or more of the following are true:

ILLDT
Illegal data type given in type.

ILLVOL
Illegal Volume; vol1 and vol2 are incompatible.

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_ScattergramFromImages(), AVW_ClassifyFromScattergram(), AVW_ClassifyScattergram(), AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_Image

NAME

AVW_SeekImageFile – seeks to a volume and image number in a file

SYNOPSIS

```
#include "AVW_ImageFile.h"

int AVW_SeekImageFile (imgfile, volume, slice)
AVW_ImageFile *imgfile;
int volume;
int slice;
```

DESCRIPTION

AVW_SeekImageFile() positions the *AVW_ImageFile* such that the subsequent call to *AVW_ReadImageFile()* will return the image specified by volume in slice. *AVW_SeekImageFile()* is also used to position the *AVW_ImageFile* so that the next call to *AVW_WriteImageFile()* will write its image to that position in the file. For image formats that do not support three and four dimensions this is a dummy routine provided for consistancy with formats that do.

Volume specifies the volume number at which the file will be positioned. *AVW* numbers volumes in an image file from 0 to (imgfile->Numvols-1).

Slice specifies the image number within the volume of an image file. *AVW* numbers slices in an image file from 0 to (imgfile->Depth-1).

RETURN VALUES

Upon success *AVW_SeekImageFile()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of failure.

ERRORS

AVW_SeekImageFile() will fail if one or more of the following are true:

BDVLNM
Bad Volume Number

BDSLNM
Bad Slice Number

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_WriteImageFile(), AVW_WriteSubVolumeDescription(), AVW_WriteVolume(), AVW_ImageFile

NAME AVW_SetError – Sets the error message and number

SYNOPSIS

```
#include "AVW.h"

void AVW_SetError(number, string)
int number
char *string;
```

DESCRIPTION *AVW_SetError()* sets the error message and number.

SEE ALSO *AVW_Error(), AVW_GetErrorMessage(), AVW_GetErrorNumber(),*

NAME

AVW_SetIdentityMatrix – sets a transformation matrix to the identity matrix

SYNOPSIS

```
#include "AVW.h"

void AVW_SetIdentityMatrix(mat)
AVW_Matrix *mat;
```

DESCRIPTION

AVW_SetIdentityMatrix() sets the 4x4 *AVW_Matrix*, *mat*, to the identity matrix.

SEE ALSO

AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_InvertMatrix(), AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_ScaleMatrix(), AVW_TranslateMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis(), AVW_Matrix

NAME

AVW_SetImage – sets every pixel of an image to a value

SYNOPSIS

```
#include "AVW.h"

int AVW_SetImage(image, value)
AVW_Image *image;
double value;
```

DESCRIPTION

AVW_SetImage() sets all of the pixels in *image* to *value.*

For *AVW_COMPLEX* images the real component of each pixel is set to *value* and the imaginary component is set to zero.

For *AVW_COLOR* images the red, green, and blue components need to be packed into *value.* See *AVW_MAKERGB().*

RETURN VALUES

If successful *AVW_SetImage()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SetImage() will fail if:

ILLDT
Illegal data type.

SEE ALSO

AVW_GetPixel(), AVW_InterpolatedPixel(), AVW_NearestNeighborPixel(), AVW_PutPixel(), AVW_SetVolume(), AVW_ThresholdImage(), AVW_Image, AVW_MAKERGB()

NAME

AVW_SetVolume – sets every voxel of a volume to a value

SYNOPSIS

```
#include "AVW.h"

int AVW_SetVolume(volume, value)
AVW_Volume *volume;
double value;
```

DESCRIPTION

AVW_SetVolume() sets all of the voxels in *volume* to *value.*

For *AVW_COMPLEX* Volumes the real component of each voxel is set to *value* and the imaginary component is set to zero.

For *AVW_COLOR* volumes the red, green, and blue components need to be packed into *value.* See *AVW_MAKERGB().*

RETURN VALUES

If successful *AVW_SetVolume()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SetVolume() will fail if:

ILLDT
Illegal data type.

SEE ALSO

AVW_GetVoxel(), AVW_InterpolatedVoxel(), AVW_NearestNeighborVoxel(), AVW_PutVoxel(), AVW_SetImage(), AVW_ThresholdVolume(), AVW_MAKERGB(), AVW_Volume

NAME

AVW_SetupImageSample – Creates a list of points meeting sample criteria

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_FPointList2 *AVW_SetupImageSample(spec,pointlist)
AVW_SampleSpec spec;
AVW_FPointList2 *pointlist;
```

DESCRIPTION

AVW_SetupImageSample() creates a list of floating-point 2-D coordinates distributed throughout a region of space as defined by the *spec*. The same structure is used to define 3-D samples. *AVW_SetupImageSample* ignores Z extent information in the *spec*

pointlist is provided as a method of re-using an existing *AVW_FPointList2*.

RETURN VALUES

If successful returns a pointer to an *AVW_FPointList2* structure which contains the list of coordinate points.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SetupImageSample() will fail if the following is true:

BADMAL
: Could not allocate memory for the results.

ILLPAR
: Sample or interpolation type or in the *spec* is not recognized.

SEE ALSO

AVW_StepSearchExtreme2D(), AVW_BoundedStepSearchExtreme2D(), AVW_SampleSpec.

NAME

AVW_SetupVolumeSample – Creates a list of points meeting sample criteria

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_FPointList3 *AVW_SetupVolumeSample(spec,pointlist)
AVW_SampleSpec spec;
AVW_FPointList3 *pointlist;
```

DESCRIPTION

AVW_SetupVolumeSample() creates a list of floating-point 3-D coordinates distributed throughout a region of space as defined by the *spec*.

pointlist is provided as a method of re-using an existing *AVW_FPointList3*.

RETURN VALUES

If successful returns a pointer to an *AVW_FPointList3* structure which contains the list of coordinate points.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_SetupVolumeSample() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

ILLPAR
Sample or interpolation type or in the *spec* is not recognized.

SEE ALSO

AVW_StepSearchExtreme(), AVW_BoundedStepSearchExtreme(), AVW_SampleSpec.

NAME

AVW_ShiftImage – shifts an image right / left / up / down

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ShiftImage(input_image, right_shift,
        up_shift, wrap, out_image)
AVW_Image *input_image;
int right_shift, up_shift, wrap;
AVW_Image *output_image;
```

DESCRIPTION

AVW_ShiftImage() shifts or translates *input_image* in the vertical or horizontal directions and places the result in *out_image.*

Right_shift specifies the number of pixels the image is to be shifted to the right, negative values cause a shift to the left.

Up_shift specifies the number of pixels the image is to be shifted up, negative values cause a downward shift.

A nonzero *wrap* value specifies that all pixels shifted off of the image are wrapped around to the opposite side of the image, otherwise the pixels are discarded and the open areas are zero filled.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ShiftImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ShiftImage() will fail if:

BADMAL
: Could not allocate enough memory for results.

SEE ALSO

AVW_FlipImage(), AVW_Rotate90Image(), AVW_PadImage(), AVW_TransformImage(), AVW_Image

NAME

AVW_ShiftPointList2 – shifts a point list right / left / up / down

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_ShiftPointList2(list, right_shift,
        up_shift, out_list)
AVW_PointList2 *list;
int right_shift, up_shift;
AVW_PointList2 *out_list;
```

DESCRIPTION

AVW_ShiftPointList2() shifts or translates *list* in the vertical or horizontal directions and places the result in *out_list*.

Right_shift specifies the number of pixels the point list is to be shifted to the right, negative values cause a shift to the left.

Up_shift specifies the number of pixels the point list is to be shifted up, negative values cause an downward shift.

Out_list is provided as a method of reusing an existing *AVW_PointList2*. If not reuseable *out_list* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ShiftPointList2()* returns an *AVW_PointList2*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ShiftPointList2() will fail if:

BADMAL
Could not allocate enough memory for results.

SEE ALSO

AVW_AddPoint2(), AVW_ClipPointList2(), AVW_CopyPointList2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_DrawPointList2(), AVW_EditPointList2(), AVW_FillPointList2(), AVW_GetPoint2(), AVW_RemovePoint2(), AVW_RotatePointList2(), AVW_ScalePointList2(), AVW_TransformPoint2(), AVW_PointList2

NAME

AVW_ShowImage – quick and dirty display

SYNOPSIS

```
#include "AVW.h"

int AVW_ShowImage(image)
AVW_Image *image;
```

DESCRIPTION

AVW_ShowImage() writes the *AVW_Image* to a temporary *AVW_ImageFile* in the $TMP (/tmp by default) directory in the *AVW_ImageFile* format. It then calls *ShowServer* to display the results on the users display.

On UNIX systems, *ShowServer* will continue to run and display any other images which are passed to *AVW_ShowImage().* Any keystroke or mouse click in an image display window will destroy that window. Once the program which called AVW_ShowImage() has terminated, any keystroke or mouse click terminates all the windows.

On PC systems, a different *ShowServer* is started for image displayed. Each window will need to be dismissed seperately.

RETURN VALUES

Upon success *AVW_ShowImage()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned, and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

Notes

The program, *ShowServer* must be in the users *path* before *AVW_ShowImage()* is called.

Example:
set path = ($AVW/$TARGET/bin ${path}) (UNIX)
set path=%AVW%\%TARGET%\lib;%AVW%\%TARGET%\bin;%path% (PC)

SEE ALSO

AVW_CreateImageFile(), AVW_CloseImageFile(), AVW_ReadImageFile(), AVW_OpenImageFile(), AVW_WriteImageFile(), AVW_Image,

NAME

AVW_SigmaFilterImage – performs a 2D Sigma filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_SigmaFilterImage(in_image, extents, sigma, out_image)
AVW_Image *in_image;
int extents[2];
int sigma;
AVW_Image *out_image;
```

DESCRIPTION

AVW_SigmaFilterImage() performs a Sigma filter transformation on *in_image. Extents[0]* and *extents[1]* specify the x and y sizes respectively of the filter.

Sigma specifies the sigma value for the sigma filter.

The sigma filter uses a local smoothing scheme. This filter smooths noise, preserves edges and can leave thin lines untouched. For each pixel in the input image, the mean value of a set of pixels within 2* *sigma* is calculated. Only pixels within the range specified by *extents* are considered in this calculation. If too few pixels within the extents lie with in the 2*sigma value, then the pixel is left unchanged; otherwise, the calculated mean is assigned as a new pixel value.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_SigmaFilterImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_AnisotropicAffineImage(), AVW_AnisotropicDiffusionImage(), AVW_LowpassFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterVolume(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_Image

NAME

AVW_SigmaFilterVolume – performs a 3D Sigma filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_SigmaFilterVolume(in_volume, extents, sigma, out_volume)
AVW_Volume *in_volume;
int extents[3];
int sigma;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_SigmaFilterVolume() performs a Sigma filter transformation on *in_volume. Extents[0]*, *extents[1]*, and, *extents[2]* specify the x, y, and z sizes respectively of the filter.

Sigma specifies the sigma value for the sigma filter.

The sigma filter uses a local smoothing scheme. This filter smooths noise, preserves edges and can leave thin lines untouched. For each voxel in the input volume, the mean value of a set of voxels within 2* *sigma* is calculated. Only voxels within the range specified by *extents* are considered in this calculation. If too few voxels within the extents lie with in the 2*sigma value, then the voxel is left unchanged; otherwise, the calculated mean is assigned as a new voxel value.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_SigmaFilterVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SigmaFilterImage(), AVW_SobelFilterEnhanceVolume(), AVW_SobelFilterVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_SincInterpolatedPixel – returns pixel value at a floating point location

SYNOPSIS

```
#include "AVW.h"

double AVW_SincInterpolatedPixel(image, point)
AVW_Image *image;
AVW_FPoint2 *point;
```

DESCRIPTION

Given a floating point location *point* within *image, AVW_SincInterpolatedPixel()* returns the calculated pixel value at the floating point location.

Uses a windows sinc algorithm to estimate the pixel value at the floating point location.

Points outside the image will return a value of 0.0.

SEE ALSO

AVW_GetPixel(), AVW_GetErrorNumner(), AVW_InterpolatedVoxel(), AVW_NearestNeighborPixel(), AVW_InterpolatedPixel(), AVW_CubicSplineInterpolatedPixel(), AVW_Image, AVW_FPoint2

NAME

AVW_SincInterpolatedVoxel – returns voxel value at a floating point location

SYNOPSIS

#include "AVW.h"

double AVW_SincInterpolatedVoxel(volume, point)
AVW_Volume *volume;
AVW_FPoint3 *point;

DESCRIPTION

Given a floating point location *point* within *volume, AVW_SincInterpolatedVoxel()* returns the calculated voxel value at the floating point location.

Uses a windowed sinc algorithm to estimate the voxel value at the floating point location.

Points outside the volume will return a value of 0.0.

SEE ALSO

AVW_GetVoxel(), AVW_GetErrorNumner(), AVW_InterpolatedPixel(), AVW_NearestNeighborVoxel(), AVW_InterpolatedVoxel(), AVW_CubicSplineInterpolatedVoxel(), AVW_FPoint3, AVW_Volume

NAME

AVW_SincWindowLimit – sets sinc window size

SYNOPSIS

```
#include    "AVW.h"

int AVW_SincWindowLimit(windowsize)
int windowsize;
```

DESCRIPTION

AVW_SincWindowLimit() Changes the size of the window used during sinc interpolation (see *AVW_SincInterpolatedPixel* and *AVW_SincInterpolatedVoxel*). *windowsize* is a positive value indicating the requested size of the sinc kernel. The kernel extends *windowsize* to each side of the interpolated position.

The return value is the new window size.

RETURN VALUES

A call to *AVW_SincWindowLimit()* returns the new value of the window used for sinc interpolation. The return value may be less than the requested value.

SEE ALSO

AVW_SincInterpolatedPixel(), AVW_SincInterpolatedVoxel

NAME

AVW_SliceVolume – extracts a contour surface from a volume

SYNOPSIS

```
#include "AVW_Model.h"

AVW_ContourSurface *AVW_SliceVolume(rp_param, oldSurface)
AVW_RPParam *rp_param;
AVW_ContourSurface *oldSurface;
```

DESCRIPTION

AVW_SliceVolume() generates a set of ribbon contours based on the parameters set in *rp_param* and generates a *AVW_ContourSurface* structure. *oldSurface* is provided as a method of reusing an existing *AVW_ContourSurface*. If reuse is not possible, *oldSurface* will be reallocated.

RETURN VALUES

If successful *AVW_SliceVolume()* returns an *AVW_ContourSurface*. *NULL* is returned should *AVW_SliceVolume()* fail.

ERRORS

Errors may occur for the following reasons:

ILLPAR
Illegal Parameter. Typically a missing or improperly set option in *rp_param*.

BADMAL
Bad Malloc. Memory allocation error.

BDOPEN
Open Failed. Unable to open file for writing.

BDWRTE
Write Failed. Unable to write to specified file

ILLVOL
Illegal Volume. Unable to perform operation on spcified volume. Typically results from trying to extract contours from an unsupported data type.

ILLDT
Illegal Datatype. Unable to perform operation on the given data type.

ILLMSK
Illegal Mask. Typically indicates a mask value that is out of range for the given volume.

SEE ALSO

AVW_InitializeRPParam(), AVW_DestroyContourSurface(), AVW_DestroyRPParam(), AVW_SaveContourSurface(), AVW_ContourSurface, AVW_RPParam

NAME

AVW_SmoothHistogram – smoothes the values of a histogram

SYNOPSIS

```
#include       "AVW.h"

int AVW_SmoothHistogram(histogram, half_width)
AVW_Histogram *histogram;
int half_width;
```

DESCRIPTION

AVW_SmoothHistogram() smoothes each value of the *histogram* using a weighted average of the neighboring bins. *Half_width* specifies how many bins to the left and right of the center bin will be used in calculating the smoothed value. The weight is inversely propotional to disance from the center bin.

RETURN VALUES

If successful, *AVW_SmoothHistogram()* returns *AVW_SUCCESS*. On failure, it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure. To insure valid results *AVW_ErrorNumber* should be checked after calling this function.

ERRORS

AVW_SmoothHistogram() will fail if one or more of the following are true:

ILLHIS
: Illegal histogram, histogram is NULL.

SEE ALSO

AVW_CreateHistogram(), AVW_GetHistogramModeValue(), AVW_GetHistogramMedianValue(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_Histogram

NAME

AVW_SobelFilterEnhanceImage − enhances an image by Sobel filtering

SYNOPSIS

#include "AVW_Filter.h"

```
AVW_Image *AVW_SobelFilterEnhanceImage(in_image, extents, out_image)
AVW_Image *in_image;
int extents[2];
AVW_Image *out_image;
```

DESCRIPTION

AVW_SobelFilterEnhanceImage() performs a Sobel filter transformation which is used for edge detection on *in_image.* The returned image *out_image,* consists of the input image added to the Sobel filtered input image.

Extents[0] and *extents[1],* specify the x and y sizes respectively of the filter.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_SobelFilterEnhanceImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_AnisotropicAffineImage(), AVW_AnisotropicDiffusionImage(), AVW_LowpassFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceVolume(), AVW_UnsharpFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_Image

NAME

AVW_SobelFilterEnhanceVolume – enhances a volume by Sobel filtering

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_SobelFilterEnhanceVolume(in_volume, extents, out_volume)
AVW_Volume *in_volume;
int extents[3];
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_SobelFilterEnhanceVolume() performs a Sobel filter transformation which is used for edge detection on *in_volume.* The returned volume, *out_volume,* consists of the input volume added to the Sobel filtered input volume.

Extents[0], extents[1], and *extents[2]* specify the x, y, and z sizes respectively of the filter.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_SobelFilterEnhanceVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterEnhanceImage(), AVW_SobelFilterVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_SobelFilterImage – performs a 2D Sobel filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_SobelFilterImage(in_image, extents, out_image)
AVW_Image *in_image;
int extents[2];
AVW_Image *out_image;
```

DESCRIPTION

AVW_SobelFilterImage() performs a Sobel filter transformation which is used for edge detection on *in_image.*

Extents[0] and *extents[1],* specify the x and y sizes respectively of the filter.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_SobelFilterImage()* returns an *AVW_Image.* On failure all it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_AnisotropicAffineImage(), AVW_AnisoTropicDiffusionImage(), AVW_LowpassFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_SobelFilterVolume(), AVW_UnsharpFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_Image

NAME

AVW_SobelFilterVolume – performs a 3D Sobel filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_SobelFilterVolume(in_volume, extents, out_volume)
AVW_Volume *in_volume;
int extents[3];
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_SobelFilterVolume() performs a Sobel filter transformation which is used for edge detection on *in_volume.*

Extents[0], *extents[1]*, and *extents[2]* specify the x, y, and z sizes respectively of the filter.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_SobelFilterVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_SobelFilterImage(), AVW_UnsharpFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_StepSearchExtreme – Searches for best registration of two volume images.

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_Matrix *AVW_StepSearchExtreme(dirflag,base,match,points,steps,func,interpolation,matrix)
int dirflag,interpolation;
AVW_Volume *base,*match;
AVW_FPointList3 *points;
AVW_Matrix *matrix;
AVW_StepSearchSpec *steps;
int func;
```

DESCRIPTION

AVW_StepSearchExtreme() Performs a stepwise search of 6-DOF physical registration space to find the nearest extreme of a voxel statistic function relating two *AVW_Volumes. AVW_StepSearchExtreme* returns the *AVW_Matrix* which transforms the match volume into the space of the base volume at the extreme.

dirflag determines whether maxima or minima are searched for. Defined values are *AVW_MAXIMUM* and *AVW_MINIMUM.*

base and *match* are the *AVW_Volume* s to be registered.

points is an *AVW_FPointList3* containing a list of coordinate points in the match image to be used as the sample voxels for the registration. *points* is usually created by a call to *AVW_SetupVolumeSample*

steps contains the specification of the step search, primarily specific step sizes for each of the 6 degrees of freedom. Bounding information in *steps* is ignored by *AVW_StepSearchExtreme.*

func specifies the statistical measure to be used. Defined values are *AVW_NMI.*

interpolation specifies the intepolation type to be used in the search, and may be any of the defined AVW interpolation types.

matrix is taken as the starting orientation for the search. If *Matrix* is NULL, a new identity matrix is created, and used as the starting orientation.

RETURN VALUES

If successful returns an *AVW_Matrix* which transforms the match volume into the space of the base volume at the extreme.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_StepSearchExtreme() will fail if the following is true:

BADMAL
Could not allocate memory for internal structures.

ILLPAR
Sample or interpolation type is not recognized.

SEE ALSO *AVW_SetupVolumeSample() AVW_BoundedStepSearchExtreme() AVW_SampleSpec AVW_StepSearchSpec*

NAME

AVW_StepSearchExtreme2D – Searches for best registration of two images.

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

AVW_Matrix *AVW_StepSearchExtreme2D(dirflag,base,match,points,steps,func,interpolation,matrix)
int dirflag,interpolation;
AVW_Image *base,*match;
AVW_FPointList2 *points;
AVW_Matrix *matrix;
AVW_StepSearchSpec *steps;
int func;
```

DESCRIPTION

AVW_StepSearchExtreme2D() Performs a stepwise search of 3-DOF physical registration space to find the nearest extreme of a voxel statistic function relating two *AVW_Images. AVW_StepSearchExtreme2D* returns the *AVW_Matrix* which transforms the match image into the space of the base image at the extreme.

dirflag determines whether maxima or minima are searched for. Defined values are *AVW_MAXIMUM* and *AVW_MINIMUM.*

base and *match* are the *AVW_Image* s to be registered.

points is an *AVW_FPointList2* containing a list of coordinate points in the match image to be used as the sample voxels for the registration. *points* is usually created by a call to *AVW_SetupImageSample*

steps contains the specification of the step search, primarily specific step sizes for each of the 3 degrees of freedom. Bounding information in *steps* is ignored by *AVW_StepSearchExtreme2D.*

func specifies the statistical measure to be used. Defined values are *AVW_NMI.*

interpolation specifies the intepolation type to be used in the search, and may be any of the defined AVW interpolation types.

matrix is taken as the starting orientation for the search. If *Matrix* is NULL, a new identity matrix is created, and used as the starting orientation.

RETURN VALUES

If successful returns an *AVW_Matrix* which transforms the match image into the space of the base image at the extreme.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_StepSearchExtreme2D() will fail if the following is true:

BADMAL
Could not allocate memory for internal structures.

ILLPAR
Sample or interpolation type is not recognized.

SEE ALSO *AVW_SetupImageSample() AVW_BoundedStepSearchExtreme2D() AVW_SampleSpec AVW_StepSearchSpec*

NAME

AVW_SumIntensityStats – sums intensity statistics

SYNOPSIS

#include "AVW_Measure.h"

```
void AVW_SumIntensityStats( stats, statsum)
AVW_IntensityStats *stats, *statsum;
```

DESCRIPTION

AVW_SumIntensityStats() adds the statistics from *stats* to *statsum*. Elements such as Mean, Standard Deviation, and Variance are recomputed based on the summed statistics.

SEE ALSO

AVW_ComputeVolumeIntensityStats() AVW_ComputeImageIntensityStats(), AVW_ResetIntensityStatistics(), AVW_IntensityStats

NAME

AVW_SwapBlock – swaps bytes in a block

SYNOPSIS

```
#include    <stdio.h>
#include    "AVW.h"

AVW_SwapBlock(pntr, bytes)
void *pntr;
int bytes;
```

DESCRIPTION

AVW_SwapBlock() swaps bytes in place over a specified number of *bytes*.

SEE ALSO

AVW_QuadSwapImage(), AVW_READSWAP(), AVW_ReverseBits(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapLong(), AVW_SwapShort(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME

AVW_SwapDouble – swaps the bytes of a double

SYNOPSIS

```
#include      <stdio.h>
#include      "AVW.h"

void AVW_SwapDouble(ptr)
double *ptr;
```

DESCRIPTION

AVW_SwapDouble(), swaps the bytes of a double.

SEE ALSO

AVW_QuadSwapImage(), AVW_READSWAP(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapLong(), AVW_SwapShort(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME

AVW_SwapFloat – swaps the bytes of a float

SYNOPSIS

```
#include	<stdio.h>
#include	"AVW.h"

void AVW_SwapFloat(ptr)
float *ptr;
```

DESCRIPTION

AVW_SwapFloat() swaps the bytes of a float.

SEE ALSO

AVW_QuadSwapImage(), AVW_READSWAP(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapLong(), AVW_SwapShort(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME AVW_SwapImage – swaps the bytes of an image

SYNOPSIS

```
#include <stdio.h>
#include "AVW.h"

AVW_SwapImage(image)
AVW_Image *image;
```

DESCRIPTION *AVW_SwapImage()* swap the bytes of an image.

SEE ALSO *AVW_QuadSwapImage(), AVW_READSWAP(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapInt(), AVW_SwapShort(), AVW_SwapLong(), AVW_WRITESWAP(), fread(), fwrite(), swab()*

NAME

AVW_SwapInt – swaps bytes of an integer

SYNOPSIS

```
#include      <stdio.h>
#include      "AVW.h"

AVW_SwapInt(pntr)
int *pntr;
```

DESCRIPTION

AVW_SwapInt() swaps the bytes of an integer.

SEE ALSO

AVW_QuadSwapImage(), AVW_READSWAP(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapLong(), AVW_SwapShort(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME

AVW_SwapLong – swaps bytes of a long integer

SYNOPSIS

```
#include      <stdio.h>
#include      "AVW.h"

AVW_SwapLong(pntr)
long int *pntr;
```

DESCRIPTION

AVW_SwapLong swaps the bytes of a long integer.

SEE ALSO

AVW_QuadSwapImage(), AVW_READSWAP(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapShort(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME

AVW_SwapShort – swaps the bytes of a short integer

SYNOPSIS

```
#include	<stdio.h>
#include	"AVW.h"

AVW_SwapShort(pntr)
short *pntr;
```

DESCRIPTION

AVW_SwapShort() swap the bytes of a short integer.

SEE ALSO

AVW_QuadSwapImage(), AVW_READSWAP(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapLong(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME

AVW_TableImage – applies a lookup table to an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_TableImage(in_image, table, out_image)
AVW_Image *in_image;
unsigned long *table;
AVW_Image *out_image;
```

DESCRIPTION

AVW_TableImage() uses the values of *in_image* as indicies in the *table* array to get the values of *out_image.*

    out_image[i] = table[in_image[i]];

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_TableImage()* returns an *AVW_Image.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_TableImage() will fail if the following is true:

NOTSUP
    Data type is not supported.

SEE ALSO

AVW_IntensityClipImage(), AVW_IntensityScaleImage(), AVW_InvertImage(), AVW_MakeMonoImage(), AVW_TableVolume(), AVW_Image

NAME

AVW_TableVolume – applies a lookup table to a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_TableVolume(in_volume, table, out_volume)
AVW_Volume *in_volume;
unsigned long *table;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_TableVolume() uses the values of *in_volume* as indicies in the *table* array to get the values of *out_volume.*

```
out_volume[i] = table[in_volume[i]];
```

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_TableVolume()* returns an *AVW_Volume.* On failure *NULL* is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_TableVolume() will fail if the following is true:

NOTSUP
: Data type is not supported.

SEE ALSO

AVW_IntensityClipVolume(), AVW_IntensityScaleVolume(), AVW_InvertVolume(), AVW_TableImage(), AVW_Volume

NAME

AVW_Thin2D – performs 2D thinning on an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_Thin2D(in_image, iterations, out_image)
AVW_Image *in_image;
int iterations;
AVW_Image *out_image;
```

DESCRIPTION

AVW_Thin2D() performs 2D thinning, using template matching, on *in_image.* The image is thinned *iterations* times or until no more changes can be made. If *iterations* is less than or equal to zero the image will be thinned to its skeleton. The thinned image is returned in *out_image.*

In_image has to be a binary valued, i.e. ones and zeroes. *In_image,* and *out_image* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_Thin2D()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_Thin2D() will fail if:

ILLDT
Data type is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_Thin3D(), AVW_Image

NAME

AVW_Thin3D – performs 3D thinning on a volume

SYNOPSIS

```
#include    "AVW.h"

AVW_Volume *AVW_Thin3D(in_volume, iterations, out_volume)
AVW_Volume *in_volume;
int iterations;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_Thin3D() performs 3D thinning, using the method described in:

S. Lobregt, P.W. Verbeek, and F.C.A. Groen, "Three-dimensional skeletonization: principle and algorithm", IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-2, no.1, pp. 75-77, Jan, 1980.

The volume is thinned *iterations* times or until no more changes can be made. If *iterations* is less than or equal to zero, the volume will be thinned to its skeleton. The thinned volume is returned in *out_volume.*

Basically, the connectivity of a 3X3X3 region surrounding a candidate voxel for deletion is evaluated with and without the candidate voxel. The candidate voxels are edge voxels found by scanning the volume from different directions. If the connectivity does not change, the voxel is deleted. If the connectivity changes the voxel is not deleted.

This algorithm requires solid objects, i.e. 2D holes must be filled (See *AVW_FillHolesImage()).*

In_Volume has to be a binary valued, i.e. ones and zeroes. *In_Volume,* and *out_volume* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_Thin3D()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_Thin3D() will fail if:

ILLDT
    Data type is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_Thin2D(), AVW_Volume

NAME

AVW_ThresholdImage – thresholds an image

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_ThresholdImage(in_image, threshold_max,
        threshold_min, out_image)
AVW_Image *in_image;
double threshold_max, threshold_min;
AVW_Image *out_image;
```

DESCRIPTION

AVW_ThresholdImage() thresholds an image, *in_image*, by setting all of the pixels greater than or equal to the *threshold_min* and less than or equal to *threshold_max* to 1 and the rest of the pixels to 0. The returned *AVW_Image* is of the data type *AVW_UNSIGNED_CHAR*.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reuseable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ThresholdImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ThresholdImage() will fail if the following is true:

ILLDT
Illegal data type.

SEE ALSO

AVW_ThresholdVolume(), AVW_Image

NAME

AVW_ThresholdVolume – thresholds a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_ThresholdVolume(in_volume, threshold_max,
        threshold_min, out_volume)
AVW_Volume *in_volume;
double threshold_max, threshold_min;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_ThresholdVolume() thresholds a volume, *in_volume*, by setting all of the voxels greater than or equal to the *threshold_min* and less than or equal to *threshold_max* to 1 and the rest of the voxels to 0. The returned *AVW_Volume* is of the data type *AVW_UNSIGNED_CHAR*.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_ThresholdVolume()* returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ThresholdVolume() will fail if the following is true:

ILLDT
Illegal data type.

SEE ALSO

AVW_ThresholdImage(), AVW_Volume

NAME

AVW_TileVolume – generates an AVW_TiledSurface

SYNOPSIS

```
#include "AVW_Model.h"

AVW_TiledSurface *AVW_TileVolume(tile_param, surface)
AVW_TileParameters *tile_param;
AVW_TiledSurface *surface;
```

DESCRIPTION

AVW_TileVolume() generates an *AVW_TiledSurface* based on the parameters set in *tile_param*.

RETURN VALUES

If successful *AVW_TileVolume()* returns *AVW_TiledSurface*. *NULL* is returned if a *AVW_TiledSurface* cannot be created.

ERRORS

Errors may occur for the following reasons:

ILLPAR
Illegal Parameter. Typically a missing or improperly set option in *tile_param*.

BADMAL
Bad Malloc. Memory allocation error.

SEE ALSO

AVW_InitializeTileParameters(), AVW_LoadTiledSurface(), AVW_SaveTiledSurface.3(), AVW_DrawTiledSurface(), AVW_DestroyTiledSurface(), AVW_DestroyTileParameters(), AVW_VolumeToSLC(), AVW_TiledSurface, AVW_TileParameters

NAME

AVW_TransformFPoint2 – applies a matrix to a point

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformFPoint2(point2, matrix, in_image, out_image)
AVW_FPoint2 *point2;
AVW_Matrix *matrix;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_TransformFPoint2() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_FPoint2, point2.*

Point coordinates are often specified with (0,0) in the corner of the image. Matrices normally specify a rotation around the center of an image. The *in_image* and *out_image* parameters are optional. If specified the dimension of the images are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddFPoint2(), AVW_GetFPoint2(), AVW_TransformFPoint3(), AVW_TransformFPointList2(), AVW_TransformIPoint2(), AVW_TransformPoint2(), AVW_FPoint2, AVW_Matrix, AVW_Image

NAME

AVW_TransformFPoint3 – applies a matrix to a point

SYNOPSIS

#include "AVW.h"

```
int AVW_TransformFPoint3(point3, matrix, in_volume, out_volume)
AVW_FPoint3 *point3;
AVW_Matrix *matrix;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_TransformFPoint3() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_FPoint3, point3.*

Point coordinates are often specified with (0,0) in the corner of the volume. Matrices normally specify a rotation around the center of an volume. The *in_volume* and *out_volume* parameters are optional. If specified the dimension of the volumes are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddFPoint3(), AVW_GetFPoint3(), AVW_TransformFPoint2(), AVW_TransformFPointList3(), AVW_TransformIPoint3(), AVW_TransformPoint3(), AVW_FPoint3, AVW_Matrix, AVW_Image

NAME

AVW_TransformFPointList2 – applies a matrix to a pointlist

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformFPointList2(plist, matrix, in_image, out_image)
AVW_FPointList2 *plist;
AVW_Matrix *matrix;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_TransformFPointList2() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_FPointList2, plist.*

Point coordinates are often specified with (0,0) in the corner of the image. Matrices normally specify a rotation around the center of an image. The *in_image* and *out_image* parameters are optional. If specified the dimension of the images are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddFPointList2(), AVW_GetFPointList2(), AVW_TransformFPointList3(), AVW_TransformFPoint2(), AVW_TransformIPointList2(), AVW_TransformPointList2(), AVW_FPointList2, AVW_Matrix, AVW_Image

NAME

AVW_TransformFPointList3 – applies a matrix to a pointlist

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformFPointList3(plist, matrix, in_volume, out_volume)
AVW_FPointList3 *plist;
AVW_Matrix *matrix;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_TransformFPointList3() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_FPointList3, plist.*

Point coordinates are often specified with (0,0) in the corner of the volume. Matrices normally specify a rotation around the center of an volume. The *in_volume* and *out_volume* parameters are optional. If specified the dimension of the volumes are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddFPointList3(), AVW_GetFPointList3(), AVW_TransformFPointList2(), AVW_TransformFPoint3(), AVW_TransformIPointList3(), AVW_TransformPointList3(), AVW_FPointList3, AVW_Matrix, AVW_Image

NAME

AVW_TransformIPoint2 – applies a matrix to a point

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformIPoint2(point2, matrix, in_image, out_image)
AVW_IPoint2 *point2;
AVW_Matrix *matrix;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_TransformIPoint2() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_IPoint2, point2.*

Point coordinates are often specified with (0,0) in the corner of the image. Matrices normally specify a rotation around the center of an image. The *in_image* and *out_image* parameters are optional. If specified the dimension of the images are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddIPoint2(), AVW_GetIPoint2(), AVW_TransformFPoint2(), AVW_TransformIPoint3(), AVW_TransformIPointList2(), AVW_TransformPoint2(), AVW_IPoint2, AVW_Matrix, AVW_Image

NAME

AVW_TransformIPoint3 – applies a matrix to a point

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformIPoint3(point3, matrix, in_volume, out_volume)
AVW_IPoint3 *point3;
AVW_Matrix *matrix;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_TransformIPoint3() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_IPoint3, point3.*

Point coordinates are often specified with (0,0) in the corner of the volume. Matrices normally specify a rotation around the center of an volume. The *in_volume* and *out_volume* parameters are optional. If specified the dimension of the volumes are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddIPoint3(), AVW_GetIPoint3(), AVW_TransformFPoint3(), AVW_TransformIPoint2(), AVW_TransformIPointlist3(), AVW_TransformPoint3(), AVW_IPoint3, AVW_Matrix, AVW_Volume

NAME

AVW_TransformIPointList2 – applies a matrix to a pointlist

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformIPointList2(plist, matrix, in_image, out_image)
AVW_IPointList2 *plist;
AVW_Matrix *matrix;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_TransformIPointList2() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_IPointList2, plist.*

Point coordinates are often specified with (0,0) in the corner of the image. Matrices normally specify a rotation around the center of an image. The *in_image* and *out_image* parameters are optional. If specified the dimension of the images are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddIPointList2(), AVW_GetIPointList2(), AVW_TransformFPointList2(), AVW_TransformIPointList3(), AVW_TransformIPoint2(), AVW_TransformPointList2(), AVW_IPointList2, AVW_Matrix, AVW_Image

NAME

AVW_TransformIPointList3 – applies a matrix to a pointlist

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformIPointList3(plist, matrix, in_volume, out_volume)
AVW_IPointList3 *plist;
AVW_Matrix *matrix;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_TransformIPointList3() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_IPointList3, plist.*

Point coordinates are often specified with (0,0) in the corner of the volume. Matrices normally specify a rotation around the center of an volume. The *in_volume* and *out_volume* parameters are optional. If specified the dimension of the volumes are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddIPointList3(), AVW_GetIPointList3(), AVW_TransformFPointList3(), AVW_TransformIPoint3(), AVW_TransformIPointList2(), AVW_TransformPointList3(), AVW_IPointList3, AVW_Matrix, AVW_Volume

NAME

AVW_TransformImage – applies a matrix to an image

SYNOPSIS

#include "AVW.h"

AVW_Image ∗AVW_TransformImage(in_image, matrix, interpolate_type, trans_image)
AVW_Image ∗in_image;
AVW_Matrix ∗matrix;
int interpolate_type;
AVW_Image ∗trans_image;

DESCRIPTION

AVW_TransformImage() applies the 4x4 transformation *AVW_Matrix, matrix,* to *in_image.*

Interpolate_type determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

Trans_image is used to return the results of the transformation. If *NULL* is passed *trans_image* will be allocated with the same dimensions as *in_image.* The results of the transformation can be placed in a larger or smaller image. In this case *AVW_CreateImage()* may be called to create an image of desired dimensions. This image is then passed to the routine and the results are returned in it. *Trans_image* will not be reallocated if it does not match the size of *in_image* as in other functions.

RETURN VALUES

If successful *AVW_TransformImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_TransformImage() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

CFLSZ
Input and output data types conflict.

SEE ALSO

AVW_FlipImage(), AVW_Rotate90Image(), AVW_ShiftImage(), AVW_TransformVolume(), AVW_Image, AVW_Matrix

NAME

AVW_TransformPoint2 – applies a matrix to a point

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformPoint2(point2, matrix, in_image, out_image)
AVW_Point2 *point2;
AVW_Matrix *matrix;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_TransformPoint2() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_Point2, point2.*

Point coordinates are often specified with (0,0) in the corner of the image. Matrices normally specify a rotation around the center of an image. The *in_image* and *out_image* parameters are optional. If specified the dimension of the images are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddPoint2(), AVW_GetPoint2(), AVW_TransformFPoint2(), AVW_TransformIPoint2(), AVW_TransformPoint3(), AVW_TransformPointList2(), AVW_Point2, AVW_Matrix, AVW_Image

NAME

AVW_TransformPoint3 – applies a matrix to a point

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformPoint3(point3, matrix, in_volume, out_volume)
AVW_Point3 *point3;
AVW_Matrix *matrix;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_TransformPoint3() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_Point3, point3.*

Point coordinates are often specified with (0,0) in the corner of the volume. Matrices normally specify a rotation around the center of an volume. The *in_volume* and *out_volume* parameters are optional. If specified the dimension of the volumes are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddPoint3(), AVW_GetPoint3(), AVW_TransformFPoint3(), AVW_TransformIPoint3(), AVW_TransformPoint2(), AVW_TransformPointList3(), AVW_Point3, AVW_Matrix, AVW_Volume

NAME

AVW_TransformPointList2 – applies a matrix to a pointlist

SYNOPSIS

```
#include "AVW.h"

int AVW_TransformPointList2(plist, matrix, in_image, out_image)
AVW_PointList2 *plist;
AVW_Matrix *matrix;
AVW_Image *in_image;
AVW_Image *out_image;
```

DESCRIPTION

AVW_TransformPointList2() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_PointList2, plist.*

Point coordinates are often specified with (0,0) in the corner of the image. Matrices normally specify a rotation around the center of an image. The *in_image* and *out_image* parameters are optional. If specified the dimension of the images are used to locate the center of the input space and output space.

SEE ALSO

AVW_AddPointList2(), AVW_GetPointList2(), AVW_TransformFPointList2(), AVW_TransformIPointList2(), AVW_TransformPointList3(), AVW_TransformPoint2(), AVW_PointList2, AVW_Matrix, AVW_Image

NAME

AVW_TransformPointList3 – applies a matrix to a pointlist

SYNOPSIS

#include "AVW.h"

```
int AVW_TransformPointList3(plist, matrix, in_volume, out_volume)
AVW_PointList3 *plist;
AVW_Matrix *matrix;
AVW_Volume *in_volume;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_TransformPointList3() applies the 4x4 transformation *AVW_Matrix, matrix,* to the *AVW_PointList3, plist.*

Point coordinates are often specified with (0,0) in the corner of the volume. Matrices normally specify a rotation around the center of an volume. The *in_volume* and *out_volume* parameters are optional. If specified the dimension of the volumes are used to locate the center of the input space and output space.

SEE ALSO

AVW_TransformFPointList3(), AVW_TransformIPointList3(), AVW_TransformPoint3(), AVW_TransformPointList2(), AVW_PointList3, AVW_Matrix, AVW_Volume

NAME

AVW_TransformVolume – applies a matrix to a volume

SYNOPSIS

```
#include "AVW.h"

AVW_Volume *AVW_TransformVolume(in_volume, matrix, interpolate_type, trans_volume)
AVW_Volume *in_volume;
AVW_Matrix *matrix;
int interpolate_type;
AVW_Volume *trans_volume;
```

DESCRIPTION

AVW_TransformVolume() applies the 4x4 transformation *AVW_Matrix, matrix,* to *in_volume.*

Interpolate_type determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

Trans_volume is used to return the results of the transformation. If *NULL* is passed *trans_volume* will be allocated with the same dimensions as *in_volume.* The results of the transformation can be placed in a larger or smaller volume. In this case *AVW_CreateVolume()* may be called to create a volume of desired dimensions. This volume is then passed to the routine and the results are returned in it. *Trans_volume* will not be reallocated if it does not match the size of *in_volume* as in other functions.

RETURN VALUES

If successful *AVW_TransformVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_TransformVolume() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

CFLSZ
Input and output data types conflict.

SEE ALSO

AVW_FlipVolume(), AVW_TransformImage(), AVW_TransformVolumeSliceBySlice(), AVW_Matrix, AVW_Volume

NAME

AVW_TransformVolumeSliceBySlice – applies a matrix to a volume slice by slice

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_TransformVolumeSliceBySlice(in_volume, matrix,
        interpolate_type, nslices, slice, trans_image)
AVW_Volume *in_volume;
AVW_Matrix *matrix;
int interpolate_type;
int nslices,
int slice;
AVW_Image *trans_image;
```

DESCRIPTION

AVW_TransformVolumeSliceBySlice() applies the 4x4 transformation *AVW_Matrix, matrix,* to *in_volume.*

Interpolate_type determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

Nslices specifies the number of slices in the reformated volume.

Slice specifies which slice to reformat in the reformated volume.

Trans_image is used to return the results of the transformation.

If *NULL* is passed *trans_image* will be allocated with the same dimensions as *in_volume* (Width & Height dimension only). The results of the transformation can be placed in a larger or smaller image. In this case *AVW_CreateImage()* may be called to create an image of desired dimensions. This image is then passed to the routine and the results are returned in it. *Trans_image* will not be reallocated if it does not match the *DataType* of *in_volume* as in other functions.

RETURN VALUES

If successful *AVW_TransformVolumeSliceBySlice()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_TransformVolumeSliceBySlice() will fail if the following is true:

BADMAL
Could not allocate memory for the results.

CFLSZ
Input and output data types conflict.

SEE ALSO

AVW_FlipVolume(), AVW_TransformImage(), AVW_TransformVolume(), AVW_Image, AVW_Matrix, AVW_Volume

NAME

AVW_TranslateMatrix – translates a transformation matrix

SYNOPSIS

```
#include "AVW.h"

AVW_Matrix *AVW_TranslateMatrix(in_matrix, xvoxels, yvoxels, zvoxels, out_matrix)
AVW_Matrix *in_matrix;
double xvoxels;
double yvoxels;
double zvoxels;
AVW_Matrix *out_matrix;
```

DESCRIPTION

AVW_TranslateMatrix() applies the translation specified by *xvoxels, yvoxels,* and *zvoxels* to the *AVW_Matrix, in_matrix.*

Out_matrix is provided as a method of reusing an existing *AVW_Matrix. (See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_TranslateMatrix()* returns an *AVW_Matrix.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_CopyMatrix(), AVW_CreateMatrix(), AVW_DestroyMatrix(), AVW_InvertMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis(), AVW_MatrixAngles(), AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_ScaleMatrix(), AVW_SetIdentityMatrix(), AVW_Matrix

NAME

AVW_TranslatePointList2 – translates the points in a point list

SYNOPSIS

```
#include    "AVW.h"

int AVW_TranslatePointList2(ptlist, xshift, yshift)
AVW_PointList2 *ptlist;
int xshift;
int yshift;
```

DESCRIPTION

AVW_TranslatePointList2() applies the translation specified by *xshift* and *yshift* to the *AVW_PointList2, ptlist.*

RETURN VALUES

If successful *AVW_TranslatePointList2()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned.

SEE ALSO

AVW_AddPoint2(), AVW_ClipPointList2(), AVW_ClosestInPointList2(), AVW_CopyPointList2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_FillPointList2(), AVW_GetPoint2(), AVW_RemovePoint2(), AVW_RotatePointList2(), AVW_ScalePointList2(), AVW_TransformPoint2(), AVW_PointList2

NAME

AVW_TreeAnalysis – calculate tree statistics

SYNOPSIS

```
#include "AVW_Tree.h"

int AVW_TreeAnalysis(tree, volume, t_max, tmin, c_radius, filename)
AVW_Tree *tree;
AVW_Volume *volume;
double t_max, tmin;
int c_radius;
char *filename;
```

DESCRIPTION

AVW_TreeAnalysis creates detailed and summary statictics files for an *AVW_Tree* structure,
.I tree,
within a volume. Small oblique images are generated for each point along the tree and statistical information is written to a disk file.

Tree is an *AVW_Tree* structure which describes a vessel structures within a volume. See *AVW_MakeTree().*

Volume contains the information to be measured.

T_max and *t_min* allow voxels outside a threshold range to excluded in calculations.

C_radius defines a circle mask, which eliminates voxels outside the mask. Small vessel trees will need a smaller circle size. *C_Radius* should be set to the expected maximum vessel radius.

Filename specifies the filename and path for the files. ".stats" will be appended to the file containing the detailed statistics and "_sum.stats" is appended for the summary file. Each individual oblique image is also saved to a file, the name of this file is "filename.moblq".

Detailed Format

```
#File Name = filename.stats
#
#
#             Brightness
#               Area
# Name  Segment   Product   Area   X    Y    Z
#====== ======= ========== ====== ==== ==== ====
  A1     1       123.56    123   12   34   56
```

Summary Format

```
#File Name = filename_sum.stats
#
#        Average
#       Brightness
#         Area      BAP     Average  Area           Children
# Name    Product   St Dev    Area   St Dev  Length  (Index,Angle)
#====== =========== ======= ======== ======= ====== =====================
  A1     123.56     12.34   123.56    1.23    123  (B1, 61.3) (B2, 123.0)
```

Name is made up of two parts, level and index. Level is represented by a single capital letter, or for complex trees, a single small letter followed by a single capitol letter. The index part, is shown as single or multiple digits. The index starts at 1 and counts up for different branchs at a given level. *Names* can be used to locate child segments.

Segment is a count of each individual oblique image along the tree path. It begins at 1 for each branch point.

Brightness Area Product is the measure of the sum of all brightness values once the *t_min* value has been subtracted.

Area is the count of all voxels within the circle mask and the threshold range.

X, *Y*, and *Z* are the coordinates for the center point of each oblique image.

Average BAP is the average of all BAPs for that branch.

BAP St Dev is the standard deviation of the BAPs.

Average Area is the average of all the areas for that branch.

Length indicates the length of the branch.

Children indicates if the tree continues. If values are present they are paired, with the name of the child branch followed by it's 3D angle from it's parent branch.

RETURN VALUES

If successful *AVW_TreeAnalysis()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_TreeAnalysis() will fail if:

SCRERR
Unable to create stats file.

SEE ALSO

AVW_MakeTree(), AVW_Tree, AVW_Volume

NAME

AVW_UltimateErosionImage – reduces an image to its final components

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_UltimateErosionImage(in_image, element, out_image)
AVW_Image *in_image;
AVW_Image *element;
AVW_Image *out_image;
```

DESCRIPTION

AVW_UltimateErosionImage() performs succesive erosions on *in_image* using *element.* The erosion process creates disconnected components which eventually disappear. The collection of these components prior to being eliminated from the image via erosion make up the *final components* of the image. *Out_image* will contain the final components which are labelled with the number of erosions required to create them.

Element is translated so that its centerpoint lies on every point of the image. At each point in the data, if a nonzero pixel in the structuring element corresponds to a zero pixel in the data the point in the result data is set to zero. Otherwise the pixel is unchanged. This process is repeated until all final components have been collected.

In_image does not have to be a binary valued image, but all nonzero pixels are treated as ones. *In_image, out_image,* and *element* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_image* and *out_image* are the same.

AVW_CreateStructuringImage() or *AVW_CreateImage(), AVW_SetImage(),* and *AVW_PutPixel()* may be used to create a structuring element

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UltimateErosionImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_UltimateErosionImage() will fail if:

ILLDT
: Data type is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_UltimateErosionVolume(), AVW_CreateStructuringImage(), AVW_ErodeImage(), AVW_CreateImage(), AVW_SetImage(), AVW_PutPixel(), AVW_Image

NAME

AVW_UltimateErosionVolume – reduces a volume to its final components

SYNOPSIS

#include "AVW.h"

AVW_Volume *AVW_UltimateErosionVolume(in_volume, element, out_volume)
AVW_Volume *in_volume;
AVW_Volume *element;
AVW_Volume *out_volume;

DESCRIPTION

AVW_UltimateErosionVolume() performs succesive erosions on *in_volume* using *element.* The erosion process creates disconnected components which eventually disappear. The collection of these components prior to being eliminated from the volume via erosion make up the *final components* of the volume. *Out_volume* will contain the final components which are labelled with the number of erosions required to create them.

Element is translated so that its centerpoint lies on every point of the volume. At each point in the data, if a nonzero voxel in the structuring element corresponds to a zero voxel in the data the point in the result data is set to zero. Otherwise the voxel is unchanged. This process is repeated until all final components have been collected.

In_volume does not have to be a binary valued volume, but all nonzero voxels are treated as ones. *In_volume, out_volume,* and *element* must be of the data type *AVW_UNSIGNED_CHAR.* This function will allocate temporary storage space for results if *in_volume* and *out_volume* are the same.

AVW_CreateStructuringVolume() or *AVW_CreateVolume(), AVW_SetVolume(),* and *AVW_PutVoxel()* may be used to create a structuring element

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UltimateErosionVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_UltimateErosionVolume() will fail if:

ILLDT
Data type is not AVW_UNSIGNED_CHAR.

SEE ALSO

AVW_UltimateErosionImage(), AVW_CreateStructuringVolume(), AVW_ErodeVolume(), AVW_CreateVolume(), AVW_SetVolume(), AVW_PutVoxel(), AVW_Volume

NAME

AVW_UnsharpFilterEnhanceImage – enhances an image by Unsharp filtering

SYNOPSIS

#include "AVW_Filter.h"

AVW_Image *AVW_UnsharpFilterEnhanceImage(in_image, extents, out_image)
AVW_Image *in_image;
int extents[2];
AVW_Image *out_image;

DESCRIPTION

AVW_UnsharpFilterEnhanceImage() performs a Unsharp filter transformation on *in_image*. The returned *out_image* is created by adding *in_image* to the Unsharp filter transformation of the input image.

Extents[0], and *extents[1]*, specify the x and y sizes respectively of the filter.

The unsharp filter is a high-pass filter created by subtracting a low-pass filtered image from the original.

Out_image is provided as a method of reusing an existing *AVW_Image*. Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UnsharpFilterEnhanceImage()* returns an *AVW_Image*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_AnisotropicAffineImage(), AVW_AnisotropicDiffusionImage(), AVW_LowpassFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterEnhanceVolume(), AVW_Image

NAME

AVW_UnsharpFilterEnhanceVolume – enhances a volume by Unsharp filtering

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_UnsharpFilterEnhanceVolume(in_volume, extents, out_volume)
AVW_Volume *in_volume;
int extents[3];
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_UnsharpFilterEnhanceVolume() performs a Unsharp filter transformation on *in_volume.* The returned *out_volume* is created by adding *in_volume* to the Unsharp filter transformation of the input volume.

Extents[0], extents[1], and *extents[2]* specify the x, y, and z sizes respectively of the filter.

The unsharp filter is a high-pass filter created by subtracting a low-pass filtered volume from the original.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UnsharpFilterEnhanceVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SobelFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_SigmaFilterVolume(), AVW_UnsharpFilterEnhanceImage(), AVW_UnsharpFilterVolume(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_UnsharpFilterImage – performs a 2D Unsharp filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Image *AVW_UnsharpFilterImage(in_image, extents, out_image)
AVW_Image *in_image;
int extents[2];
AVW_Image *out_image;
```

DESCRIPTION

AVW_UnsharpFilterImage() performs an Unsharp filter transformation on *in_image.*

Extents[0] and *extents[1]* specify the x and y sizes respectively of the filter.

The unsharp filter is a high-pass filter created by subtracting a low-pass filtered image from the original.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UnsharpFilterImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEImage(), AVW_AnisotropicAffineImage(), AVW_AnisotropicDiffusionImage(), AVW_LowpassFilterImage(), AVW_OrthoGradFilterImage(), AVW_RankFilterImage(), AVW_SigmaFilterImage(), AVW_SobelFilterImage(), AVW_SobelFilterEnhanceImage(), AVW_UnsharpFilterEnhanceImage(), AVW_UnsharpFilterVolume(), AVW_Image

NAME

AVW_UnsharpFilterVolume – performs a 3D Unsharp filter

SYNOPSIS

#include "AVW_Filter.h"

AVW_Volume *AVW_UnsharpFilterVolume(in_volume, extents, out_volume)
AVW_Volume *in_volume;
int extents[3];
AVW_Volume *out_volume;

DESCRIPTION

AVW_UnsharpFilterVolume() performs an Unsharp filter transformation on *in_volume.*

Extents[0], *extents[1]*, and *extents[2]* specify the x, y, and z sizes respectively of the filter.

The unsharp filter is a high-pass filter created by subtracting a low-pass filtered volume from the original.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UnsharpFilterVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SobelFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_SigmaFilterVolume(), AVW_UnsharpFilterImage(), AVW_UnsharpFilterEnhanceImage(), AVW_VSFMeanFilterVolume(), AVW_Volume

NAME

AVW_UnsuperClassifyImage – classifies pixels from multi-spectral data sets

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_UnsuperClassifyImage(imgs, numimgs, type,
            centroidArray, threshold, pchange,
            classes, out_image)

AVW_Image **imgs;
int numimgs;
int type;
char *centroidfile;
int threshold;
double pchange;
int classes;
AVW_Image *out_image;
```

DESCRIPTION

AVW_UnsuperClassifyImage() without supervision or training samples classifies pixels given multi-spectral data consisting of several input images. Three different algorithms can be used to classify unclassified pixels from the input mask image.

***Imgs* is a list of spatially correlated images, each of which represents one spectra of the multi-spectral data set. These input images must be of the same dimension and of data type *AVW_UNSIGNED_CHAR*. This input data is unchanged by the classification process.

Numimgs is the number of images in the array. *imgs.*

Type specifies the unsupervised classification algorithm. Acceptable values are *AVW_UNSUPERVISED_CHAIN*, *AVW_UNSUPERVISED_ISODATA*, and *AVW_UNSUPERVISED_ISOMERGE*

The *AVW_UNSUPERVISED_CHAIN* technique arbitrarily assigns the first voxel within the data set to class 1, making it the initial centroid of that class. Subsequent voxels are placed into that class or a new class depending on their Euclidean distance from the class cetroids. The centroids are updated every time a new member is added to the class.

The *AVW_UNSUPERVISED_ISODATA* classifier calculates a set of centroids to cover a broad region of the feature space. Proximity of initial centroids to their final positions determines the number of iterations it takes to meet the stopping criteria. Voxels are classified based on their distance from the initial class centroids similarly to the superised Nearest Neighbors algorithm.

The *AVW_UNSUPERVISED_ISOMERGE* technique expands on the ISODATA algorithm to include merging of clusters whose centroids are closer than a user-defined threshold distance in feature space. At that point everything starts over, each voxel being compared to the newly merged cluster centroids in order to determine class membership.

Centroidfile is a text file containing centroids used to pass starting centroids of starting classes to *AVW_UnsuperClassifyImage().*

Threshold specifies the maximum distance that a voxel can be from existing class centroids before a new class is formed when type is AVW_UNSUPERVISED_CHAIN. For the *AVW_UNSUPERVISED_ISOMERGE algorithm, this distance is the*

AVW_UNSUPERVISED_ISODATA threshold is ignored.

Pchange specifies the stopping criterion for all the unsupervised classifiers. When the percentage of voxels which have changed from the previous iteration falls below this number, the classification stops.

Classes is used to randomly determine the number of class centroids chosen in feature space as initial class centroids.

Out_image is the returned classified image. Pixels which have been successfully classified are set to values of pixels from the training sample mask image which they are most similar to. Pixels with a value of 0 were not classified by the function.

Out_image is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_image* meet the requirements of the function. In this case the pointer to *out_image* is returned by the function. If not reusable *out_image* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UnsuperClassifyImage()* returns an *AVW_Image.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyImage will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLIMG
Illegal Image. The images are not all the same dimension.

INSPEC
Insufficient Specifications. Fewer than two input images were supplied.

TOMNCLS
To Many Unsupervised Classes. More than AVW_MAX_UNSUPERVISED_CLASSES developed during classification.Try a smaller value for threshold.

SEE ALSO

AVW_ClassifiedImageToCentroidFile(), AVW_ClassifyImage()
AVW_ClassifyImageFromSampleFile(), AVW_MaskImageToSampleFile(),
AVW_UnsuperClassifyVolume() AVW_Image

REFERENCES

G.H. Ball & D.J. Hall "ISODATA", an iterative method of multivariate data analysis and pattern classification. In IEEE International Communications Conference, Philadelphia, June 1966.

NAME

AVW_UnsuperClassifyVolume – classifies pixels from multi-spectral data sets

SYNOPSIS

```
#include "AVW.h"

AVW_Volme *AVW_UnsuperClassifyVolume(vols, numvols, type,
        centroidfile, threshold, pchange,
        classes, out_vol)

AVW_Volume **vols;
int numvols;
int type;
char *centroidfile;
int threshold;
double pchange;
int classes;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_UnsuperClassifyVolume() without supervision or training samples classifies voxels given multi-spectral data consisting of several input volumes. Three different algorithms can be used to classify unclassified voxels from the input volumes.

***Vols* is a list of spatially correlated volumes, each of which represents one spectra of the multi-spectral data set. These input volumes must be of the same dimension and of data type *AVW_UNSIGNED_CHAR*. This input data is unchanged by the classification process.

Numvols is the number of vols in the array *vols.*

Type specifies the unsupervised classification algorithm. Acceptable values are *AVW_UNSUPERVISED_CHAIN*, *AVW_UNSUPERVISED_ISODATA*, and *AVW_UNSUPERVISED_ISOMERGE*

The *AVW_UNSUPERVISED_CHAIN* technique arbitrarily assigns the first voxel within the data set to class 1, making it the initial centroid of that class. Subsequent voxels are placed into that class or a new class depending on their Euclidean distance from the class centroids. The centroids are updated every time a new member is added to the class.

The *AVW_UNSUPERVISED_ISODATA* classifier calculates a set of centroids to cover a broad region of the feature space. Proximity of initial centroids to their final positions determines the number of iterations it takes to meet the stopping criteria. Voxels are classified based on their distance from the initial class centroids similar to the supervised Nearest Neighbors algorithm.

The *AVW_UNSUPERVISED_ISOMERGE* technique expands on the ISODATA algorithm to include merging of clusters whose centroids are closer than a user-defined threshold distance in feature space. At that point everything starts over, each voxel being compared to the newly merged cluster centroids in order to determine class membership.

Centroidfile is a text file containing centroids used to pass starting centroids of starting classes to *AVW_UnsuperClassifyImage().*

Threshold specifies the maximum distance that a voxel can be from existing class centroids before a new class is formed when type is AVW_UNSUPERVISED_CHAIN. For the *AVW_UNSUPERVISED_ISOMERGE* algorithm, this distance is the minimum distance between class centroids. Classes whose centroids are closer than this distance will be merged into that class. For *AVW_UNSUPERVISED_ISODATA threshold* is ignored.

Pchange specifies the stopping criterion for all the unsupervised classifiers. When the percentage of voxels which have changed from the previous iteration falls below this number, the classification stops.

Classes is used to randomly determine the number of class centroids chosen in feature space as initial class centroids.

Out_volume is the returned classified volume. Voxels which have been successfully classified are set to values of voxels from the training sample mask image to which they are most similar. Voxels with a value of 0 were not classified by the function.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_UnsuperClassifyVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_ClassifyVolume will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLVOL
Illegal Volume. The volumes are not all the same dimension.

INSPEC
Insufficient Specifications. Fewer than two input volumes were supplied.

TOMNCLS
To Many Unsupervised Classes. More than AVW_MAX_UNSUPERVISED_CLASSES developed during classification.Try a smaller value for threshold.

SEE ALSO

AVW_ClassifiedVolumeToCentroidFile(), AVW_ClassifyVolume(), AVW_ClassifyVolumeFromSampleFile(), AVW_MaskVolumeToSampleFile(), AVW_UnsuperClassifyImage(), AVW_Volume

REFERENCES

G.H. Ball & D.J. Hall "ISODATA", an iterative method of multivariate data analysis and pattern classification. In IEEE International Communications Conference, Philadelphia, June 1966.

NAME

AVW_UpdateConfidenceClasses – updates liklihood data from AVW_GetLiklihoods()

SYNOPSIS

```
#include "AVW.h"

int AVW_UpdateConfidenceClasses(classifiedImage, liklihoods, alpha)
AVW_Image *classifiedImage;
AVW_Volume *liklihoods
double alpha;
```

DESCRIPTION

AVW_UpdateConfidenceClasses() updates the liklihood data from *AVW_GetLiklihoods().* and reclassifies the passed in previously *classifiedImage.*

classifiedImage is image initially classified with the *AVW_GAUSSIAN_CLUSTER, AVW_NEURAL_NETWORK,* or *AVW_PARZEN_WINDOWS* classification algorithms.

liklihoods is the liklihood data returned by *AVW_GetLiklihoods().*

alpha is a confidence-weigthing parameter which controls the strength of the iterative relaxation. High alpha means that voxels are influenced more strongly by classification results of their spatial neighbors, and results in a smoother overall classification; low *alpha* means the opposite (alpha - 0.0) turns relaxation off). High values of Alpha (˜4.0) are suggested for the AVW_GAUUSIAN_CLUSTER and AVW_PARZEN_WINDOWS classifiers and low values (˜0.25) for the AVW_NEURAL_NETWORK clasiifier

AVW_UpdateConfidenceClasses is used with other AVW functions to perform iterative relaxationon

classification performed with *AVW_ClassifyImage()* or *AVW_ClassifyVolume()* when the *autotype* parameter is set to one of the statistics based classification algorithms. *AVW_GAUSSIAN_CLUSTER, AVW_NEURAL_NETWORK,* or *AVW_PARZEN_WINDOWS*

Once an initial multispectral classification has been performed, the data in the returned liklihood volume can be used in a process of iterative relaxation with the additional AVW functions *AVW_UpdateConfidenceClasses()* and *AVW_UpdateImageClassification()*

to re-evaluate the classification of individual pixels based on the class assignments of neighboring pixels and the relative probabilities that a pixel belongs to each class.

RETURN VALUES

AVW_UpdateConfidenceClasses() returns the number of pixels which were reclassified in *classifiedImage.*

AVW_UpdateConfidenceClasses() will fail if one or more of the following are true:

BADMAL
Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_GetScatLiklihoods(), AVW_UpdateImageClassification()

NAME

AVW_UpdateImageClassification – updates an image classification through iterative relaxation

SYNOPSIS

```
#include "AVW.h"

int AVW_UpdateImageClassification(classifiedImage, liklihoods, alpha)
AVW_Image *classifiedImage;
AVW_Volume *liklihoods
double alpha;
```

DESCRIPTION

AVW_UpdateImageClassification() uses iterative relaxation and liklihood information from a previous classification to re-evaluate and change voxels based on class assignements of neighboring voxels liklihood data from the last multispectral classification.

classifiedImage is an image initially classified with the *AVW_GAUSSIAN_CLUSTER*, *AVW_NEURAL_NETWORK*, or *AVW_PARZEN_WINDOWS* classification algorithms.

liklihoods is the liklihood data returned by *AVW_GetLiklihoods()*.

alpha is a confidence-weigthing parameter which controls the strength of the iterative relaxation. High alpha means that voxels are influenced more strongly by classification results of their spatial neighbors, and results in a smoother overall classification; low *alpha* means the opposite (alpha - 0.0) turns relaxation off). High values of Alpha (˜4.0) are suggested for the AVW_GAUUSIAN_CLUSTER and AVW_PARZEN_WINDOWS classifiers and low values (˜0.25) for the AVW_NEURAL_NETWORK clasiifier

RETURN VALUES

If successful *AVW_UpdateImageClassification()* returns an the number of changed voxels in *classifiedImage*. On failure 0 is returned and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values corresponding to the cause of the failure.

ERRORS

AVW_UpdateImageClassification() will fail ifthe following is true:

BADMAL
: Malloc Failed. Unable to allocate memory for structure and / or pixel memory.

SEE ALSO

AVW_ClassifyImage(), AVW_ClassifyVolume(), AVW_GetScatLiklihoods(), AVW_UpdateConfidenceClasses(), AVW_GetLiklihoods()

NAME

AVW_UpdateImageMask – updates image pixel values

SYNOPSIS

```
#include "AVW.h"

int AVW_UpdateImageMask(image, mask, mode, value)
AVW_Image *image;
AVW_Image *mask;
int mode;
double value;
```

DESCRIPTION

AVW_UpdateImageMask() sets the value of all of the pixels of *image* to a value based on the corresponding value in *mask* and the *mode* value.

The *mode* parameter determines how pixels values are determined:

A value of *AVW_REPLACE_MASK_VALUE* specifies that all of the pixels values of *image* are set to *value* if the corresponding pixel in *mask* is nonzero.

A value of *AVW_MAINTAIN_MASK_VALUE* means that only the zero valued pixels of *image* that correspond to nonzero pixels in *mask* are set to *value.* The nonzero pixels of *image* maintain their value regadless of the corresponding pixel value in *mask.*

AVW_NEW_MASK_VALUE causes each seperate connected intersection of the two images to become a new intensity. The maximum of the *image* is determined and the intersections are assigned values above the maximum as they are encountered. Zero valued pixels of *image* that correspond to nonzero pixels in *mask* are set to *value.*

A value of *AVW_OR_MASK_VALUE* causes the pixel value in *image* to be OR'd with *value* for each non-zero pixel in *mask.*

A value of *AVW_AND_MASK_VALUE* causes the pixel value in *image* to be AND'd with *value* for each non-zero pixel in *mask.*

Mask must be of data type *AVW_UNSIGNED_CHAR*.

RETURN VALUES

If successful *AVW_UpdateImageMask()* returns *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_UpdateImageMask() will fail if:

ILLDT
Illegal data type.

ILLPAR
A NULL image was specified.

CFLSZ
The images were not the same size.

SEE ALSO

AVW_GetMaskedImage(), AVW_MakeMaskFromTrace(), AVW_PutMaskedImage(), AVW_UpdateVolumeMask(), AVW_Image

NAME

AVW_UpdateVolumeMask – updates volume voxel values

SYNOPSIS

```
#include "AVW.h"

int AVW_UpdateVolumeMask(volume, mask, mode, value)
AVW_Volume *volume;
AVW_Volume *mask;
int mode, value;
```

DESCRIPTION

AVW_UpdateVolumeMask() sets the value of all of the voxels of *volume* to a value based on the corresponding value in *mask* and the *mode* value.

The *mode* parameter determines how voxels values are determined:

A value of *AVW_REPLACE_MASK_VALUE* specifies that all of the voxels values of *volume* are set to *value* if the corresponding voxel in *mask* is nonzero.

A value of *AVW_MAINTAIN_MASK_VALUE* means that only the zero valued voxels of *volume* that correspond to nonzero voxels in *mask* are set to *value.* The nonzero voxels of *volume* maintain their value regadless of the corresponding voxel value in *mask.*

AVW_NEW_MASK_VALUE causes each seperate connected intersection of the two volumes to become a new intensity. The maximum of the *volume* is determined and the intersections are assigned values above the maximum as they are encountered. Zero valued voxels of *volume* that correspond to nonzero voxels in *mask* are set to *value.*

A value of *AVW_OR_MASK_VALUE* causes the voxel value in *volume* to be OR'd with *value* for each non-zero voxel in *mask.*

A value of *AVW_AND_MASK_VALUE* causes the voxel value in *volume* to be AND'd with *value* for each non-zero voxel in *mask.*

Volume and *Mask* must be of data type *AVW_UNSIGNED_CHAR.*

RETURN VALUES

If successful *AVW_UpdateVolumeMask()* returns *AVW_SUCCESS.* On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_UpdateVolumeMask() will fail if:

ILLDT
: Illegal data type.

ILLPAR
: A NULL volume was specified.

CFLSZ
: The volumes were not the same size.

SEE ALSO

AVW_GetMaskedVolume(), AVW_MakeMaskFromTrace(), AVW_PutMaskedVolume(), AVW_UpdateImageMask(), AVW_Volume

NAME

AVW_VSFMeanFilterVolume – performs a VSFmean filter

SYNOPSIS

```
#include "AVW_Filter.h"

AVW_Volume *AVW_VSFMeanFilterVolume(in_volume, ring, sigma, out_volume)
AVW_Volume *in_volume;
int ring;
int sigma;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_VSFMeanFilterVolume() performs a VSFmean filter transformation on the input volume. Voxels are averaged in a circular neighborhood of size *ring* voxels about a seed voxel. Neighboring voxels whose values differ from the seed voxel by a value greater than *sigma* are not included in the mean calculation.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reusable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_VSFMeanFilterVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

SEE ALSO

AVW_AHEVolume(), AVW_InhomogeneityCorrectVolume(), AVW_LowpassFilterVolume(), AVW_OrthoGradFilterVolume(), AVW_RankFilterVolume(), AVW_SigmaFilterVolume(), AVW_SobelFilterVolume(), AVW_SobelFilterEnhanceVolume(), AVW_UnsharpFilterVolume(), AVW_UnsharpFilterEnhanceVolume(), AVW_Volume

NAME

AVW_ValidationErrorFunction – indicates the function which handles validation error reporting

SYNOPSIS

#include "AVW.h"

void AVW_ValidationErrorFunction(int (∗function)())

DESCRIPTION

By default, *AVW* dumps all validation errors to either *stdout* or *stderr*. *AVW_ValidationErrorFunction* allows the programmer the capability of reporting the errors to the user in a friendlier fashion. The user can also choose to attempt recovery, if the condition is recoverable.

When a validation error is detected, AVW_VALIDATE_UNRECOVERABLE, AVW_VALIDATE_RECOVERABLE or *AVW_VALIDATE_WARNING*. If the type is *AVW_VALIDATE_RECOVERABLE* the return value from the function is checked, if one (1), the function which reported the error will be retried.

NAME

AVW_VerifyDataType – verifies valid AVW data types

SYNOPSIS

```
#include    "AVW.h"

int     AVW_VerifyDataType(type)
int type;
```

DESCRIPTION

AVW_VerifyDataType() verifies that *type* is a supported *AVW* data type.

Valid *AVW* data types, defined in *AVW.h,* include:

AVW_UNSIGNED_CHAR

AVW_SIGNED_CHAR

AVW_UNSIGNED_SHORT

AVW_SIGNED_SHORT

AVW_UNSIGNED_INT

AVW_SIGNED_INT

AVW_FLOAT

AVW_COMPLEX

AVW_COLOR

RETURN VALUES

If successful *AVW_VerifyDataType()* returns *AVW_TRUE.* On failure it returns *AVW_FALSE.*

SEE ALSO

AVW_VerifyImage(), AVW_VerifyVolume()

NAME

AVW_VerifyHistogram – verifies a valid AVW_Histogram

SYNOPSIS

```
#include "AVW_Histogram.h"

int AVW_VerifyHistogram(histo)
AVW_Histogram *histo;
```

DESCRIPTION

AVW_VerifyHistogram() verifies that *histo* is a supported *AVW_Histogram.*

RETURN VALUES

If successful *AVW_VerifyHistogram()* returns *AVW_TRUE.* On failure it returns *AVW_FALSE.*

SEE ALSO

AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_DestroyHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_NormalizeHistogram(), AVW_ReadHistogram(), AVW_WriteHistogram(), AVW_Histogram

NAME

AVW_VerifyImage – verifies a valid AVW_Image structure

SYNOPSIS

```
#include "AVW.h"

int AVW_VerifyImage(image)
AVW_Image *image;
```

DESCRIPTION

AVW_VerifyImage() checks consistancy within the *AVW_Image* structure elements to determine if the pointer is to a valid *AVW_Image.*

An invalid *AVW_Image* is detected if:

image->Mem is *NULL,*

image->DataType is invalid,

image->Width or *image->Height* are zero or negative,

image->BytesPerPixel incorrect for *image->Datatype,*

image->BytesPerLine does not equal *image->BytesPerPixel* X *image->Width,*

image->BytesPerImage does not equal *image->BytesPerLine* X *image->Height,*

image->PixelsPerImage does not equal *image->Width* X *image->Height.*

RETURN VALUES

If successful *AVW_VerifyImage()* returns *AVW_TRUE.* On failure it returns *AVW_FALSE.*

SEE ALSO

AVW_ClosestPointInImage(), AVW_CopyImage(), AVW_CreateImage(), AVW_DestroyImage(), AVW_VerifyDataType(), AVW_VerifyVolume(), AVW_Image

NAME

AVW_VerifyVolume – verifies a valid AVW_Volume structure

SYNOPSIS

```
#include "AVW.h"

int AVW_VerifyVolume(volume)
AVW_Volume *volume;
```

DESCRIPTION

AVW_VerifyVolume() checks consistancy within the *AVW_Volume* structure elements to determine if the pointer is to a valid *AVW_Volume.*

An invalid *AVW_Volume* is detected if:

volume->Mem is *NULL,*

volume->Datatype is illegal,

volume->Width, volume->Height or *volume->Depth* are zero or negative,

volume->BytesPerPixel incorrect for *volume->DataType,*

volume->BytesPerLine does not equal *volume->BytesPerPixel* X *volume->Width,*

volume->BytesPerImage does not equal *volume->BytesPerLine* X *volume->Height,*

volume->PixelsPerImage does not equal *volume->Width* X *volume->Height,*

volume->BytesPerVolume does not equal *volume->BytesPerImage* X *volume->Depth,*

volume->VoxelsPerVolume does not equal *volume->Width* X *volume->Height* times *volume->Depth.*

RETURN VALUES

If successful *AVW_VerifyVolume()* returns *AVW_TRUE.* On failure it returns *AVW_FALSE.*

SEE ALSO

AVW_CopyVolume(), AVW_CreateVolume(), AVW_DestroyVolume(), AVW_MakeVolumeFromImage(), AVW_VerifyDataType(), AVW_VerifyImage(), AVW_Volume

NAME

AVW_VolumeOpConstant – transforms a volume mathematically

SYNOPSIS

```
#include "AVW_Parse.h"

AVW_Volume *AVW_VolumeOpConstant(in_volume, operation, value, out_volume)
AVW_Volume *in_volume;
int operation;
double value;
AVW_Volume *out_volume;
```

DESCRIPTION

AVW_VolumeOpConstant() applies the *operation* and *value* to *in_volume* and returns the resulting volume:

output = *in_volume operation value*

AVW_VolumeOpConstant(), calls *AVW_ImageOpConstant()*, with each slice in the input volume to produce the output volume.

Out_volume is provided as a method of reusing an existing *AVW_Volume*. Reuse is possible only if the size and data type of the provided *out_volume* meet the requirements of the function. In this case the pointer to *out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

The following operations are defined in *AVW_Parse.h:*

AVW_OP_ADD

AVW_OP_SUB

AVW_OP_MUL

AVW_OP_DIV

AVW_OP_LT

AVW_OP_GT

AVW_OP_LE

AVW_OP_GE

AVW_OP_EQ

AVW_OP_NE

AVW_OP_AND

AVW_OP_OR

AVW_OP_MOD

RETURN VALUES

If successful *AVW_VolumeOpConstant()*, returns an *AVW_Volume*. On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_VolumeOpConstant() will fail if one or more of the following are true:

NOTSUP
Operation is not supported.

DIVZER
Division by zero.

BADMAL
Memory could not be allocated for results.

ILLDT
Data type is not defined or supported.

CFLSZ
Input volumes conflict in size.

SEE ALSO

AVW_ConstantOpImage(), AVW_ConstantOpVolume(), AVW_EvalFormula(), AVW_FunctionImage(), AVW_FunctionVolume(), AVW_ImageOpConstant(), AVW_ImageOpImage(), AVW_VolumeOpVolume(), AVW_Volume

NAME

AVW_VolumeOpVolume – transforms a volume mathematically

SYNOPSIS

#include "AVW_Parse.h"

AVW_Volume *AVW_VolumeOpVolume(in_volume1, operation, in_volume2, out_volume)
AVW_Volume *in_volume1;
int operation;
AVW_Volume *in_volume2;
AVW_Volume *out_volume;

DESCRIPTION

AVW_VolumeOpVolume() applies the *operation* and *in_volume2* to *in_volume1* and returns the resulting image:

output = *in_volume1 operation in_volume2*

AVW_VolumeOpVolume() calls *AVW_ImageOpImage()* with each slice in the input volumes to produce the output volume.

Out_volume is provided as a method of reusing an existing *AVW_Volume.* Reuse is possible only if the size and data type of the provided *out_volume out_volume* is returned by the function. If not reuseable *out_volume* will be reallocated. (See *Memory Usage* in the *AVW Programmer's Guide.)*

The following operations are defined in *AVW_Parse.h:*

AVW_OP_ADD

AVW_OP_SUB

AVW_OP_MUL

AVW_OP_DIV

AVW_OP_LT

AVW_OP_GT

AVW_OP_LE

AVW_OP_GE

AVW_OP_EQ

AVW_OP_NE

AVW_OP_AND

AVW_OP_OR

AVW_OP_MOD

RETURN VALUES

If successful *AVW_VolumeOpVolume()* returns an *AVW_Volume.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_VolumeOpVolume() will fail if one or more of the following are true:

NOTSUP
Operation is not supported.

DIVZER
Division by zero.

BADMAL
Memory could not be allocated for results.

ILLDT
Data type is not defined or supported.

CFLSZ
Input volumes conflict in size.

SEE ALSO

AVW_ConstantOpImage(), AVW_ConstantOpVolume(), AVW_EvalFormula(), AVW_FunctionImage(), AVW_FunctionVolume(), AVW_ImageOpConstant(), AVW_ImageOpImage(), AVW_VolumeOpConstant(), AVW_Volume

NAME

AVW_VolumeSampleEntropy – Calculate the entropy of a sample of voxels

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

double AVW_VolumeSampleEntropy(volume,points,interpolate)
AVW_Volume *volume;
AVW_FPointList3 *points;
int interpolate;
```

DESCRIPTION

AVW_VolumeSampleEntropy() calculates the entropy of a sample of voxels defined by a list of floating-point 3-D coordinates.

Interpolate determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

RETURN VALUES

If successful returns the entropy.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_VolumeSampleEntropy() will fail if the following is true:

ILLVOL
Illegal Volume.

ILLPAR
Interpolation type is not recognized.

SEE ALSO

AVW_VolumeSampleJointEntropy(), AVW_VolumeSampleNMI(), AVW_ImageSampleEntropy().

NAME

AVW_VolumeSampleJointEntropy – Calculate the joint entropy of a sample of voxels from two volumes

SYNOPSIS

#include "AVW_MatchVoxels.h"

```
double AVW_VolumeSampleJointEntropy(base,match,points,matrix,interpolate)
AVW_Volume *base,*match;
AVW_FPointList3 *points;
AVW_Matrix *matrix;
int interpolate;
```

DESCRIPTION

AVW_VolumeSampleJointEntropy() calculates the joint entropy of a sample of voxels defined by a list of floating-point 3-D coordinates. *points* defines a set of 3-D coordinates in the *match* volume to be sampled. Those voxel values are paired with those from the same coordinates transformed by *matrix* in the *base* volume. The joint entropy of the samples is returned

Interpolate determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

RETURN VALUES

If successful returns the joint entropy.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_VolumeSampleJointEntropy() will fail if the following is true:

ILLVOL
Illegal Volume. Floating point, Complex, and color images are not supported

BADMAL
Unable to allocate memory for internal calculations.

ILLPAR
Interpolation type is not recognized.

SEE ALSO

AVW_VolumeSampleEntropy(), AVW_VolumeSampleNMI(), AVW_ImageSampleJointEntropy().

NAME

AVW_VolumeSampleNMI – Calculate the Normalized Mutual Information of a sample of voxels from two volumes

SYNOPSIS

```
#include "AVW_MatchVoxels.h"

double AVW_VolumeSampleNMI(base,match,points,matrix,interpolate)
AVW_Volume *base,*match;
AVW_FPointList3 *points;
AVW_Matrix *matrix;
int interpolate;
```

DESCRIPTION

AVW_VolumeSampleNMI() calculates the normalized mutual information of a sample of voxels defined by a list of floating-point 3-D coordinates. *points* defines a set of 3-D coordinates in the *match* volume to be sampled. Those voxel values are paired with those from the same coordinates transformed by *matrix* in the *base* volume. The normalized mutual information (sum of individual image entropies divided by joint entropy) of the samples is returned

Interpolate determines the method of interpolation to use. Choose from:

AVW_NEAREST_NEIGHBOR_INTERPOLATE

AVW_LINEAR_INTERPOLATE

AVW_CUBIC_SPLINE_INTERPOLATE

AVW_WINDOWED_SINC_INTERPOLATE

RETURN VALUES

If successful returns the normalized mutual information.

On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_VolumeSampleNMI() will fail if the following is true:

ILLVOL
Illegal Volume. Floating point, Complex, and color images are not supported

BADMAL
Unable to allocate memory for internal calculations.

ILLPAR
Interpolation type is not recognized.

SEE ALSO

AVW_VolumeSampleEntropy(), AVW_VolumeSampleJointEntropy(), AVW_ImageSampleNMI().

NAME

AVW_WaveletCompressAndDecompressImage – uses wavelet transformation and run-length encoding to compress and then decompress an image.

SYNOPSIS

```
#include "AVW.h"

AVW_Image *AVW_WaveletCompressAndDecompressImage(in_img,
                Levels, HVSFlag, MasterBin,
                compressedSize, out_img)
AVW_Image *in_img;
int Levels;
int HSVFlag;
double MasterBin;
int *compressedSize
AVW_Image *out_img;
```

DESCRIPTION

AVW_WaveletCompressAnddecompressImage() is a convenience function which invokes *AVW_WaveletCompressImage* and *AVW_DecompressWaveletBuffer* into a single call facilitating evaluation of the quality of compression.

in_img is an *AVW_Image* to be compressed.

Levels number of levels in the wavelet decomposition.

HVS_Flag (Human Visual System) Flag - if set to 1, the quantization is varied in the different levels to approximate the human visual system sensitivity to different frequencies. At a given compression ratio, this should result in smoother images that technically have poorer fidelity but are more pleasing to the eye.

MasterBin the quantization parameter by which the wavelet coefficients are divided. A larger value means coarser approximations of the wavelet coefficients, poorer image fidelity, and higher compression ratio; a smaller value means the reverse.

returnBufferSize is the size in bytes of the returned data buffer.

out_img is provided as a method of reusing an existing *AVW_Image.* Reuse is possible only if the size and data type of the provided *out_img* meet the requirements of the function. In this case the pointer to out_image is returned by the function. If not reuseable *out_image will be reallocated. (See Memory Usage* in the *AVW Programmer's Guide.)*

RETURN VALUES

If successful *AVW_WaveletCompressAndDecompressImage()* returns an AVW_Image. On failure it returns *AVW_NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_WaveletCompressImageFile will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLDT
Illegal Datatype. Does not support multi-band datatypes such as AVW_COLOR or AVW_COMPLEX.

SEE ALSO *AVW_WaveletCompressImage(), AVW_WaveletCompressImageFile(), AVW_Image*

REFERENCE

Manduca, A. (1997) "Compressing Images with Wavelet / Subband Coding", IEEE Engineering in Medicine and Biology, 14(5), 639-646.

NAME

AVW_WaveletCompressImage – uses wavelet transformation and run-length encoding to compress an image.

SYNOPSIS

```
#include     "AVW.h"

unsigned char  *AVW_WaveletCompressImage(in_img, Levels, HVSFlag,
                        MasterBin, returnBufferSize)
AVW_Image *in_img;
int Levels;
int HSVFlag;
double MasterBin;
int *returnBufferSize;
```

DESCRIPTION

AVW_WaveletCompressImage() compresses an AVW_Image wavelet transformation and runlength encoding into compressed a binary data buffer.

The compression scheme used, is based on uniformly quantizing the wavelet coefficients by dividing by a user-specified quanization value and rounding off (typically, a large majority of coefficients with very small values are quantized to zero by this step). The zeros in the resulting sequence are then run-length encoded, and the entire sequence is Huffman encoded. If the quantization value is increased, more coefficients are quantized to zero, the remaining ones are quantized more coarsely, the representation accuracy decreases, and the compression ratio increases; if the value is decreased, the reverse happens. An alternative (HVS) compression scheme uses the user-specified quantization value at the lowest scale, but adjusts this value at other scales in accordance with the frequency sensitivity of the human visual system - the idea begin to preferentially suppress information which a human observer cannot perceive. The differences may be subtle; typically the images may look better with HVS quantization but the RMS error between the original and compressed versions may be smaller with uniform quantization.

in_img is an *AVW_Image* to be compressed.

Levels number of levels in the wavelet decomposition.

HVS_Flag (Human Visual System) Flag - if set to 1, the quantization is varied in the different levels to approximate the human visual system sensitivity to different frequencies. At a given compression ratio, this should result in smoother images that technically have poorer fidelity but are more pleasing to the eye.

MasterBin the quantization parameter by which the wavelet coefficients are divided. A larger value means coarser approximations of the wavelet coefficients, poorer image fidelity, and higher compression ratio; a smaller value means the reverse.

returnBufferSize is the size in bytes of the returned data buffer.

The resulting data buffer can decompressed with *AVW_WaveletDecompressWaveletBuffer().*

RETURN VALUES

If successful *AVW_WaveletCompressImage()* returns an pointer to an unsigned char. On failure it returns *AVW_NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_WaveletCompressImageFile will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLDT
Illegal Datatype. Does not support multi-band datatypes such as AVW_COLOR or AVW_COMPLEX.

SEE ALSO

AVW_WaveletCompressAndDecompressImage(), AVW_WaveletCompressImageFile(), AVW_DecompressWaveletBuffer(), AVW_Image

REFERENCE

Manduca, A. (1997) "Compressing Images with Wavelet / Subband Coding", IEEE Engineering in Medicine and Biology, 14(5), 639-646.

NAME

AVW_WaveletCompressImageFile – uses wavelet transformation and run-length encoding to compress an image file

SYNOPSIS

```
#include     "AVW.h"

int *AVW_WaveletCompressImageFile(in_name, Levels,
            HVSFlag, MasterBin, outName)
char *in_name;
int Levels;
int HSVFlag;
double MasterBin;
char *outName;
```

DESCRIPTION

AVW_WaveletCompressImageFile() compresses any image file recognizeable by AVW with wavelet transformation and runlength encoding into a file of format AVW_ImageFile.

The compression scheme used, is based on uniformly quantizing the wavelet coefficients by dividing by a user-specified quanization value and rounding off (typically, a large majority of coefficients with very small values are quantized to zero by this step). The zeros in the resulting sequence are then run-length encoded, and the entire sequence is Huffman encoded. If the quantization value is increased, more coefficients are quantized to zero, the remaining ones are quantized more coarsely, the representation accuracy decreases, and the compression ratio increases; if the value is decreased, the reverse happens. An alternative (HVS) compression scheme uses the user-specified quantization value at the lowest scale, but adjusts this value at other scales in accordance with the frequency sensitivity of the human visual system - the idea begin to preferentially suppress information which a human observer cannot perceive. The differences may be subtle; typically the images may look better with HVS quantization but the RMS error between the original and compressed versions may be smaller with uniform quantization.

in_name is the name of the input file.

Levels number of levels in the wavelet decomposition.

HVS_Flag (Human Visual System) Flag - if set to 1, the quantization is varied in the different levels to approximate the human visual system sensitivity to different frequencies. At a given compression ratio, this should result in smoother images that technically have poorer fidelity but are more pleasing to the eye.

MasterBin the quantization parameter by which the wavelet coefficients are divided. A larger value means coarser approximations of the wavelet coefficients, poorer image fidelity, and higher compression ratio; a smaller value means the reverse.

out_file is the name of the output file.

The resulting output file can be read from and written to as transparantly as any other AVW_ImageFile format file. Each slice is individually compressed.

RETURN VALUES

If successful *AVW_WaveletCompressImageFile()* returns an *AVW_SUCCESS*. On failure it returns *AVW_FAIL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_WaveletCompressImageFile will fail if one or more of the following are true:

BADMAL
Malloc Failed. A memory allocation failed.

ILLIMG
Illegal Image. The images are not all the same dimension.

ILLDT
Illegal Datatype. Does not support multi-band datatypes such as AVW_COLOR or AVW_COMPLEX.

SEE ALSO

AVW_WaveletCompressAndDecompressImage(), AVW_WaveletCompressImage(), AVW_Image

REFERENCE

Manduca, A. (1997) "Compressing Images with Wavelet / Subband Coding", IEEE Engineering in Medicine and Biology, 14(5), 639-646.

NAME

AVW_WireGrid – creates a wire-mesh plot from an image

SYNOPSIS

```
#include "AVW.h"

AVW_PointList2 *AVW_WireGrid(image, matrix, xspace, yspace, out_plist)
AVW_Image *image;
AVW_Matrix *matrix;
int xspace, yspace;
AVW_PointList2 *out_plist;
```

DESCRIPTION

AVW_WireGrid() generates a wire-mesh plot in the form of an *AVW_PointList2,* from an *AVW_Image, image,* an *AVW_Matrix, matrix,* and two integer values, *xspace* and *yspace.* The *image->Width* and *image->Height* determine the size of the wiregrid, and the intensity values at each pixel within the image, determine the amount that the grid is raised or lowered at the pixel position.

Matrix is used to transform the 2-D image into 3-D space. *AVW_RotateMatrix()* is used to specify the angle at which the wire grid is viewed from. Wiregrids are best viewed at non-orthogonal angles like 30-45 degrees. *AVW_ScaleMatrix()* can be used to enlarge or shrink the grid on one or all of the axes. A good starting point is probably 1. for X & Y and .1 for Z. *AVW_TranslateMatrix()* is normally used to translate the points returned to the center of an output window. *AVW_Wiregrid()* uses the center of the input *image* as point (0,0). *AVW_TranslateMatrix(mat, win_width/2., win_height/2., 0.);* could be used to translate the point list returned to be displayed at the center of a window which has dimensions of win_width by win_height.

Xspace and *yspace* determine at which points along the grid the image is sampled. Small values causes the *image* to be sampled very finely. Large values result in a very course sampling. Very large and very small values will not produce very good results. Start with values in the 5-10 range.

Out_plist is provided as a method of reusing an existing *AVW_PointList2.*

RETURN VALUES

If successful *AVW_WireGrid()* returns an *AVW_PointList2.* On failure it returns *NULL* and sets *AVW_ErrorNumber* and *AVW_ErrorMessage* to values corresponding to the cause of the failure.

ERRORS

AVW_WireGrid() will fail if:

ILLDT
Data type is not defined or supported.

ILLIMG
An illegal image was passed to the function.

SEE ALSO

AVW_PointList2, AVW_Image

NAME

AVW_WriteHistogram – writes a histogram to a file

SYNOPSIS

#include "AVW_Histogram.h"

```
int AVW_WriteHistogram(histo, filename)
AVW_Histogram *histo;
char *filename;
```

DESCRIPTION

AVW_WriteHistogram() writes the *AVW_Histogram*, *histo*, to a file called *filename.*

RETURN VALUES

Upon success *AVW_WriteHistogram()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned, and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

AVW_WriteHistogram() will fail if one or more of the following is true:

BDOPEN
Bad Open. Failure opening file for writing.

ILLHIS
Illegal Histogram.

SEE ALSO

AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_DestroyHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_MatchVolumeHistogram(), AVW_MatchImageHistogram(), AVW_NormalizeHistogram(), AVW_ReadHistogram(), AVW_VerifyHistogram(), AVW_Histogram

NAME

AVW_WriteImageFile – writes an image to an image file

SYNOPSIS

```
#include "AVW_ImageFile.h"

int AVW_WriteImageFile (imgfile, image)
AVW_ImageFile *imgfile;
AVW_Image     *image;
```

DESCRIPTION

AVW_WriteImageFile() writes the *AVW_Image* to an *AVW_ImageFile.* The *AVW_ImageFile* must have been opened with the write mode specified.

RETURN VALUES

Upon success *AVW_WriteImageFile()* returns *AVW_SUCCESS.* On failure *AVW_FAIL* is returned, and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

AVW_WriteImageFile() will fail if one or more of the following are true:

ICPIMG
: Incompatible Image. The image and imgfile are incompatible with regard to image size and / or data type.

BDWRTE
: Bad Write. Error writing image.

NOSPACE
: No Space On Device. Disk is full.

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendExternalLibs(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_SeekImageFile(), AVW_WriteSubVolumeDescription(), AVW_WriteVolume(), AVW_Image, AVW_ImageFile

NAME

AVW_WriteSubVolumeDescription – writes a subvolume description file

SYNOPSIS

#include "AVW_ImageFile.h"

```
int AVW_WriteSubVolumeDescription (outname, infile, involnum, subv, info)
char *outname;
char *infile;
int involnum;
AVW_Rect3 *subv;
char *info;
```

DESCRIPTION

AVW_WriteSubVolumeDescription() writes a subvolume description file. A subvolume description file is a type of *AVW_VolumeFile* which contains a description of a subvolume of interest contained in another image file. The description file is opened as if it were an *AVW_VolumeFile.*

Outname is the name of the output file to be written.

Infile is the name of the image file containing the described subvolume. *Infile* can be any image file format supported by *AVW*, including the *AVW_VolumeFile* format.

Involnum is the volume number containing the specified subvolume. Acceptable values are from 0 to the number of volumes in *infile* less 1.

Subv specifies the position and size of the subvolume.

Info is an *AVW_Info* string whose elements will be written in the file. Set *info* to NULL if no additional info is to be written to the file.

RETURN VALUES

Upon success *AVW_WriteSubVolumeDescription()* returns *AVW_SUCCESS*. On failure *AVW_FAIL* is returned, and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

AVW_WriteSubVolumeDescription() will fail if one or more of the following is true:

BDOPEN
Unable to open infile.

ICPVOL
Subv is incompatible with infile.

BDVLNM
Involnum incompatible with infile.

BDCRT
Error creating outfile.

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendExternalLibs(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume(), AVW_Rect3

NAME

AVW_WriteVolume – writes a volume to an image file

SYNOPSIS

```
#include "AVW_ImageFile.h"

int AVW_WriteVolume (imgfile, volnum, volume)
AVW_ImageFile *imgfile;
int volnum;
AVW_Volume *volume;
```

DESCRIPTION

AVW_WriteVolume() writes the *AVW_Volume, volume,* to an *AVW_ImageFile, imgfile.* The *AVW_ImageFile* must have been opened with the write mode specified.

Volnum is the volume number to be written. Volumes are numbered from 0 to imgfile->NumVols-1. Thus if the first volume of a file is to be written, specify 0.

RETURN VALUES

Upon success *AVW_WriteVolume()* returns *AVW_SUCCESS.* On failure *AVW_FAIL* is returned, and *AVW_ErrorNumber* and *AVW_ErrorMessage* are set to values indicating the cause of error.

ERRORS

AVW_WriteVolume() will fail if one or more of the following is true:

ICPVOL
Incompatible Volume.The volume and imgfile are incompatible with regard to dimensions and / or data type.

BDWRTE
Bad Write. Error writing image.

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendExternalLibs(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteSubVolumeDescription(), AVW_ImageFile, AVW_Volume

NAME

EXTEND.conf– configuration file for loading AVW shared libraries

SYNOPSIS

$AVW / app-defaults / EXTEND.conf

DESCRIPTION

The file *EXTEND.conf* provides a list of shared libraries and configuration routines, and optionally other external libraries which may need to be loaded to resolve all references to the listed library. This file is opened and read by the function *AVW_ExtendExternalFormats()* which executes the steps necessary to link the named libraries into a running AVW based program. This function is designed to implement loading of image file extensions to AVW, to avoid the need to recompile each application if a new file format is added to the library. If the extensions are loaded into shared libraries and referenced in the the *EXTEND.conf* configuration file, the *AVW_ExtendExternalFormats()* function will load and install them in a running program.

The format of the file consists of lines containing 4 entries:

Target-System Is the name of AVW known system target types. Currently these are: SPARC, SGI5, ALPHA, PC_NT, HP. If the extension applies to all versions, the keyword "ALL" can be used.

Library-Name Is the name of a shared library containing functions to be linked into the running AVW based program. For most systems, this library is found in the path described in the environment variable LD_LIBRARY_PATH and / or / usr / lib.

Initialization-Function If the listed library is located and loaded, the library is searched for a function of this name, and if found, this function is executed with no arguments. Any return value is discarded.

Other-Libraries In some cases, AVW extensions will rely on other libraries to resolve internal references. (For example, a image format extension for movie files may require movie libraries provided by another vendor). This field contains the single character N if no other libraries are required, else it contains a comma separated list of other libraries.

EXAMPLES

To include an image file extension for AVW for Compuserve GIF image files, which has been compiled and placed in a library called avwGIF.so, and is initialized into the application by calling the function "extend_for_gif()" from the shared library, and no other libraries are required:

```
ALL avwGIF.so     extend_for_gif   N
```

To include a multimedia image file format implemented on SGI only into AVW applications which is compiled into a library called mmedia.so, and is initialized by a call to "init_multimedia()" in the library, and which depends on vendor provided libraries libmovie.so, libaudio.so,
(located in / usr / lib):

```
SGI6 libavwSGIMOVIE.so avw_extend_for_sgi_movie libmovie.a,libaudio.a
```

SEE ALSO

AVW_ExtendExternalLibs(), AVW_ExtendImageFile(), AVW_ExtendIO

FILES

This file is sought in the following search order:

$AVW/app-defaults/EXTEND.conf (default) /home/AVW/app-defaults/EXTEND.conf a file specified by the environment variable AVW_EXTEND

NOTES

There must be 4 fields on each line.

Comments are NOT allowed in this file.

There must not be spaces in the comma separated list of other libraries.

NAME

AVW_2DShapeStats – 2D Shape Stats Structure

SYNOPSIS

```
typedef struct
        {
        float Area;
        float Perimeter;
        float MERAngle;
        float MERArea;
        float MERAspect;
        float RFF;
        float Circularity;
        AVW_FPoint2 Centroid;
        AVW_FPoint2 MER1;
        AVW_FPoint2 MER2;
        AVW_FPoint2 MER3;
        AVW_FPoint2 MER4;
        } AVW_2DShapeStats;
```

DESCRIPTION

The *AVW_2DShapeStats* structure provides an easy way to return shape statistics from *AVW* functions.

Area - count of pixels.

Perimeter - distance around the shape.

MERAngle - angle in degrees of Minimum Enclosing Rectangle.

MERArea - area of the Minimum Enclosing Rectangle.

MERAspect - aspect ratio of the Minimum Enclosing Rectangle.

RFF - Rectangular Fit Factor.

Circularity - ratio of region perimeter sqaured to its area.

Centroid - average of x and y coordinates in the region.

MER1, *MER2*, *MER3* and *MER4* are the corner coordinates of the Minimum Enclosing Rectangle.

LOCATION

AVW_Measure.h

SEE ALSO

AVW_FPoint2, AVW_Compute2DShapeStats(), AVW_ComputeCircularity(), AVW_ComputeImageCentroid(), AVW_ComputeMER(), AVW_ComputePerimeter(), AVW_ComputeRFF()

NAME

AVW_Colormap – Colormap Structure

SYNOPSIS

```
typedef struct
   {
   int Size;
   unsigned char *Red;
   unsigned char *Green;
   unsigned char *Blue;
   } AVW_Colormap;
```

DESCRIPTION

The *AVW_Colormap* structure provides an easy way to pass colors to and from *AVW* functions.

Size indicates the number of colors defined by this colormap.

Red, *Green* and *Blue* are pointers to the arrays.

LOCATION

AVW.h

SEE ALSO

AVW_CreateColormap(), AVW_CopyColormap(), AVW_DestroyColormap(), AVW_DESTROYCOLORMAP(), AVW_Image, AVW_Volume

NAME

AVW_Complex – Complex Number Structure

SYNOPSIS

```
typedef struct
   {
   float Real, Imaginary;
   } AVW_Complex;
```

DESCRIPTION

The *AVW_Complex* structure provides an easy way to pass and return Complex numbers.

Real contains the "real" portion of the complex numbers.

Imaginary contains the "imaginary" portion of the complex numbers.

LOCATION

AVW.h

NAME

AVW_CompositeInfo – Composite Information Structure

SYNOPSIS

```
typedef struct
{
        int RedType;
        int GreenType;
        int BlueType;
        int AlphaType;
        AVW_FPointList2* Red;
        AVW_FPointList2* Green;
        AVW_FPointList2* Blue;
        AVW_FPointList2* Alpha;
} AVW_CompositeInfo;
```

DESCRIPTION

The *AVW_CompositeInfo* structure provides an easy way to pass composite information to the *AVW_RenderParameters* structure.

RedType specifies the curve type used to interpolate the red color values. *RedType* is one of *AVW_LINEAR* or *AVW_SPLINE. Red* contains the Intensity and Red value coordinate used to generate the appropriate color during Volume Compositing. The X coordinate of each point corresponds to voxel intensity and the Y value correspondes to the value of the Red Component of the color at that intensity. The Red component must be in the range 0.0 to 1.0.

GreenType specifies the curve type used to interpolate the green color values. *GreenType* is one of *AVW_LINEAR* or *AVW_SPLINE. Green* contains the Intensity and Green value coordinate used to generate the appropriate color during Volume Compositing. The X coordinate of each point corresponds to voxel intensity and the Y value correspondes to the value of the Green Component of the color at that intensity. The Green component must be in the range 0.0 to 1.0.

BlueType specifies the curve type used to interpolate the blue color values. *BlueType* is one of *AVW_LINEAR* or *AVW_SPLINE. Blue* contains the Intensity and Blue value coordinate used to generate the appropriate color during Volume Compositing. The X coordinate of each point corresponds to voxel intensity and the Y value correspondes to the value of the Blue Component of the color at that intensity. The Blue component must be in the range 0.0 to 1.0.

AlphaType specifies the curve type used to interpolate the Alpha values. *AlphaType* is one of *AVW_LINEAR* or *AVW_SPLINE. Alpha* contains the opacity value coordinate used to generate the appropriate opacity during Volume Compositing. The X coordinate of each point corresponds to voxel intensity and the Y value corresponds to the value of the Alpha Component at that intensity. The Alpha component must be in the range 0.0 to 1.0.

LOCATION

AVW_CompositeInfo.h

SEE ALSO

AVW_FPointList2, AVW_RenderParameters

NAME

AVW_ContourSurface – Contour Surface Structure

SYNOPSIS

```
typedef struct
{
AVW_MultiList2        *Interior;
AVW_MultiList2        *Exterior;

AVW_Point3 MinExtent;
AVW_Point3 MaxExtent;

unsigned int StartSlice;
unsigned int EndSlice;
unsigned int TotalSlices;
double LayerThickness;
} AVW_ContourSurface;
```

DESCRIPTION

The *AVW_ContourSurface* structure represents the surface of an object as a series of stacked edges or contours. This two and a half dimensional representation of a surface is often used by rapid prototyping applications.

The *Interior* and *Exterior* multilist arrays contain pointers to a single *AVW_MultiList2* structure for each data slice. The arrays are indexed from 0 to *TotalSlices - 1*. The *AVW_MultiList2s* that comprise *Interior* contain the interior edges of the object, while those that comprise *Exterior* contain the exterior edges. Each edge is a series of X,Y points. The Z values begin with *StartSlice* and end with *EndSlice;* Thus the edges found in *Exterior[0]* would have a Z value of *StartSlice* and those found in *Exterior[TotalSlices - 1]* would have a Z value of *EndSlice.*

LayerThickness gives the "thickness" of each contour. It is calculated by *AVW_SliceVolume* and is currently limited to the resolution of the original volumetric data.

LOCATION

AVW_Model.h

SEE ALSO

AVW_DestroyContourSurface(), AVW_SaveContourSurface(), AVW_SliceVolume()
AVW_MultiList2

NAME

AVW_ExtendIO – Extend I/O Structure

SYNOPSIS

```
typedef struct
   {
   char          *Extension;
   char          *Description;
   int           MagicNumber;
   int           Properties;
   AVW_ImageFile *(*Open)();
   int           (*Seek)();
   AVW_Image     *(*Read)();
   int           (*Write)();
   int           (*Close)();
   AVW_ImageFile *(*Create)();
   int           (*Query)();
   } AVW_ExtendIO;
```

DESCRIPTION

The *AVW_ExtendIO* structure enables the user to extend the AVW_Image IO functions to support additional image file formats. The elements of the structure are set and the structure is then passed to AVW_ExtendImageFile().

Extension is a pointer to the extension (if any) by which this file may be easily identified .

Description is the name by which AVW will recognize this format. This is the value that is used by AVW_CreateImageFile() to specify the file format.

MagicNumber is a magic number (if any) by which this file format may be identified.

Properties specifies what properties are supported by this file format. These symbols may be combined with the bitwise OR "|" operator to set the supported properties flag.

AVW_SUPPORT_2D, AVW_SUPPORT_3D, AVW_SUPPORT_4D,
AVW_SUPPORT_READ, AVW_SUPPORT_WRITE,
AVW_SUPPORT_UNSIGNED_CHAR, AVW_SUPPORT_SIGNED_CHAR,
AVW_SUPPORT_UNSIGNED_SHORT, AVW_SUPPORT_SIGNED_SHORT,
AVW_SUPPORT_UNSIGNED_INT, AVW_SUPPORT_SIGNED_INT,
AVW_SUPPORT_FLOAT, AVW_SUPPORT_COMPLEX, AVW_SUPPORT_COLOR

Open() specifies the user supplied function which opens and returns an AVW_ImageFile structure for this format. AVW_OpenImageFile() passes its arguments to this function and returns what is returned by this routine.

Seek() specifies the user supplied function which performs image seeks for this format. AVW_SeekImageFile() passes its arguments to this function and returns what is returned by this function.

Read() specifies the user supplied function which reads and returns an AVW_ImageFile for this format. AVW_ReadImageFile() file passes its arguments to this function and returns what is returned by this function.

Write() specifies the user supplied function which writes an AVW_ImageFile for this format. AVW_WriteImageFile() passes its arguments to this function and returns what is returned by this function.

Close() specifies the user supplied function which closes an AVW_ImageFile for this format. AVW_CloseImageFile() passes its arguments to this function and returns what is returned by this function.

Create() specifies the user supplied function which creates an AVW_ImageFile for this format. AVW_CreateImageFile() passes its arguments to this function and returns what is returned by this function.

Query() specifies the user supplied function which tests whether a file is of this format. This function is called by AVW_OpenImageFile() to determine if a file is of this format. Given the name of the file; this function returns AVW_TRUE is the file is of the file format and AVW_FALSE if it is not. AVW_OpenImageFile() uses the routine to assign the appropriate image file IO routines once a file's data format is determined.

LOCATION

AVW_ImageFile.h

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendExternalLibs(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteSubVolumeDescription(), AVW_WriteVolume(), AVW_Image, AVW_ImageFile

NAME

AVW_FPoint2 – 2D (Float) Point Structure

SYNOPSIS

```
typedef struct
   {
   float X, Y;
   } AVW_FPoint2;
```

DESCRIPTION

The *AVW_FPoint2* structure provides an easy way to pass and return 2D floating point coordinates.

X and *Y* specify coordinate values.

LOCATION

AVW.h

SEE ALSO

AVW_Point2, AVW_IPoint2, AVW_FPoint3, AVW_FPointList2, AVW_AddFPoint2(), AVW_GetFPoint2(), AVW_Compute2DShapeStats(), AVW_ComputeImageCentroid(), AVW_CreateFPointList2(), AVW_InterpolatedPixel(), AVW_NearestNeighborPixel(), AVW_RemoveFPoint2()

NAME

AVW_FPoint3 – 3D (float) Point Structure

SYNOPSIS

```
typedef struct
   {
   float X, Y, Z;
   } AVW_FPoint3;
```

DESCRIPTION

The *AVW_FPoint3* structure provides an easy way to pass and return 3D float coordinates.

X, *Y*, and *Z* specify coordinate values.

LOCATION

AVW.h

SEE ALSO

AVW_FPoint2, AVW_Point3, AVW_IPoint3, AVW_FPointList3, AVW_AddFPoint3(), AVW_ComputeVolumeCentroid(), AVW_CreateFPointList3(), AVW_GetFPoint3(), AVW_InterpolatedVoxel(), AVW_NearestNeighborVoxel(), AVW_RemoveFPoint3()

NAME

AVW_FPointList2 – List of 2D (float) points

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfPoints;
   unsigned int MaximumPoints;
   unsigned int BlockSize;
   AVW_FPoint2 *Points;
   } AVW_FPointList2;
```

DESCRIPTION

The *AVW_FPointList2* structure provides an easy way to pass and return a list of 2D (float) points.

NumberOfPoints is the current number of points in the list.

MaximumPoints indicates the number of points this list can hold before automatic reallocation.

BlockSize indicates the number added to the current size when reallocation is necessary.

Points is a pointer to an array of *AVW_FPoint2s.*

LOCATION

AVW.h

SEE ALSO

AVW_FPoint2, AVW_AddFPoint2(), AVW_CreateFPointList2(), AVW_DestroyFPointList2(), AVW_DESTROYFPOINTLIST2(), AVW_GetFPoint2(), AVW_MakeFPointList2(), AVW_RemoveFPoint2(),

NAME

AVW_FPointList3 – List of 3D (float) points

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfPoints;
   unsigned int MaximumPoints;
   unsigned int BlockSize;
   AVW_FPoint3 *Points;
   } AVW_FPointList3;
```

DESCRIPTION

The *AVW_FPointList3* structure provides an easy way to pass and return a list of 3D (float) points.

NumberOfPoints is the current number of points in the list.

MaximumPoints indicates the number of points this list can hold before automatic reallocation.

BlockSize indicates the number added to the current size when reallocation is necessary.

Points is a pointer to an array of *AVW_FPoint3s.*

LOCATION

AVW.h

SEE ALSO

AVW_FPoint3, AVW_FPointList2, AVW_AddFPoint3(), AVW_CreateFPointList3(), AVW_DestroyFPointList3(), AVW_DESTROYFPOINTLIST3(), AVW_GetFPoint3() AVW_RemoveFPoint3()

NAME

AVW_FilterCoeffs – Coefficents Structure

SYNOPSIS

```
typedef struct
   {
   int Number;
   float *Coeffs;
   } AVW_FilterCoeffs;
```

DESCRIPTION

The *AVW_FilterCoeffs* structure provides an easy way to pass and return Coefficents.

Number indicates the total number of coefficents.

Coeffs is a pointer to an array of floating point coefficents.

LOCATION

AVW_Filter.h

SEE ALSO

AVW_CreateButterworthCoeffs(), AVW_CreateCircularMTF(), AVW_CreateCoeffs(), AVW_CreateGaussianCoeffs(), AVW_CreateSphericalMTF(), AVW_CreateStoksethMTF(), AVW_DestroyCoeffs(), AVW_DESTROYCOEFFS()

NAME

AVW_FullWidthHalfMax – Full width half max measurments.

SYNOPSIS

```
typedef struct
   {
          AVW_Point2 Start;
          AVW_Point2 End;
          double Maximum;
          AVW_Point2 MaximumPoint;
          double HalfMax;
          int StartIndex;
          AVW_FPoint2 HM_Start;
          int EndIndex;
          AVW_FPoint2 HM_End;
          double FWHM_Distance;
          double Mean;
          double StdDev;
   } AVW_FullWidthHalfMax;
```

DESCRIPTION

The *AVW_FullWidthHalfMax* structure contains all of the full width half max measurements which are usually computed by *AVW_ComputeFullWidthHalfMax().* The measurements contained in this structure are computed from a line profile which is contained in an *AVW_PointValueList.* Some of the measurements are related the *AVW_PointValueList.*

Start is the coordinates of the first point in the line profile.

End is is the coordinates of the last point in the line profile.

Maximum indicates the maximum value of the line profile.

MaximumPoint is the coordinates of the first occurrence of the *Maximum* value in the line profile.

HalfMax indicates the half max value of the line profile.

StartIndex indicates index of the line profile where the first half max occurrs.

HM_Start indicates the coordinates of the first half max.

EndIndex indicates index of the line profile where the second half max occurrs.

HM_End indicates the coordinates of the second half max.

FWHM_Distance specifies the distance, which is calibrated according to the voxel size, between *HM_Start* and *HM_End.*

Mean specifies the mean of all of the values in the line profile.

StdDev specifies the standard deviation of all of the values in the line profile.

LOCATION

AVW_Measure.h

SEE ALSO *AVW_Point2, AVW_FPoint2, AVW_PointValueList, AVW_ComputeFullWidthHalfMax() AVW_ComputeLineProfile()*

NAME

AVW_GradientPoint – Gradient Point Structure

SYNOPSIS

```
typedef struct
   {
   AVW_Point3 Location;
   AVW_Point3 Gradient;
   } AVW_GradientPoint;
```

DESCRIPTION

The *AVW_GradientPoint* structure holds the location and precalculated gradient information for a predetermined surface voxel.

Location indicates a voxels 3-D location.

Gradient contains the pre-calculated gradient information for this voxel.

LOCATION

AVW_Render.h

SEE ALSO

AVW_Gradients, AVW_Point3

NAME

AVW_Gradients – Gradients Structure

SYNOPSIS

```
typedef struct
   {
   int NumberOfGradients;
   int MaximumGradients;
   AVW_GradientPoint *GradientPoint;
   } AVW_Gradients;
```

DESCRIPTION

The *AVW_Gradients* structure is used to pass pre-determined surface information to and from *AVW* functions.

NumberOfGradients is the number of points in this surface.

MaximumGradients is the number of points that can be held in the *GradientPoint* array, before it must be realloc'd.

GradientPoint is an array of *AVW_GradientPoint's.*

LOCATION

AVW_Render.h

SEE ALSO

AVW_ExtractGradients(), AVW_DestroyGradients(), AVW_RenderGradients(), AVW_GradientPoint

NAME

AVW_Histogram – Histogram Structure

SYNOPSIS

```
typedef struct
        {
        double *Mem;
        double Max;
        double Min;
        double Step;
        unsigned int Bins;
        } AVW_Histogram;
```

DESCRIPTION

The *AVW_Histogram* structure provides an easy way to pass and return histograms.

Mem is a pointer to an array of doubles. Each value in the array indicates the number of points in this bin.

Max and *Min* indicate the values of the first and last bins.

Step is the increment between bins.

Bins is the total number of bins.

LOCATION

AVW_Histogram.h

SEE ALSO

AVW_ClearHistogram(), AVW_CreateHistogram(), AVW_DestroyHistogram(), AVW_FlattenImageHistogram(), AVW_FlattenVolumeHistogram(), AVW_GetImageHistogram(), AVW_GetVolumeHistogram(), AVW_MatchImageHistogram(), AVW_MatchVolumeHistogram(), AVW_NormalizeHistogram(), AVW_PreserveImageHistogram(), AVW_PreserveVolumeHistogram(), AVW_VerifyHistogram()

NAME

AVW_IPoint2 – 2D (Integer) Point Structure

SYNOPSIS

```
typedef struct
   {
   int X, Y;
   } AVW_IPoint2;
```

DESCRIPTION

The *AVW_IPoint2* structure provides an easy way to pass and return 2D integer point coordinates.

X and *Y* specify coordinate values.

LOCATION

AVW.h

SEE ALSO

AVW_IPoint3, AVW_FPoint2, AVW_Point2, AVW_IPointList2, AVW_AddIPoint2(), AVW_CreateIPointList2(), AVW_GetIPoint2() AVW_RemoveIPoint2()

NAME

AVW_IPoint3 – 3D (integer) Point Structure

SYNOPSIS

```
typedef struct
   {
   int X, Y, Z;
   } AVW_IPoint3;
```

DESCRIPTION

The *AVW_IPoint3* structure provides an easy way to pass and return 3D integer coordinates.

X, *Y*, and *Z* specify coordinate values.

LOCATION

AVW.h

SEE ALSO

AVW_IPoint2, AVW_FPoint2, AVW_FPoint3, AVW_Point2, AVW_Point3, AVW_IPointList3, AVW_AddIPoint3(), AVW_GetIPoint3(), AVW_RemoveIPoint3()

NAME

AVW_IPointList2 – List of 2D (integer) points

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfPoints;
   unsigned int MaximumPoints;
   unsigned int BlockSize;
   AVW_IPoint2 *Points;
   } AVW_IPointList2;
```

DESCRIPTION

The *AVW_IPointList2* structure provides an easy way to pass and return a list of 2D (integer) points.

NumberOfPoints is the current number of points in the list.

MaximumPoints indicates the number of points this list can hold before automatic reallocation.

BlockSize indicates the number added to the current size when reallocation is necessary.

Points is a pointer to an array of *AVW_IPoint2s.*

LOCATION

AVW.h

SEE ALSO

AVW_IPointList3, AVW_FPointList2, AVW_FPointList3, AVW_PointList2, AVW_PointList3, AVW_PointValueList, AVW_AddIPoint2(), AVW_CreateIPointList2(), AVW_DestroyIPointList2(), AVW_GetIPoint2(), AVW_DESTROYIPOINTLIST2(), AVW_RemoveIPoint2(),

NAME

AVW_IPointList3 – List of 3D (integer) points

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfPoints;
   unsigned int MaximumPoints;
   unsigned int BlockSize;
   AVW_IPoint3 *Points;
   } AVW_IPointList3;
```

DESCRIPTION

The *AVW_IPointList3* structure provides an easy way to pass and return a list of 3D (integer) points.

NumberOfPoints is the current number of points in the list.

MaximumPoints indicates the number of points this list can hold before automatic reallocation.

BlockSize indicates the number added to the current size when reallocation is necessary.

Points is a pointer to an array of *AVW_IPoint3s.*

LOCATION

AVW.h

SEE ALSO

AVW_IPoint3, AVW_IPointList2, AVW_AddIPoint3(), AVW_CreateIPointList3(), AVW_DestroyIPointList3(), AVW_GetIPoint3(), AVW_DESTROYIPOINTLIST3(), AVW_RemoveIPoint3()

NAME

AVW_Image – Image Structure

SYNOPSIS

```
typedef struct
   {
   void *Mem;
   int DataType;
   unsigned int Width;
   unsigned int Height;
   unsigned int BytesPerPixel;
   unsigned int BytesPerLine;
   unsigned int BytesPerImage;
   unsigned int PixelsPerImage;
   AVW_Colormap *Colormap;
   char *Info;
   unsigned int *YTable;
   } AVW_Image;
```

DESCRIPTION

The *AVW_Image* structure provides an easy way to pass images to and from *AVW* functions.

Mem is a pointer to the raw data. It is often necessary to typecast this pointer when using it.

Example: ptr = (unsigned char *) image->Mem

DataType specifies pixel data type. Types include all AVW data types: *AVW_UNSIGNED_CHAR*, *AVW_SIGNED_CHAR*, *AVW_UNSIGNED_SHORT*, *AVW_SIGNED_SHORT*, *AVW_UNSIGNED_INT*, *AVW_SIGNED_INT*, *AVW_FLOAT*, *AVW_COMPLEX*, *AVW_COLOR*.

Width and *Height* specify the dimensions of the image.

BytesPerPixel, *BytesPerLine*, *BytesPerImage*, and *PixelsPerImage* contain often used precalcuated values.

Colormap is a pointer to an *AVW_Colormap*.

Info is a pointer to an *AVW* information string which is used to store additional information about the image.

YTable is an array of pre-calculated offsets to the start of each line in the image. The following example shows how to get the starting address of the tenth line of the image.

```
ptr = (unsigned char *) image->Mem + image->YTable[9];
```

LOCATION

AVW.h

SEE ALSO

AVW_Colormap, *AVW_Volume*, *AVW_CreateImage()*, *AVW_DestroyImage()*, *AVW_DESTROYIMAGE()*

NAME

AVW_ImageFile – Image File Structure

SYNOPSIS

```
typedef struct
   {
   char *FileName;
   char *FileModes;

   int DataFormat;
   int DataType;

   unsigned int Width;
   unsigned int Height;
   unsigned int Depth;
   unsigned int NumVols;

   unsigned int BitsPerPixel;
   unsigned int BytesPerPixel;
   unsigned int BytesPerLine;
   unsigned int BytesPerImage;
   unsigned int BytesPerVolume;
   unsigned int BytesPerFile;
   unsigned int PixelsPerImage;
   unsigned int VoxelsPerVolume;
   unsigned int VoxelsPerFile;

   int CurrentSlice;
   int CurrentVolume;

   AVW_Colormap *Colormap;

   void *NativeData;
   char *Info;
   } AVW_ImageFile;
```

DESCRIPTION

The *AVW_ImageFile* structure provides a transparent way to read and write image files in a variety of supported formats.

FileName is a pointer to the name of this file.

FileModes is the mode string with which the file has been opened.

DataFormat is an index into a list of supported Data Formats.

DataType specifies what type of image(s) are contained in the file. Types include: *AVW_UNSIGNED_CHAR*, *AVW_SIGNED_CHAR*, *AVW_UNSIGNED_SHORT*, *AVW_SIGNED_SHORT*, *AVW_UNSIGNED_INT*, *AVW_SIGNED_INT*, *AVW_FLOAT*, *AVW_COMPLEX*, *AVW_COLOR*.

Width and *Height* specify the dimensions of the image(s).

Depth specifies the number of images in a volume.

NumVols specifies the number of volumes in the file.

BitsPerPixel, *BytesPerPixel*, *BytesPerLine*, *BytesPerImage*, *BytesPerVolume*, *PixelsPerImage*, *VoxelsPerVolume*, and *VoxelsPerFile* contain often used pre-calcuated values.

CurrentSlice and *CurrentVolume* indicate the volume and image at which the file is currently positioned to read or write.

Colormap is a pointer to an *AVW_Colormap*.

Info is a pointer to an *AVW* information string.

NativeData is a pointer to information about the image file specific to its particular format.

LOCATION

AVW_ImageFile.h

SEE ALSO

AVW_CloseImageFile(), AVW_CreateImageFile(), AVW_ExtendImageFile(), AVW_FormatSupports(), AVW_ListFormats(), AVW_MMapVolume(), AVW_OpenImageFile(), AVW_ReadImageFile(), AVW_ReadVolume(), AVW_SeekImageFile(), AVW_WriteImageFile(), AVW_WriteVolume()

NAME

AVW_Instructions – Instruction Structure

SYNOPSIS

```
typedef struct
        {
        int NumberOfOperations;
        unsigned short **Operations;
        int NumberOfConstants;
        char **ConstantNames;
        int NumberOfVariables;
        char **VariableNames;
        char **FileName;
        char **DataFormat;
        int *DataType;
        int *StartingVolume;
        int *VolumeIncrement;
        int *VolumesToProcess;
        int *StartingSlice;
        int *SliceIncrement;
        int *SlicesToProcess;
        int *SlicesPerVolume;
        int *MaxMinOption;
        int (*Communicator)();
        } AVW_Instructions;
```

DESCRIPTION

The *AVW_Instructions* structure provides an easy way to pass and return instructions.

LOCATION

AVW_Parse.h

SEE ALSO

AVW_Parse(), AVW_DoInstructions(), AVW_DestroyInstructions()

NAME

AVW_IntensityStats – Intensity Statistics Structure

SYNOPSIS

```
typedef struct
        {
        double Mean;
        double StandardDeviation;
        double Variance;
        double Sum;
        double SumOfSquares;
        unsigned long NumberOfVoxels;
        double Area;
        double Volume;
        double HighestIntensity;
        AVW_Point3 HighestPoint;
        double LowestIntensity;
        AVW_Point3 LowestPoint;
        double RangeMaximum;
        double RangeMinimum;
        double MeanInRange;
        double StandardDeviationInRange;
        double VarianceInRange;
        double SumInRange;
        double SumOfSquaresInRange;
        unsigned long NumberBelowRange;
        unsigned long NumberInRange;
        unsigned long NumberAboveRange;
        double AreaInRange;
        double VolumeInRange;
        double BrightnessAreaProduct;
        } AVW_IntensityStats;
```

DESCRIPTION

The *AVW_IntensityStats* structure provides an easy way to return intensity statistics from *AVW* functions.

LOCATION

AVW_Measure.h

SEE ALSO

AVW_Point3, *AVW_ComputeImageIntensityStats()*, *AVW_ComputeVolumeIntensityStats()*

NAME AVW_Line2 – 2D Line Structure

SYNOPSIS

```
typedef struct
   {
   AVW_Point2 Start, End;
   } AVW_Line2;
```

DESCRIPTION The *AVW_Line2* structure provides an easy way to pass and return 2D lines.

Start and *End* specify end points of the line.

LOCATION *AVW.h*

SEE ALSO *AVW_Point2, AVW_Line3*

NAME AVW_Line3 – 3D Line Structure

SYNOPSIS

```
typedef struct
   {
   AVW_Point3 Start, End;
   } AVW_Line3;
```

DESCRIPTION The *AVW_Line3* structure provides an easy way to pass and return 3D lines.

Start and *End* specify end points of the line.

LOCATION *AVW.h*

SEE ALSO *AVW_Point3, AVW_Line2*

NAME AVW_List – List Structure

SYNOPSIS

```
typedef struct
   {
   int NumberOfEntries;
   char **Entry;
   } AVW_List;
```

DESCRIPTION The *AVW_List* structure provides an easy way to pass and return lists.

NumberOfEntries indicates the total number of entries in the list.

Entry is a pointer to an array of char pointers.

LOCATION *AVW.h*

SEE ALSO *AVW_DestroyList(), AVW_ListFormats(), AVW_ListInfo()*

NAME

AVW_MatchParameters – Surface Match Parameters Structure

SYNOPSIS

```
typedef struct
        {
        int SamplePoints;
        int Centroid;
        float TranslationX, TranslationY, TranslationZ;
        float TranslationRange;

        float RotationPrecession, RotationNutation, RotationSpin;
        float RotationRange;
        float RotationInterval;
        } AVW_MatchParameters;
```

DESCRIPTION

The *AVW_MatchParameters* structure controls the initial position of the match surface and the range of translational and rotational motion allowed during the search for the best match.

SamplePoints determines how many randomly-selected points on the match surface are used for the search. Larger numbers give better matches, but require more search time.

Centroid, if nonzero, indicates that the *TranslationX*, *TranslationY* and *TranslationZ* values are to be ignored, and the centroids of the base and match surface used for the initial search position.

TranslationX, *TranslationY* and *TranslationZ* specify the starting match position if *Centroid* is zero.

TranslationRange specifies the maximum translation (in voxels) allowed in the search.

RotationPrecession, *RotationNutation*, and *RotationSpin* indicate the initial 3D rotational search position.

RotationRange specifies the total allowable range of rotation (about all three axes) for the search.

RotationInterval specifies the number of degrees per step in the final (high-resoluton) search for the best matching position. The default is one degree.

LOCATION

AVW_SurfaceMatch.h

SEE ALSO

AVW_MatchSurfaces(), *AVW_MatchResults*

NAME

AVW_MatchResult – Surface Match Results Structure

SYNOPSIS

```
typedef struct
        {
        AVW_Matrix *Matrix;
        float MeanSquareDistance;
        AVW_MatchParameters *NextInput;
        } AVW_MatchResult;
```

DESCRIPTION

The *AVW_MatchResult* structure provides the scpecification of the best matching position found as well as a modified *AVW_MatchParameters* structure set up for a more rigorous (i.e. finer resolution) search over a restricted search range.

Matrix contains the entire translational, rotational, and scaling transformation required to take the Match image into the coordinate space of the base image.

MeanSquareDistance is the residual error of the current match. Since this absolute number is dependent upon the specific surfaces used for matching, it is only usefull as a comparison of different matches of the same surfaces.

NextInput is a *AVW_MatchParameters* structure set up for a more rigorous search over a restricted search range.

LOCATION

AVW_SurfaceMatch.h

SEE ALSO

AVW_MatchParameters, AVW_MatchSurfaces()

NAME

AVW_MatchVoxelParams – Voxel Match Parameters Structure

SYNOPSIS

```
typedef struct
        {
        double Ftol;
        double Ptol;
        int   Iterations;
        int   Interpolate;
        int   Smpl1to1[3];
        int   Smpl2to1[3];
        int   Smpl4to1[3];
        int   Smpl8to1[3];
        double InitGuess[6];
        double SearchLength[6];
} AVW_MatchVoxelParams;
```

DESCRIPTION

The *AVW_MatchVoxelParams* structure controls the convergence tolerance for the searching strategy, total number of iterations, the interpolation method used to evaluate the transformed volume voxels, the sub-sampling used for histogram evaluation in each scaling level, initial position of the match surface and the range of translational and rotational motion allowed during the search for the best match.

Ftol determines the convergence tolerance for the search strategy. If the change in the cost-function is smaller than this value, the search will stop assuming this is the minimum.

Ptol specifies the transformation parameters convergence tolerance. That is, if the total change in all of the 6 transformation parameters (X,Y,Z rotations and translations) is less than Ptol for a number of successive iterations the subroutine will terminate the search.

Interpolate specifies whether the transformed image will be computed with bilinear (AVW_TRUE) or nearest neighbor (AVW_FALSE) interpolation.

Smpl1to1, *Smpl2to1*, *Smpl4to1* and *Smpl8to1* Specifies sampling in the X, Y and Z directions. (i.e., if X and Y are set to 3, and Z is 1, than the calculation of the cost function will use every third voxel in the X and Y directions, and all of the slices in the Z direction.

The search is done in stages, first on a volume scaled to a size of 8:1, than 4:1, 2:1 and finally 1:1. The sampling values can be specified for each one of these scaling stages (Smpl8to1, Smpl4to1, Smpl2to1 and Smpl1to1 respectively) . If scaling to a certain size would cause a the volume to become to small the stage will be skipped. It is the users responsibility to assign reasonable values for sampling. Values which cause the use of only a very small number of voxels, will lead to non-accurate results. To determine a good experimental value, define the sample in such a way that there is 30 to 50 points in each direction. This will usually lead to good results with the best possible computation time.

InitGuess specifies the initial position (X,Y,Z rotation and translations) of the match volume.

SearchLength defines the problem characteristic scale in X, Y, Z rotation and translation. Theses parameters limit the distance of the search algorithm.

LOCATION *AVW_MatchVoxels.h*

SEE ALSO *AVW_InitializeMatchVoxelParams(), AVW_DestroyMatchVoxelParams(), AVW_MatchVoxels().*

NAME

AVW_Matrix – Transformation Matrix Definition

SYNOPSIS

```
typedef struct
   {
   unsigned int Rows;
   unsigned int Columns;
   double **Matrix;
   } AVW_Matrix;
```

DESCRIPTION

The *AVW_Matrix* structure provides an easy way to pass and return a transformation matrix.

Rows and *Columns* specify the dimensions of the matrix. These values will almost always be 4 and 4.

Matrix is a double dimensioned array which allows access to each element within the structure. Example: mat->Matrix[1][2] returns the value in the row 1 and column 2.

LOCATION

AVW.h

SEE ALSO

AVW_CreateMatrix(), AVW_DestroyMatrix(), AVW_RotateMatrix(), AVW_CopyMatrix(), AVW_InvertMatrix(), AVW_MakeMatrixFrom3Points(), AVW_MakeMatrixFromAxis(), AVW_MirrorMatrix(), AVW_MultiplyMatrix(), AVW_RotateMatrix(), AVW_ScaleMatrix(), AVW_SetIdentityMatrix(), AVW_TranslateMatrix(), AVW_TransformImage(), AVW_TransformVolume()

NAME

AVW_MergedMap – Merged Map Structure

SYNOPSIS

```
typedef struct
   {
           int NumberMerged;
           AVW_Image *Image;
           AVW_Matrix **Matrix;
           } AVW_MergedMap;
```

DESCRIPTION

The *AVW_MergedMap* structure is filled in the *AVW_MergeRendered()* function. It contains information about where each pixel in the output image came from and what *AVW_Matrix* was used to create it.

NumberMerged is the number of images merged to produce the output.

Image is an *AVW_Image* which indicates which image each pixel came from.

Matrix is an array of *AVW_Matrix's* which were used in creating the original pixels before merging.

LOCATION

AVW_MergedMap.h

SEE ALSO

AVW_MergeRendered(), AVW_DestroyMergedMap(), AVW_Image, AVW_Matrix, AVW_RenderedImage

NAME

AVW_MultiList2 – List of AVW_PointList2s

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfLists;
   AVW_PointList2 **Lists;
   } AVW_MultiList2;
```

DESCRIPTION

The *AVW_MultiList2* structure provides an easy way to pass and return a list of AVW_PointList2s.

NumberOfList is the current number of lists.

Lists is a pointer to an array of *AVW_PointList2s.*

LOCATION

AVW.h

SEE ALSO

AVW_DestroyMultiList2(), AVW_PointList2

NAME

AVW_Object – Object Structure

SYNOPSIS

```
typedef struct
        {
        char Name[32];
        int DisplayFlag;
        unsigned char TransformFlag;
        unsigned char MirrorFlag;
        unsigned char StatusFlag;
        unsigned char NeighborsUsedFlag;
        int Shades;
        int StartRed, StartGreen, StartBlue;
        int EndRed, EndGreen, EndBlue;
        int XRotation, YRotation, ZRotation;
        int XTranslation, YTranslation, ZTranslation;
        int XCenter, YCenter, ZCenter;
        int XRotationIncrement, YRotationIncrement, ZRotationIncrement;
        int XTranslationIncrement, YTranslationIncrement, ZTranslationIncrement;
        short int MinimumXValue, MinimumYValue, MinimumZValue;
        short int MaximumXValue, MaximumYValue, MaximumZValue;
        float Opacity;
        int OpacityThickness;
        int Dummy;
        } AVW_Object;
```

DESCRIPTION

The *AVW_Object* structure contains all the attributes for each object in a *AVW_ObjectMap.*

Name specifies an identifier for this object.

DisplayFlag specifies whether this object will be visable or not. If this flag is set to *AVW_FALSE,* all voxels of this object type will be ignored during the next render.

TransformFlag enables and disables the special object transformation parameters. A value of *zero (0), indicates that the Rotation, Translation, Mirror,* and *Region* parameters are ignored. A value of *one (1),* causes a seperate rendering pass for this object where the transformations are applied. A value of *two (2),* indicates that this objects should be rendered in both the untransformed and the transformed rendering pass.

MirrorFlag specifies an axis which this object is mirrored against. [See TransformFlag]

StatusFlag is currently unused.

NeighborsUsedFlag indicates how neighbors are processed.

Shades indicates the number of shades to allow for this object. A total of 256 (250 in AnalyzeAVW) total shades are available.

StartRed, StartGreen, and *StartBlue* specifies the starting color for this object. These values range from 0-255. Black is 0, 0, 0. Red is 255, 0, 0. etc... AnalyzeAVW normally defaults this to 10% of the ending color.

EndRed, EndGreen, and *EndBlue* specifies the ending color.

XRotation, YRotation, and *ZRotation* indicates the amount of rotation to apply specifically to this object. [See TransformFlag]

XTranslation, YTranslation, and *ZTranslation* indicates the amount of translation to apply specifically to this object. [See TransformFlag]

XCenter, YCenter, and *ZCenter* indicates the center of rotation for this object. [See TransformFlag]

XRotationIncrement, YRotationIncrement, and *ZRotationIncrement* specifies the amount of desired rotational change for this object during a SEQUENCE.

XTranslationIncrement, YTranslationIncrement, and *ZTranslationIncrement* specifies the amount of desired translational change for this object during a SEQUENCE.

MinimumXValue, MinimumYValue, MinimumZValue, MaximumXValue, MaximumYValue, and *MaximumZValue* specifies the minimum bounding box (Region) for this object. These values allow transformed objects to be rendered in less time. The *AVW_CalculateObjectRegions()* function can be used to set these values. [See Transform-Flag]

Opacity specifies a value from .001 (Very Transparent) to 1.000 (Opaque) for this object. This value is used when the render *Type* is set to *AVW_TRANSPARENCY_SHADING.*

OpacityThickness specifies a count of voxels for which the *Opacity* value is calculated. Normally this is set to one and only the 1st voxel of each surface encountered is processed. Setting this to a higher value allows the rendering to show object thicknesses, because thicker regions will obscure more of a object than thinner regions.

Dummy is just a space filler and is not used.

LOCATION *AVW_ObjectMap.h*

SEE ALSO *AVW_ObjectMap*

NAME

AVW_ObjectMap – Object Map Structure

SYNOPSIS

```
typedef struct
        {
        int Version;
        int NumberOfObjects;
        AVW_Volume *Volume;
        AVW_Object *Object[256];
        unsigned char ShowObject[256];
        unsigned char MinimumPixelValue[256];
        unsigned char MaximumPixelValue[256];
        int NeedsSaving;
        int NeedsRegionsCalculated;
        } AVW_ObjectMap;
```

DESCRIPTION

The *AVW_ObjectMap* structure provides an easy way to pass an object map to and from *AVW* functions.

Version specifies which version of a *.obj* file this *AVW_ObjectMap* was read from. This field is not used, but is for information purposes only.

NumberOfObjects indicates the current number of objects defined in this object map.

Volume is a pointer to an *AVW_Volume.* This volume contains a zero (0) in each voxel which has been defined as being part of *Object[0].* A one (1) for *Object[1],* and so on. This volume may be manipulated with the other routines.

Object is an array of pointer to *AVW_Object* structures.

ShowObject is an array of flags used internally to indicate whether an object is currently being rendered. This is for internal use only.

MinimumPixelValue and *MaximumPixelValue* are arrays which indicate at which position in a color lookup table each objects colors start or end. This is for internal use only.

NeedsSaving is a flag which can be checked by an interface program to determine if the ObjectMap should be saved before exiting or unloading.

NeedsRegionsCalculated is a flag which indicates that the ObjectMap data has changed and that *AVW_CalculateObjectRegions()* should be called to calculate the minimum bounding box (Region).

LOCATION

AVW_ObjectMap.h

SEE ALSO

AVW_Object, *AVW_Volume*

NAME

AVW_Point2 – 2D Point Structure

SYNOPSIS

```
typedef struct
   {
   short X, Y;
   } AVW_Point2;
```

DESCRIPTION

The *AVW_Point2* structure provides an easy way to pass and return 2D coordinates.

X and *Y* specify coordinate values.

LOCATION

AVW.h

SEE ALSO

AVW_Point3, AVW_IPoint2, AVW_FPoint2, AVW_PointList2, AVW_AddPoint2(), AVW_GetPixel(), AVW_GetPoint2() AVW_PutPixel(), AVW_RemovePoint2()

NAME

AVW_Point3 – 3D Point Structure

SYNOPSIS

```
typedef struct
   {
   short X, Y, Z;
   short padding;
   } AVW_Point3;
```

DESCRIPTION

The *AVW_Point3* structure provides an easy way to pass and return 3D coordinates.

X, *Y*, and *Z* specify coordinate values.

LOCATION

AVW.h

SEE ALSO

AVW_Point2, AVW_IPoint3, AVW_FPoint3, AVW_PointList3, AVW_AddPoint3(), AVW_GetPoint3() AVW_GetVoxel(), AVW_PutVoxel(), AVW_RemovePoint3()

NAME

AVW_PointList2 – List of 2D points

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfPoints;
   unsigned int MaximumPoints;
   unsigned int BlockSize;
   AVW_Point2 *Points;
   } AVW_PointList2;
```

DESCRIPTION

The *AVW_PointList2* structure provides an easy way to pass and return a list of 2D points.

NumberOfPoints is the current number of points in the list.

MaximumPoints indicates the number of points this list can hold before automatic reallocation.

BlockSize indicates the number added to the current size when reallocation is necessary.

Points is a pointer to an array of *AVW_Point2s.*

LOCATION

AVW.h

SEE ALSO

AVW_Point2, AVW_AddPoint2(), AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_GetPoint2() AVW_RemovePoint2()

NAME

AVW_PointList3 – List of 3D points

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfPoints;
   unsigned int MaximumPoints;
   unsigned int BlockSize;
   AVW_Point3 *Points;
   } AVW_PointList3;
```

DESCRIPTION

The *AVW_PointList3* structure provides an easy way to pass and return a list of 3D points.

NumberOfPoints is the current number of points in the list.

MaximumPoints indicates the number of points this list can hold before automatic reallocation.

BlockSize indicates the number added to the current size when reallocation is necessary.

Points is a pointer to an array of *AVW_Point3s.*

LOCATION

AVW.h

SEE ALSO

AVW_Point3, AVW_AddPoint3(), AVW_CreatePointList3(), AVW_DestroyPointList3(), AVW_GetPoint3() AVW_RemovePoint3()

NAME

AVW_PointValueList – List of 2D points w / values

SYNOPSIS

```
typedef struct
   {
   unsigned int NumberOfPoints;
   unsigned int MaximumPoints;
   unsigned int BlockSize;
   AVW_Point2 *Points;
   double *Values;
   } AVW_PointValueList;
```

DESCRIPTION

The *AVW_PointValueList* structure provides an easy way to pass and return a list of 2D points.

NumberOfPoints is the current number of points in the list.

MaximumPoints indicates the number of points this list can hold before automatic reallocation.

BlockSize indicates the number added to the current size when reallocation is necessary.

Points is a pointer to an array of *AVW_Point2s.*

Values is a pointer to an array of *doubles.*

LOCATION

AVW.h

SEE ALSO

AVW_Point2, AVW_AddPointValue(), AVW_ComputeLineProfile(), AVW_CreatePointValueList(), AVW_DestroyPointValueList(), AVW_GetPointValue() AVW_RemovePointValue()

NAME

AVW_RPParam – Tile Structure

SYNOPSIS

```
typedef struct
        {
        AVW_Volume  *MaskVolume;

        double          MaskValue;
        double          AngleResolution;

        int                     InterpolateFlag;
        int                     SubvolumeFlag;
        int                     Format;
        int                     Orientation;
        int                     HandleBlankSlices;
        int                     Connectivity;

        } AVW_RPParam;
```

DESCRIPTION

The *AVW_RPParam* structure provides a method of passing parameters to and from the AVW contour extraction routines.

Format specifies the output format and is used by *AVW_SaveContourSurface*. Supported values include:

AVW_HPGL_SURFACE – A modified form of HPGL plotter commands designed to support rapid prototyping or stereolithography machines
AVW_POGO_SURFACE – A compressed version of the binary SLC format. This format does not differentiate between internal and external boundaries.
AVW_SLC_SURFACE – The standard binary SLC format.
AVW_SSD_ASCII_SURFACE – The ASCII version of the standard Analyze SSD format.

SubvolumeFlag if set, *AVW_SliceVolume* will subvolume the dataset so that the object of interest is contained within a minimum enclosing volume. This may cause the data to be reoriented prior to contour extraction.

MaskValue specify the mask value of the object within the *AVW_Volume* whose contours are to be extracted.

InterpolateFlag if set, *AVW_SliceVolume* will subvolume the dataset using trilinear interpolation.

Orientation specifies which orthogonal axis will be used during contour extraction This parameter may be one of:
AVW_TRANSVERSE [default], *AVW_CORONAL*, or *AVW_SAGITTAL*

AngleResolution specifies the angle resolution used while determining the minimum enclosing box.

HandleBlankSlices if set, *AVW_SaveContourSurface* will repeat the previous slice's contours if the current slice is blank and the output file format supports slice repetition. Otherwise, *AVW_SaveContourSurface* will terminate on a blank slice if the output file format is unable to handle missing data. At present, this flag only effects

SLC formatted files.

Connectivity is used by *AVW_SliceVolume* to determine the connectivity of the extracted edges. It may be set to: *AVW_4_CONNECTED* or *AVW_8_CONNECTED.*

LOCATION *AVW_Model.h*

SEE ALSO *AVW_DestroyRPParam(), AVW_InitializeRPParam(), AVW_SliceVolume(), AVW_Volume*

NAME

AVW_Rect2 – 2D Rectangle Structure

SYNOPSIS

```
typedef struct
   {
   AVW_Point2 PointA, PointB;
   } AVW_Rect2;
```

DESCRIPTION

The *AVW_Rect2* structure provides an easy way to pass and return 2D rectangles.

PointA and *PointB* specify opposite corners.

LOCATION

AVW.h

SEE ALSO

AVW_Point2, AVW_Rect3, AVW_GetSubImage()

NAME

AVW_Rect3 – 3D Rectangle (Cube) Structure

SYNOPSIS

```
typedef struct
   {
   AVW_Point3 PointA, PointB;
   } AVW_Rect3;
```

DESCRIPTION

The *AVW_Rect2* structure provides an easy way to pass and return 3D rectangles (cubes).

PointA and *PointB* specify opposite corners.

LOCATION

AVW.h

SEE ALSO

AVW_Point3, AVW_Rect2, AVW_GetSubVolume()

NAME AVW_RenderParameters – Render Structure

SYNOPSIS

```
typedef struct
   {
   int Type;
   double ThresholdMinimum, ThresholdMaximum;
   int ClipLowX, ClipLowY, ClipLowZ;
   int ClipHighX, ClipHighY, ClipHighZ;
   int ClipPlaneMinimum, ClipPlaneMaximum;
   int ClipShading;
   int RenderWidth, RenderHeight, RenderDepth;
   int MaximumPixelValue, MinimumPixelValue;
   int SurfaceThickness;
   AVW_Matrix *Matrix;
   AVW_Matrix *LightMatrix;
   AVW_Image *RenderMask;
   int MaskValue;
   int DeleteDepth;
   double DeleteValue;
   double ScaleX, ScaleY, ScaleZ;
   int PerspectiveType;
   AVW_FPoint3 EyePosition;
   double XFieldOfViewAngle, YFieldOfViewAngle;
   double SpecularFactor;
   double SpecularExponent;
   int SurfaceSkip;
   int MIP_Weight;
   AVW_CompositeInfo *CompositeInfo;
   int BackgroundColor;
   double BackgroundValue;
   int RenderMode;
   int InteractiveObject;
   AVW_InternalParameters Internal;
   } AVW_RenderParameters;
```

DESCRIPTION

The *AVW_RenderParameters* structure provides a method of passing the large number of render parameters to and from the AVW rendering routines.

Type determines the type of raycasting preformed. The default is *AVW_GRADIENT_SHADING.* Possible values are: *AVW_USER_DEFINED, AVW_DEPTH_SHADING, AVW_GRADIENT_SHADING, AVW_VOLUME_COMPOSITING, AVW_MAX_INTENSITY_PROJECTION, AVW_SUMMED_VOXEL_PROJECTION, AVW_SURFACE_PROJECTION, AVW_TRANSPARENCY_SHADING,* and *AVW_DELETE_VOXELS,*

ThresholdMinimum and *ThresholdMaximum* specifies the range of voxels value to process.

ClipLowX, ClipLowY, ClipLowZ, ClipHighX, ClipHighY, and *ClipHighZ* specify the portion of the input volume that is processed.

ClipPlaneMinimum and *ClipPlaneMaximum* specify the starting and ending depths for the ray casting process.

ClipShading determines the type of shading that is used when the ray casting process begins at a voxel which is within the threshold range (and object enabled). The value is only used when the render *Type* is set to *AVW_GRADIENT_SHADING*. Possible values are: *AVW_CLIP_SHADED*, *AVW_CLIP_ACTUAL*, *AVW_CLIP_REMOVE_AND_RENDER*, and *AVW_CLIP_RENDER_AS_IS*. [Default = *AVW_CLIP_SHADED]*

RenderWidth and *RenderHeight* specify the dimension of the output image and buffers. *RenderDepth* determines the range of depths. *Matrix*. The *ScaleX*, *ScaleY*, and *ScaleZ* also need to be considered when determining the proper *RenderWidth*, *RenderHeight* and *RenderDepth*.

MaximumPixelValue, and *MinimumPixelValue* specify the output intensity range.

SurfaceThickness specify the number of voxels to sum during a Surface Projection.

SurfaceSkip specify the number of voxels to skip before summing begins during a Surface Projection.

MIP_Weight specify the weighting options used during a Maximum Intensity Projections. Options include: *AVW_NO_WEIGHTING*, *AVW_WEIGHT_BEFORE*, and *AVW_WEIGHT_AFTER*. *AVW_WEIGHT_BEFORE* indicates that before a voxel is checked to see if it is the maximum value, a weighting factor is applied. The weighting factor is determined by dividing the length of the ray left to cast, by it's total length. AVW_WEIGHT_AFTER determines the maximum voxel along the entire ray casting path and then applies the weightinh factor described above.

Matrix is an *AVW_Matrix* which specifies any rotation and translation that is applied to the input volume. Scale could also be specified as part of the matrix, but it's recommended that the *ScaleX*, *ScaleY*, and *ScaleZ* parameters be used for Scale.

LightMatrix is an *AVW_Matrix* which specifies the location of the light source.

RenderMask, if not *NULL*, restricts the rendering process to all voxels in the *RenderMask* image, which have a value of *MaskValue*. This must always be either NULL, or have a dimension of *RenderWidth* by *RenderHeight*.

DeleteDepth and *DeleteValue* are used in conjuction with the *AVW_DELETE_VOXELS* rendering *Type* and the *RenderMask* and *MaskValue* to delete voxels in the input volume.

DeleteDepth should be set to one of the following: *AVW_DELETE_ALL_THE_WAY*, *AVW_DELETE_SINGLE_LAYER*, *AVW_DELETE_SINGLE_VOXEL*, *AVW_DEFINE_ALL_THE_WAY*, *AVW_DEFINE_SINGLE_LAYER*, or *AVW_DEFINE_SINGLE_VOXEL*.

If the *DEFINE* values are used, objects within an object_map are redefined.

ScaleX, *ScaleY*, and *ScaleZ* specify the scale factors to be applied to each input axis during the rendering process. A value of 1.0 indicates no scaling is applied.

PerspectiveType specifies the type of rendering to be performed. *AVW_PERSPECTIVE_INT* renders the image using the voxels without interpolation, possibly resulting in "blocky" renderings. If *PerspectiveType* is set to *AVW_PERSPECTIVE_FLOAT*, the rendered image is generated using an on the fly interpolation rendering algorithm. If *PerspectiveType* is set to *AVW_PERSPECTIVE_OFF*, parallel rendering is performed. For perspective rendering, the render matrix is interperted as a viewing direction for the camera model, however the parallel rendering conventions are followed, i.e. the matrix is left handed, and the identity matrix provides a view along the positive Z axis with X increasing to the right and Y increasing in the vertical direction. Perspective rendering does not support scaling at this time. For anisotropic data, rescaling during loading is the best option.

EyePosition specifies the location of the camera model used to generate the rendering. This parameter is only used when *PerspectiveType* is set to *AVW_PERSPECTIVE_FLOAT* or *AVW_PERSPECTIVE_INT.* The *X, Y,* and *Z* coordinates of the *AVW_FPoint3* structure refer to the position of the camera relative to the center of the volume, thus a position of (0,0,0) in a particular volume is located at voxel (Width / 2, Height / 2, Depth / 2).

XFieldOfViewAngle and *YFieldOfViewAngle* specify the field of view (FOV) angle of the camera model used to generate the image. Increasing the FOV results in a zoom out effect, if the position remains the same. Decreasing the FOV results in a zoom in effect.

SpecularFactor specifies the ratio of gradient shading to specular shading. If *SpecularFactor* is set to 0, the rendering will be shaded entirely using gradient shading with no performance penalty. If *SpecularFactor* is set to .1, the rendering is shaded entirely using the specular shading model.

SpecularExponent specifies the degree of fall-off for specular shading. High values (about 10) result in very small specular highlights, while small values (about 1 or 2) result in diffuse highlights.

CompositeInfo specifies a pointer to a structure which contains compositing information. This must contain a valid pointer when the *Type is set to AVW_VOLUME_COMPOSITING.* See *AVW_CompositeInfo* for more information.

BackgroundColor and *BackgroundValue* are used to specify the background color or value. The rendering type, input volumes datatype, and whether an object map is loaded determines which value is used. If the output is a grayscale image, *BackgroundValue* is used. If the output is a color image, then *BackgroundColor* will be used. *Background Color* is a packed RGB value which can be produced with the *AVW_RGB* macro.

RenderMode is normally set to *AVW_RENDER_NORMAL,* but when set to *AVW_PREPARE_FOR_MOVE,* the *AVW_RenderVolume()* function will process the object specified in the *InteractiveObject* seperately and return the "visible surface" in the *AVW_RenderedImage* member called *InteractiveSurface.* This results in the input to *AVW_RenderVisibleSurface()* which produces output which can be combined using *AVW_MergeRendered()* with the rendering returned at the time the *InteractiveSurface* was produced. This entire process allows an interface to be build to interactively move and rotate objects. *RenderMode* can be set to *AVW_RERENDER_MOVED,* at the completion of the object transformation to rerender any missing data.

The *AVW* rendering routines also have a variety of other parameters and tables which are not available for user modification. These are stored in the *Internal* member of the AVW_RenderParameters structure.

LOCATION

AVW_Render.h

SEE ALSO

AVW_DestroyRenderParameters(), AVW_ExtractSurface(), AVW_InitializeRenderParameters(), AVW_RenderSurface(), AVW_RenderVolume(), AVW_FPoint3, AVW_CompositeInfo, AVW_Image, AVW_Matrix

NAME

AVW_RenderedImage – Rendered Image Structure

SYNOPSIS

```
typedef struct
   {
   int Width, Height, Depth;
   double MaximumPixelValue, MinimumPixelValue;
   AVW_Image *Image;
   AVW_Image *ZBuffer;
   AVW_Image *PBuffer;
   AVW_Volume *Volume;
   AVW_ObjectMap *ObjectMap;
   AVW_Matrix *Matrix;
   AVW_Matrix *InverseMatrix;
   int PerspectiveType;
   AVW_FPoint3 EyePosition;
   double XFieldOfViewAngle, YFieldOfViewAngle;
   AVW_MergedMap *MergedMap;
   AVW_VisibleSurface *InteractiveSurface;
   double ReservedForFuture1;
   double ReservedForFuture2;
   double ReservedForFuture3;
   } AVW_RenderedImage;
```

DESCRIPTION

The *AVW_RenderedImage* structure is provides an easy way for rendered images to be returned from or passed to *AVW* functions.

Width, and *Height* indicate the size of the rendered image.

Depth indicates the maximum depth value.

MaximumPixelValue and *MinimumPixelValue* specify the range of pixels possible in the output image.

Image contains the visual information that resulted from the rendering process.

ZBuffer contains depth information used in calculating ray lengths.

PBuffer contains depth information for points found.

Volume is a pointer to the input volume of the last rendering process.

ObjectMap is a pointer to the object map of the last rendering process.

Matrix is the matrix used during the creation of this rendered image.

InverseMatrix is the inverse of the matrix used to create this rendered image.

PerspectiveType indicates if one of the perspective types was used to generate this rendered image.

EyePosition indicates the eye position of the last perspective rendering.

XFieldOfViewAngle and *YFieldOfViewAngle* indicate the field of view for the last perspective rendering.

MergedMap is normally *NULL, but after a call to AVW_MergeRendered* it is filled with information about where pixels came from and how they were generated.

InteractiveSurface is normally *NULL*. It's filled only when *AVW_RenderVolume()* is used with the *InteractiveMode* set to *AVW_PREPARE_FOR_MOVE* or *AVW_RERENDER_MOVED*.

LOCATION *AVW_Render.h*

SEE ALSO *AVW_FPoint3, AVW_Image, AVW_Matrix, AVW_MergedMap, AVW_ObjectMap, AVW_Volume*

NAME

AVW_SampleSpec – Image or Volume Sample Specification

SYNOPSIS

```
typedef struct {
        int SampleType;
        int Interpolation;
        int XSamples,YSamples,ZSamples;
        double XStart,YStart,ZStart;
        double XEnd,YEnd,ZEnd;
        } AVW_SampleSpec;
```

DESCRIPTION

The *AVW_SampleSpec* structure provides the specification for an sample of 2D or 3D coordinate points to be used for voxel statistics registration functions.

SampleType specifies the type of sample. Currently defined are AVW_GRID_SAMPLE and AVW_RANDOM_SAMPLE.

Interpolation is any of the standard AVW interpolation types. If *Interpolation* is AVW_NEAREST_NEIGHBOR_INTERPOLATE, all coordinate values will be truncated to integer values, otherwise floating point (double) coordinates are calculated.

XSamples,YSamples,ZSamples indicates the sampling density in all three dimensions. The total number of coordinates calculated is *XSamples**YSamples*ZSamples.*

XStart,YStart,ZStart,XEnd,YEnd,ZEnd are starting and ending coordinates for the region to be sampled.

LOCATION

AVW_MatchVoxels.h

SEE ALSO

AVW_StepSearchSpec, AVW_SetupVolumeSample() AVW_SampleVolume()
AVW_SetupImageSample() AVW_SampleImage()

NAME

AVW_StepSearchSpec – 6-DOF Bounded Step Search Specification

SYNOPSIS

```
typedef struct {
        double XStep,YStep,ZStep;
        double XRStep,YRStep,ZRStep;
        int XBound,YBound,ZBound,XRBound,YRBound,ZRBound;
        } AVW_StepSearchSpec;
```

DESCRIPTION

The *AVW_StepSearchSpec* structure provides the specification for a bounded, stepwise extrema search of a 6-DOF physical registration parameter space.

XStep,YStep,ZStep specifies the translational step size in voxels, and may be non-integer.

XRStep,YRStep,ZRStep specifies the rotational step size in degrees, and may be non-integer.

XBound,YBound,ZBound,XRBound,YRBound,ZRBound specifies the maximum allowable number of steps in 6-DOF.

LOCATION

AVW_MatchVoxels.h

SEE ALSO

AVW_StepSearchExtreme() AVW_StepSearchExtreme2D() AVW_BoundedStepSearchExtreme() AVW_BoundedStepSearchExtreme2D()

NAME

AVW_TileParameters – Tile Structure

SYNOPSIS

```
typedef struct
    {
    AVW_Volume          *Vol;
    AVW_ObjectMap       *Omap;

    unsigned int    Type;
    unsigned int    Checkpoint;
    int                         MaskValue;

    int                         CurveOpRadius;
    int                         CloseSrfcFlag;
    int                         KohonenMajorAxis;
    unsigned int    KohonenShapeOrient;
    float       KohonenShapeOffset;
    unsigned int    KohonenFlag;
    unsigned int    PolygonBudget;
    unsigned int    KohonenRepetitions;
    unsigned int    KohonenNeighborhood;
    unsigned int    KohonenTopology;
    float           KohonenNeighborRadius;
    float           KohonenAlpha;

    unsigned int   AddNodeFreq;
    unsigned int   MaximumAge;
    float       Eb, En;
    float       GrowingAlpha;
    float       GrowingD;

    int                         Edge;
    int                         Iteration;
    double              Step;
    double              Kd;
    double              Kj;
    } AVW_TileParameters;
```

DESCRIPTION

The *AVW_TileParameters* structure provides a method of passing the large number of tiling parameters to and from the AVW tiling routines.

Type determines the type of tiling preformed. The default is *AVW_DEFORMATION*. Possible values are:

AVW_TILE_KOHONEN, *AVW_TILE_GROW*, *AVW_DEFORMATION* and *AVW_MARCHING_CUBES*,

MaskValue specifies the object to tile. The object is considered to be all voxels having the value *MaskValue* (if the data is an object map, mask is the object number).

CurveOpRadius specifies the spacing between the elements of the curvature operator. Increasing the spacing between the elements reduces the effects of local noise on the curvature calculation.

CloseSrfcFlag if set will cause the generation of a surface with closed ends.

KohonenMajorAxis specifies the major adaptation axis for the kohonen tiling algorithm.

KohonenShapeOrient specifies the initial orientation of the network for the kohonen tiling algorithm.

KohonenShapeOffset specifies the multiplier used to determine the initial distance from the network to the object for the kohonen tiling algorithm

KohonenFlag specifies modifications to the kohonen tiling algorithm. May be one of:
AVW_TRAIN_IN_MAJOR_AXIS, and / or *AVW_TRAIN_WITH_WEIGHT*,

PolygonBudget specifies the maximum number of polygons that may be in the surface produced by the tiling algorithm(s). If *AVW_DEFORMATION* is set to *AVW_DEFORMATION*, the algorithm will attempt to determine the optimal number of polygonal tiles based on the size of the deresolution element.

KohonenRepetitions, specifies the number of times the data is presented to the network during adaptation.

KohonenNeighborhood specifies how a neighborhood of cells will move during adaptation of the kohonen network. May be one of
AVW_BUBBLE_NEIGHBORHOOD, *AVW_GAUSS_NEIGHBORHOOD*, and / or *AVW_TRIANGLE_NEIGHBORHOOD*,

KohonenTopology describes the network topology for the kohonen tiling algorithm. Currently only *AVW_RECTANGULAR_TOPOLOGY* is supported

KohonenNeighborRadius specifies the initial radius of a kohonen neighborhood. If the value is less than 1, the initial radius will be set to be one third of the distance around the 2-D bounding oval orthogonal to the major axis.

KohonenAlpha specifies the initial learning rate for the kohonen tiling algorithm.

AddNodeFreq specifies the number of adaptation steps envoked by the growing net algorithm before a new node is added.

MaximumAge specifies the maximal age number allowed for the growing net edges (connections). Edges with age greater than *MaximumAge* are removing from the growing net.

Eb, En specifies the adaptation factors for the best matching unit (bmu) node and its direct neighbors, respectively.

GrowingAlpha specifies the factor for decreasing the error counters of the 1st and 2nd bmus after the insertion of a new node to the growing net, respectively.

GrowingD specifies the factor for decreasing the error counters of all nodes after the insertion of a new node to the growing net.

Edge specifies the length (in voxels) of the initial deresolution cube. if *PolygonBudget* is set to 0, this parameter will be used in determining the number of polyonal tiles used in the surface. The smaller the value, the greater then number of polygonal tiles. A value of 1 will tile at the voxel level. It is possible for the algorithm to start thrashing if the *PolygonBudget* and *Edge* are set to small values. In this case, the algorithm will be unable to build a topology that meets the error conditions.

Iteration specifies the number of deformation steps the algorithm will take during its adaptation phase. The more iterations the surface under goes, the better its fit. It is possible for the algorithm to start thrashing if the number of iterations is too high as it will be unable to meet the error conditions.

Step specifies the time step size for the adaptation phase.

Kd specifies the amount of attraction between the object's surface and the polygonal topology. The greater the attraction, the faster the polygonal topology will approach the surface. Conversely, the greater the attraction, the harder it is for the algorithm to avoid getting stuck in a local rather than global minima.

Kj specifies the stiffness of the polygonal topology. The stiffer the topology, the less effect small amounts of surface noise has on the final geometric surface. Likewise, the stiffer the surface, the more likely it will be that small surface details will be lost.

LOCATION *AVW_Model.h*

SEE ALSO *AVW_DestroyTileParameters(), AVW_InitializeTileParameters(), AVW_TileVolume(), AVW_ObjectMap, AVW_Volume*

NAME

AVW_TiledSurface – 3D polygonal surface description

SYNOPSIS

```
typedef struct
   {
   AVW_FPointList3 *Coords;
   unsigned int    NumberOfIndices;
   unsigned int    MaximumIndices;
   unsigned int    BlockSize;
   int             *Indices
   } AVW_TiledSurface
```

DESCRIPTION

The *AVW_TiledSurface* structure describes a 3D polygonal surface. It consists of a list of 3-space coordinates given in *Coords* and a list of the interconnections between those coordinates given in *Indices*. Thus a surface consisting of a single unit quadralateral polygon would have *Coords* containing: (0,0,0) (0,1,0) (1,1,0) (1,0,0) and *Indices* containing: 0 1 2 3 SRFC_END_INDEX *NumberOfIndices* is the current number of indices in the list. *MaximumIndices* indicates the number of indices this list can hold before automatic reallocation. *BlockSize* indicates the number of indices added to the current size when reallocation is necessary.

LOCATION

AVW_Model.h

SEE ALSO

AVW_ExtractTiledSurface(), AVW_LoadTiledSurface(), AVW_SaveTiledSurface(), AVW_DestroyTiledSurface(), AVW_FPointList3

NAME

AVW_Tree – List of tree points

SYNOPSIS

```
typedef struct
        {
        unsigned int NumberOfPoints;
        unsigned int MaximumPoints;
        unsigned int BlockSize;
        AVW_TreePoint *Points;
        } AVW_Tree;
```

DESCRIPTION

The *AVW_Tree* structure provides way to pass and return a list of skeletal trees.

LOCATION

AVW_Tree.h

SEE ALSO

AVW_TreePoint, AVW_CreateTree(), AVW_AddTreeChild(), AVW_DestroyTree(), AVW_FindTreeIndex() AVW_LoadTree(), AVW_SaveTree()

NAME

AVW_TreePoint – Point of a tree

SYNOPSIS

```
typedef struct
   {
   short X, Y, Z;
   unsigned short Level;
   int ParentIndex;
   unsigned int NumberOfChildren;
   unsigned int *ChildIndex;
   } AVW_TreePoint;
```

DESCRIPTION

The *AVW_TreePoint* structure provides way to pass points of skeletal trees.

LOCATION

AVW_TreePoint.h

SEE ALSO

AVW_Tree, *AVW_CreateTree()*, *AVW_AddTreeChild()*, *AVW_DestroyTree()*, *AVW_FindTreeIndex()* *AVW_LoadTree()*, *AVW_SaveTree()*

NAME

AVW_VisibleSurface – Visible Surface Structure

SYNOPSIS

```
typedef struct
   {
   int NumberOfSurfaces;
   int MaximumSurfaces;
   AVW_VisibleSurfacePoint *SurfacePoint;
   } AVW_VisibleSurface;
```

DESCRIPTION

The *AVW_VisibleSurface* structure is used to pass pre-determined surface information to and from *AVW* functions.

NumberOfSurfaces is the number of points in this surface.

MaximumSurfaces is the number of points that can be held in the *SurfacePoint* array, before it must be realloc'd.

SurfacePoint is an array of *AVW_VisibleSurfacePoint's.*

LOCATION

AVW_Render.h

SEE ALSO

AVW_VisibleSurfacePoint

NAME AVW_VisibleSurfacePoint – Surface Point Structure

SYNOPSIS

```
typedef struct
   {
   short int X, Y, Z;
   short int Padding;
   double Value;
   } AVW_VisibleSurfacePoint;
```

DESCRIPTION The *AVW_VisibleSurfacePoint* structure holds the location and pixel value for a predetermined surface voxel.

X, *Y*, and *Z* indicates a voxels 3-D location.

Padding is unused. It exists for byte alignment purposes only.

value contains the pixel value information.

LOCATION *AVW_Render.h*

SEE ALSO *AVW_VisibleSurface*

NAME

AVW_Volume – Volume Structure

SYNOPSIS

```
typedef struct
   {
   void *Mem;
   int DataType;
   unsigned int Width;
   unsigned int Height;
   unsigned int Depth;
   unsigned int BytesPerPixel;
   unsigned int BytesPerLine;
   unsigned int BytesPerImage;
   unsigned int PixelsPerImage;
   unsigned int BytesPerVolume;
   unsigned int VoxelsPerVolume;
   AVW_Colormap *Colormap;
   char *Info;
   unsigned int *ZTable;
   unsigned int *YTable;
   } AVW_Volume;
```

DESCRIPTION

The *AVW_Volume* structure provides an easy way to pass volumes to and from *AVW* functions.

Mem is a pointer to the raw data. It is often necessary to typecast this pointer when using it. For example:

```
ptr = (unsigned char *) volume->Mem
```

DataType specifies the voxel data type. All AVW data types are valid: *AVW_UNSIGNED_CHAR*, *AVW_SIGNED_CHAR*, *AVW_UNSIGNED_SHORT*, *AVW_SIGNED_SHORT*, *AVW_UNSIGNED_INT*, *AVW_SIGNED_INT*, *AVW_FLOAT*, *AVW_COMPLEX*, and *AVW_COLOR*.

Width, *Height*, and *Depth* specify the dimensions of the volume.

BytesPerPixel, *BytesPerLine*, *BytesPerImage*, *PixelsPerImage*, *BytesPerVolume*, and *VoxelsPerVolume* contain often used pre-calcuated values.

Colormap is a pointer to an *AVW_Colormap*.

Info is a pointer to an *AVW* information string.

ZTable and *YTable* is an array of pre-calculated offsets. The following example shows how to get the address of the tenth line of the fifth image in the volume.

```
ptr = (unsigned char *) volume->Mem + volume->ZTable[4] + volume->YTable[9]
```

LOCATION

AVW.h

SEE ALSO

AVW_Colormap, *AVW_Image*, *AVW_CreateVolume()*, *AVW_DestroyVolume()*, *AVW_DESTROYVOLUME()*

NAME

AVW_DESTROYCOEFFS – Destroy Coefficent Macro

SYNOPSIS

```
#define AVW_DESTROYCOEFFS(coeffs)      \
    {                                  \
    AVW_DestroyCoeffs(coeffs);         \
    coeffs = NULL;                     \
    }
```

DESCRIPTION

The *AVW_DESTROYCOEFFS* macro provides a convenient method of freeing an *AVW_FilterCoeffs* structure and setting the pointer to *NULL*.

LOCATION

AVW_Filter.h

SEE ALSO

AVW_DestroyCoeffs(), AVW_CreateButterworthCoeffs(), AVW_CreateCoeffs(), AVW_CreateCircularMTF(), AVW_CreateGaussianCoeffs(), AVW_CreateSphericalMTF(), AVW_CreateStoksethMTF(), AVW_FilterCoeffs

NAME

AVW_DESTROYCOLORMAP – Destroy Colormap Macro

SYNOPSIS

```
#define AVW_DESTROYCOLORMAP(colormap)     \
    {                                     \
    AVW_DestroyColormap(colormap);        \
    colormap = NULL;                      \
    }
```

DESCRIPTION

The *AVW_DESTROYCOLORMAP* macro provides a convenient method of freeing an *AVW_Colormap* and setting the pointer to that colormap to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_DestroyColormap(), AVW_CopyColormap(), AVW_CreateColormap(), AVW_DitherImage(), AVW_DitherVolume(), AVW_IsGrayColormap(), AVW_ReduceColors(), AVW_Colormap

NAME

AVW_DESTROYCOMPOSITEINFO – Destroy CompositeInfo Macro

SYNOPSIS

```
#define AVW_DESTROYCOMPOSITEINFO(cinfo)     \
    {                                       \
    AVW_DestroyCompositeInfo(cinfo);        \
    cinfo = NULL;                           \
    }
```

DESCRIPTION

The *AVW_DESTROYCOMPOSITEINFO* macro provides a convenient method of freeing an *AVW_CompositeInfo* and setting the pointer to that composite info to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_DestroyCompositeInfo(), AVW_CreateCompositeInfo(), AVW_CompositeInfo

NAME

AVW_DESTROYFPOINTLIST2 – Destroy AVW_FPointList2 Macro

SYNOPSIS

```
#define AVW_DESTROYFPOINTLIST2(plist)     \
    {                              \
    AVW_DestroyFPointList2(plist);       \
    plist = NULL;                  \
    }
```

DESCRIPTION

The *AVW_DESTROYFPOINTLIST2* macro provides a convenient method of freeing an *AVW_FPointList2* and setting the pointer to that point list to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_CreateFPointList2(), AVW_DestroyFPointList2(), AVW_FPointList2

NAME

AVW_DESTROYFPOINTLIST3 – Destroy AVW_FPointList3 Macro

SYNOPSIS

```
#define AVW_DESTROYFPOINTLIST3(plist)     \
   {                                \
   AVW_DestroyFPointList3(plist);       \
   plist = NULL;                    \
   }
```

DESCRIPTION

The *AVW_DESTROYFPOINTLIST3* macro provides a convenient method of freeing an *AVW_FPointList3* and setting the pointer to that point list to *NULL.*

LOCATION

AVW.h

SEE ALSO

AVW_CreateFPointList3(), AVW_DestroyFPointList3(), AVW_FPointList3

NAME

AVW_DESTROYHISTOGRAM – Destroy Histogram Macro

SYNOPSIS

```
#define AVW_DESTROYHISTOGRAM(histo)      \
    {                                    \
    AVW_DestroyHistogram(histo);         \
    histo = NULL;                        \
    }
```

DESCRIPTION

The *AVW_DESTROYHISTOGRAM* macro provides a convenient method of freeing an *AVW_Histogram* and setting the pointer to that histogram to *NULL.*

LOCATION

AVW_Hisogram.h

SEE ALSO

AVW_DestroyHistogram(), AVW_CreateHistogram(), AVW_Histogram

NAME

AVW_DESTROYIMAGE – Destroy Image Macro

SYNOPSIS

```
#define AVW_DESTROYIMAGE(image)     \
   {                                \
   AVW_DestroyImage(image);         \
   image = NULL;                    \
   }
```

DESCRIPTION

The *AVW_DESTROYIMAGE* macro provides a convenient method of freeing an *AVW_Image* and setting the pointer to that image to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_DestroyImage(), AVW_CreateImage(), AVW_Image

NAME

AVW_DESTROYIPOINTLIST2 – Destroy AVW_IPointList2 Macro

SYNOPSIS

```
#define AVW_DESTROYIPOINTLIST2(plist)     \
   {                          \
   AVW_DestroyIPointList2(plist);       \
   plist = NULL;                  \
   }
```

DESCRIPTION

The *AVW_DESTROYIPOINTLIST2* macro provides a convenient method of freeing an *AVW_IPointList2* and setting the pointer to that point list to *NULL.*

LOCATION

AVW.h

SEE ALSO

AVW_CreateIPointList2(), AVW_DestroyIPointList2(), AVW_IPointList2

NAME

AVW_DESTROYIPOINTLIST3 – Destroy AVW_IPointList3 Macro

SYNOPSIS

```
#define AVW_DESTROYIPOINTLIST3(plist)      \
    {                                      \
    AVW_DestroyIPointList3(plist);         \
    plist = NULL;                          \
    }
```

DESCRIPTION

The *AVW_DESTROYIPOINTLIST3* macro provides a convenient method of freeing an *AVW_IPointList3* and setting the pointer to that point list to *NULL.*

LOCATION

AVW.h

SEE ALSO

AVW_CreateIPointList3(), AVW_DestroyIPointList3(), AVW_IPointList3

NAME

AVW_DESTROYLIST – Destroy List Macro

SYNOPSIS

```
#define AVW_DESTROYLIST(list)     \
   {                              \
   AVW_DestroyList(list);         \
   list = NULL;                   \
   }
```

DESCRIPTION

The *AVW_DESTROYLIST* macro provides a convenient method of freeing an *AVW_List* and setting the pointer to that list to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_DestroyList(), AVW_List

NAME

AVW_DESTROYPOINTLIST2 – Destroy AVW_PointList2 Macro

SYNOPSIS

```
#define AVW_DESTROYPOINTLIST2(plist)     \
   {                                     \
   AVW_DestroyPointList2(plist);         \
   plist = NULL;                         \
   }
```

DESCRIPTION

The *AVW_DESTROYPOINTLIST2* macro provides a convenient method of freeing an *AVW_PointList2* and setting the pointer to that point list to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_CreatePointList2(), AVW_DestroyPointList2(), AVW_PointList2

NAME

AVW_DESTROYPOINTLIST3 – Destroy AVW_PointList3 Macro

SYNOPSIS

```
#define AVW_DESTROYPOINTLIST3(plist)     \
   {                                     \
   AVW_DestroyPointList3(plist);         \
   plist = NULL;                         \
   }
```

DESCRIPTION

The *AVW_DESTROYPOINTLIST3* macro provides a convenient method of freeing an *AVW_PointList3* and setting the pointer to that point list to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_CreatePointList3(), AVW_DestroyPointList3(), AVW_PointList3

NAME

AVW_DESTROYPOINTVALUELIST – Destroy AVW_PointValueList Macro

SYNOPSIS

```
#define AVW_DESTROYPOINTVALUELIST(plist)      \
   {                              \
   AVW_DestroyPointValueList(plist);        \
   plist = NULL;                  \
   }
```

DESCRIPTION

The *AVW_DESTROYPOINTVALUELIST* macro provides a convenient method of freeing an *AVW_PointValueList* and setting the pointer to that point list to *NULL*.

LOCATION

AVW.h

SEE ALSO

AVW_CreatePointValueList(), AVW_DestroyPointValueList(), AVW_PointValueList

NAME

AVW_DESTROYRENDEREDIMAGE – Destroy Rendered Image Macro

SYNOPSIS

```
#define AVW_DESTROYRENDEREDIMAGE(rendered)     \
   {                        \
   AVW_DestroyRenderedImage(rendered);       \
   rendered = NULL;                 \
   }
```

DESCRIPTION

The *AVW_DESTROYRENDEREDIMAGE* macro provides a convenient method of freeing an *AVW_RenderedImage* and setting the pointer to that rendered image to *NULL.*

LOCATION

AVW_Render.h

SEE ALSO

AVW_DestroyRenderedImage(), AVW_RenderVolume(), AVW_RenderedImage

NAME

AVW_DESTROYRENDERPARAMETERS – Destroy Render Parameters Macro

SYNOPSIS

```
#define AVW_DESTROYRENDERPARAMETERS(render_param)     \
    {                                \
    AVW_DestroyRenderParameters(render_param);        \
    render_param = NULL;                   \
    }
```

DESCRIPTION

The *AVW_DESTROYRENDERPARAMETERS* macro provides a convenient method of freeing an *AVW_RenderParameters* and setting the pointer to that render_param to *NULL*.

LOCATION

AVW_Render.h

SEE ALSO

AVW_DestroyRenderParameters(), AVW_InitializeRenderParameters(), AVW_RenderParameters

NAME

AVW_DESTROYVOLUME – Destroy Volume Macro

SYNOPSIS

```
#define AVW_DESTROYVOLUME(volume)    \
   {                         \
   AVW_DestroyVolume(volume);      \
   volume = NULL;                  \
   }
```

DESCRIPTION

The *AVW_DESTROYVOLUME* macro provides a convenient method of freeing an *AVW_Volume* and setting the pointer to that volume to *NULL.*

LOCATION

AVW.h

SEE ALSO

AVW_DestroyVolume(), AVW_CreateVolume(), AVW_Volume

NAME AVW_GETBLUE – Get Blue Macro

SYNOPSIS #define AVW_GETBLUE(v) (((int) (v) & 0xff)

DESCRIPTION The *AVW_GETBLUE* macro gets the blue portion of a packed RGB value.

LOCATION *AVW.h*

SEE ALSO *AVW_GETRED(), AVW_GETGREEN(), AVW_MAKERGB(), AVW_GetPixel(), AVW_GetVoxel()*

NAME AVW_GETGREEN – Get Green Macro

SYNOPSIS

```
#define AVW_GETGREEN(v)   ((((int) (v)) >> 8) & 0xff)
```

DESCRIPTION The *AVW_GETGREEN* macro gets the green portion of a packed RGB value.

LOCATION *AVW.h*

SEE ALSO *AVW_GETRED(), AVW_GETBLUE(), AVW_MAKERGB(), AVW_GetPixel(), AVW_GetVoxel()*

NAME AVW_GETRED – Get Red Macro

SYNOPSIS #define AVW_GETRED(v) ((((int) (v)) >> 16) & 0xff)

DESCRIPTION The *AVW_GETRED* macro gets the red portion of a packed RGB value.

LOCATION *AVW.h*

SEE ALSO *AVW_GETGREEN(), AVW_GETBLUE(), AVW_MAKERGB(), AVW_GetPixel(), AVW_GetVoxel()*

NAME

AVW_MAKERGB – Make RGB Macro

SYNOPSIS

```
#define AVW_MAKERGB(r, g, b)  ((double) ((((int) (r)) << 16) | (((int) (g)) << 8)
                             | ((int) (b))))
```

DESCRIPTION

The *AVW_MAKERGB* macro provides an easy method of packing *Red*, *Green*, and *Blue* values into a *double*. Some AVW function require the packed RGB values.

LOCATION

AVW.h

SEE ALSO

AVW_GETRED(), AVW_GETGREEN(), AVW_GETBLUE(), AVW_PutPixel(), AVW_PutVoxel(), AVW_SetImage(), AVW_SetVolume()

NAME

AVW_READSWAP – swaps bytes in data as it is read

SYNOPSIS

```
#include    <stdio.h>
#include    "AVW.h"

AVW_READSWAP(ptr, size, nitems, stream)
void *ptr;
int size, nitems;
FILE *stream;
```

DESCRIPTION

AVW_READSWAP() is a modification of the system call fread() which causes data to be byte swapped on Little Endian machines. Byte swapping data as it is read from disk allows machines of different types to share data files.

RETURN VALUES

AVW_READSWAP() returns the value returned from the internal call to fread(). Please refer to the fread() call for possible return values and error checking.

SEE ALSO

AVW_QuadSwapImage(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapLong(), AVW_SwapShort(), AVW_WRITESWAP(), fread(), fwrite(), swab()

NAME

AVW_WRITESWAP – swaps bytes on disk writes

SYNOPSIS

```
#include      <stdio.h>
#include      "AVW.h"

AVW_WRITESWAP(ptr, size, nitems, stream)
void *ptr;
int size, nitems;
FILE *stream;
```

DESCRIPTION

AVW_WRITESWAP() is a modification of the system call fwrite() which causes data to be byte swapped on Little Endian machines. Byte swapping data as it is written to disk allows machines of different types to share data files.

RETURN VALUES

AVW_WRITESWAP() returns the value returned from the internal call to fwrite(). Refer to the fwrite() call for possible return values and error checking.

SEE ALSO

AVW_DataTypeToBands(), AVW_DataTypeToBytes(), AVW_QuadSwapImage(), AVW_ReverseBits(), AVW_SwapBlock(), AVW_SwapDouble(), AVW_SwapFloat(), AVW_SwapImage(), AVW_SwapInt(), AVW_SwapLong(), AVW_SwapShort(), AVW_READSWAP(), fread(), fwrite(), swab()