BR

Sumário

1 Como compilar GDAL **1**

2 Modelo de Dados de GDAL **3**

2.1 Dataset 4

2.1.1 Sistema de Coordenadas 4

2.1.2 Transformacao afim 5

2.1.3 GCPs 5

2.1.4 Metadados 6

2.1.4.1 Dominio de SUBDATASETS 6

2.1.4.2 Dominio IMAGE_STRUCTURE 6

2.1.4.3 Dominio xml: 7

2.2 Banda de Imagem Raster 7

2.3 Tabela de cores 8

2.4 Overviews 8

3 Tutorial da API do GDAL **9**

3.1 Abrindo um Arquivo 10

3.2 Extraindo Informacoes do Arquivo 11

3.3 Extraindo uma Banda Raster 12

3.4 Lendo dato Raster 13

3.5 Fechando o Dataset 14

3.6 Tecnicas para criar arquivos 15

3.7 Usando CreateCopy() 16

3.8 Usando Create() 17

4 GDAL - Biblioteca de Abstracoes de Dados Geo-Espaciais **19**

4.1 Documentacao para usuario 20

4.2 Documentacao para desenvolvedor 20

4.3 Lista de discussao por correio eletronico 21

4.4 Como Reportar Problemas (Bugs) 21
4.5 GDAL em outras linguagens 21

Capítulo 1

Como compilar GDAL

Este tópico mudo-se para o endereço wiki: `http://trac.osgeo.org/gdal/wiki/BuildHints`

Capítulo 2

Modelo de Dados de GDAL

Este documento descreve o Modelo de Dados implementado pela bilbioteca GDAL, os tipos de informação que GDAL pode conter e a sua semântica.

2.1 Dataset

Um dataset (representado pela classe GDALDataset) é um agrupamento de bandas "raster" e um conjunto de informação comum a estas bandas. Em particular, um dataset possui o conceito de tamanho (em pixels e em linhas) que se aplica à todas as bandas. O Dataset também é reponsável pelas definições de referência geográficas e sistema de coordenadas comum a todas as bandas. O Dataset também pode possuir metadado, representado por uma lista de strings no formato de pares nome=valor.

Note que o GDALDataet e o modelo de bandas "raster" em GDAL são ligeiramente baseados na especificação do "Grid Coverage" do OpenGIS (OGC).

2.1.1 Sistema de Coordenadas

O sistema de coordenadas de um dataset é representado por uma string no formato "Well Kown Text" que contém:

- Nome geral para o sistema de coordenadas.
- Nome do sistema de coordenadas geográficas.
- Nome do datum.
- Nome da Elipsóide, semi-eixo maior e o inverso do achatamento.
- Nome do meridiano de origem e sua distância de Greenwich.
- Nome da projeção (ex. Transverse Mercator).
- Lista de parâmetros de projeção (ex. central_meridian).
- Nome das unidades e fator de conversão para metros ou radianos.
- Nome e order dos eixos.
- Códigos para relacionar a maioria dos items acima com tabelas de autoridades tais como EPSG.

Para maiores informações sobre o sistema de coordenadas OpenGIS WKT, e seu o mecanismo de manipulação, refira-se ao documento `osr_tutorial` e/ou a documentação da classe OGRSpatialReference.

O sistema de coordenadas retornado pelo método GDALDataset::GetProjectionRef() descreve as coordenadas geo-referenciadas que devem ser usadas na transformação afim retornada pelo método GDALDataset::GetGeoTransform(). O sistema de coordenadas retornado pelo métido GDALDataset::GetGCPProjection() descreve as coordenadas geo-referenciadas dos GCPs (Ground Control Points ou ponto de controle na superficíe) retornados pelo método GDALDataset::GetGCPs().

Note que a string do sistema de coordenada retornado for "" isto indica que nada se sabe sobre o sistema de coordenadas do referido dataset.

2.1.2 Transformacao afim

GDAL datasets tem duas formas de descrever o relationamento entre poisição em raster (coordenadas de pixel) e coordenadas geográficas. O primeiro e mais usado é a transformação afim (o outro usa os GCPs).

A transformação afim consiste em seis coeficiente retornados por GDALDataset::GetGeoTransform() que mapeiam coordenadas de pixel/linhas em coordenadas referenciadas spacialmente usango o seguinte relacionamento:

```
Xgeo = GT(0) + Xpixel*GT(1) + Yline*GT(2)
Ygeo = GT(3) + Xpixel*GT(4) + Yline*GT(5)
```

No caso de imagens orientadas em direção ao norte, os coeficientes GT(2) e GT(4) são igual a zero, o GT(1) é a largura en pixels, e GT(5) é a altura en pixels. A posição (GT(0), GT(3)) é o canto superior esquerdo do pixel no canto superior esquerdo da imagem.

Note que a coordenada pixel/linha neste formato se extendem de (0.0, 0.0) no pixel do canto superior esquerdo localizado no canto superior esquerdo da imagem até o canto inferior direito do pixel localizado no canto inferior direito da imagem. Consequentement, a coordenada do pixel no canto superior esquerdo é (0.5, 0.5).

2.1.3 GCPs

Um dataset pode possuir um conjunto de pontos de controle relacionando uma ou mais posições no raster com coordenadas geo-referenciadas. Todos os GCPs compartilham o mesmo sistema de coordanadas (retornado pelo GDALDataset::GetGCPProjection()). Cada GCP (representado pelo classe GDAL_GCP) contén as seguintes estrutura:

```
typedef struct
{
    char *pszId;
    char *pszInfo;
    double  dfGCPPixel;
    double dfGCPLine;
    double dfGCPX;
    double dfGCPY;
    double dfGCPZ;
} GDAL_GCP;
```

A string pszId deve conter um único (e geralmente, mas nem sempre, numérico) identificador para cada GCP dentro do conjunto de GCPs do dataset. A string pszInfo é usualmente vazia mas pode conter qualquer texto associado ao GCP que o usuário defina. Potencialmente pszInfo poderá conter informação, decodificável automaticamente, sobre o estado do GCP. O que no entando ainda não tem sido feito até o momento.

Os campos (Pixel, Line) contém a localização do GCP no raster. Os campos (X,Y,Z) estão associados com a localixação geo-referenciada sendo que Z é normalmente zero.

O modelo de dados de GDAL nao impõe nenhum mecanismo de transformação para que deve ser generado a partir dos GCPs... Isto é deixado para o applicação. No entanto polinomios de primeiro a quinto grau são comument utilizados.

Normalmente um dataset ira conter transformações afins (geotransform), GCPs or nenhum dos dois. É incomum encontrar os dois e neste caso não existe uma definição de qual deve ter precedência.

2.1.4 Metadados

O metado do GDAL é um formato auxiliar de dado textual específico para cada aplicação que contém uma lista de pares nome=valor. É requerido que os nomes sejam chaves facilmente distintas (sem espaço e caracteres especiais). O valor pode ser de qualquer tamanho e pode conter qualquer coisa, exceto um caracter núlo (zero ASCII).

O sistema de manipulação de metadas não é apropriado para suportar grandes volume de metados. Mais de 100K bytes de metada para um dataset provavelmente causará degradação na performance.

Com o tempo espera-se que nomes bem conhecidos sejam identificados e bem estabelicidos semanticamente, no entanto isto ainda não tem ocorrido até o momento.

Alguns formatos suportam metados genéricos (definidos pelo usuário), enquanto outros formatos supportam nomes specificos. Por exemplo o TIFF driver retorna algumas de suas "tags" na forma de metadas tais como data e hora:

```
TIFFTAG_DATETIME=1999:05:11 11:29:56
```

O matado é dividido em grupos chamados domínios (domains). O domínio default tem o nome em branco (NULL ou ""). Existem alguns domínios specificos para propósitos especiais. Note que correntement não existe uma forma de enumerar todos os domínios disponíveis para um dado objeto mas as aplicações podem testar a existência de domínios conhecidos.

2.1.4.1 Dominio de SUBDATASETS

Os domínios SUBDATASETS podem listar uma hierarquia de datasets filhos. Normalmente isto é usado para prover endereços para várias imagens (ou datasets) internas à um único arquivo de images (como HDF ou NITF). Por exemplo, um arquivo NITF com quatro imagens pode ter a seguinte lista de subdataset:

```
SUBDATASET_1_NAME=NITF_IM:0:multi_1b.ntf
SUBDATASET_1_DESC=Image 1 of multi_1b.ntf
SUBDATASET_2_NAME=NITF_IM:1:multi_1b.ntf
SUBDATASET_2_DESC=Image 2 of multi_1b.ntf
SUBDATASET_3_NAME=NITF_IM:2:multi_1b.ntf
SUBDATASET_3_DESC=Image 3 of multi_1b.ntf
SUBDATASET_4_NAME=NITF_IM:3:multi_1b.ntf
SUBDATASET_4_DESC=Image 4 of multi_1b.ntf
SUBDATASET_5_NAME=NITF_IM:4:multi_1b.ntf
SUBDATASET_5_DESC=Image 5 of multi_1b.ntf
```

O valor de _NAME é a string que pode ser usada em uma chamada à GDALOpen() para acessar o arquivo. O valor de _DESC tem a intensão de apresentar uma descrição ao usuário no caso da seleção de subdataset.

2.1.4.2 Dominio IMAGE_STRUCTURE

O domínio defult do metadata ("") está relacionado à image e não particularmente a forma como a image é organizada no arquivo. Isto significa que este domínio é apropriado para se copiar junto com o dataset quanto a image é copida para um novo format. Alguma informações pertence somente a um particular format de arquivo e sua forma de armazenamento. Para que esta informação não seja copiada inadvertidamente este tipo de informação é colocado num domínio especial chamado IMAGE_STRUCTURE que não normalmente não deve ser copiado de um formato para outro.

Um item que aparece no domínio IMAGE_STRUCTURE é o esquema de compressão usado pelo formato. O nome deste item no metada é COMPRESSION mas o valor dele pode ser específico para cada formato.

2.1.4.3 Dominio xml:

Qualquer domínio cujo o prefix do nome é "xml:" não é um item de metada normal mas sim um único documento XML armazenado numa string longa.

2.2 Banda de Imagem Raster

Uma banda de image raster é representada em GDAL pela classe GDALRaseterBand. Ela representa uma única banda, canal ou camada. Em GDAL, uma banda não representa necessáriamente uma imagem inteira. Uma imagem 24bit RGB, por exemplo, é normalmente representada por um dataset com três bandas, uma para vermelho, uma pra verde e outra para azul.

Um banda raster tem as seguintes propriedades:

- Altura e largura em número de pixels e linhas. Pode ser o mesmo valor definido para o dataset, caso seje uma banda de resolução total.
- Um tipo de dato. Podendo ser Byte, UInt16, Int16, UInt32, Int32, Float32, Float64, and the complex types CInt16, CInt32, CFloat32, and CFloat64.
- Um tamanho de bloco, que seria o valor preferido (eficiente) para se acessar um pedaço do arquivo. Para imagens dividida em "tiles" este seria o tamanho de um bloco. Para image organizadas por linha de varredura (scanLine), tamalho de bloco seria normalmente uma "scanline".
- Lista de pares nome/valor do metada, no mesmo formato que o dataset mas com informações que sejam potencialmente especificas da banda.
- Um string com uma descrição opcional
- Uma lista de nomes de categorias (efetivamente, nome de classe numa imagem temática)
- Opcionalmente, mínimos e máximos valores de pixel.
- Opcionalmente, o valor do deslocamento e da escala para transformar os valores de banda raster em valores significativoes (ex. para traduzir altura para metros).
- Opcionalmente, o nome da unidade do raster. Indicando, por exemplo, a unidade linear para um dado de elevação.
- A tipo de interpretação das cores da banda, que pode ser:
  - GCI_Undefined: O default, Nada ou desconhecido
  - GCI_GrayIndex: Image em toms de cinza independente
  - GCI_PaletteIndex: A banda representa índice para tabela de cores
  - GCI_RedBand: A banda representa a proção vermelha de uma image RGB ou RGBA
  - GCI_GreenBand: A banda representa a proção verde de uma image RGB ou RGBA
  - GCI_BlueBand: A banda representa a proção azul de uma image RGB ou RGBA
  - GCI_AlphaBand: A banda representa a proção alpha de uma image RGBA
  - GCI_HueBand: A banda representa a proção intensidade de uma image HLS
  - GCI_SaturationBand: A banda representa a proção saturação de uma image HLS
  - GCI_LightnessBand: A banda representa a proção luz de uma image HLS
  - GCI_CyanBand: A banda representa a proção ciano de uma image CMY ou CMYK

  - GCI_MagentaBand: A banda representa a proção magenta de uma image CMY ou CMYK
  - GCI_YellowBand: A banda representa a proção amarelo de uma image CMY ou CMYK
  - GCI_BlackBand: A banda representa a proção preto de uma image CMYK

- Uma tabela de cores. A ser detalhada porteriormente.
- Indicação da existência de versões reduzida de banda (piramides).

2.3 Tabela de cores

A tabela de cores consiste de zero ou mais items decritos em C na seguinte estrucura:

```
typedef struct
{
    /- gray, red, cyan or hue -/
    short      c1;

    /- green, magenta, or lightness -/
    short      c2;

    /- blue, yellow, or saturation -/
    short      c3;

    /- alpha or blackband -/
    short      c4;
} GDALColorEntry;
```

A tabela de cores também possiu um valor de interpretação (GDALPaletteInterp) com um dos seguintes valores, que indica como os valores de c1/c2/c3/c4 dos item da tabela devem ser interpretados.

- GPI_Gray: Usa c1 como valor de tom de cinza.
- GPI_RGB: Usa c1 como vermelho, c2 como verde, c3 como azul e c4 como alpha.
- GPI_CMYK: Usa c1 como ciano, c2 como magenta, c3 como amarelo e c4 como preto.
- GPI_HLS: Usa c1 como intensidade, c2 como saturação, c3 como luminosidade.

Para associar a cor com o pixel, o valor de pixel deve ser usado como um &iacute&ndice dentro da tabela de cores. Isto significa que as cores devem ser sempre aplicadas começando do zero em diante. Não existe provisão para macanismo de calcular escala antes de se buscar valores na tabela de cores.

2.4 Overviews

Uma banda pode ter zero ou mais overviews. Cada overview é representada por um banda GDALRaster-Band independente. O tamanho (em pixels e linhas) de um overview é diferente do raster que ela representa, mas a região geográfica covertida pela overview é a mesma da image de resolução total.

As overviews são usadas para mostrar uma image de resolução reduzida de forma mais rápida do que se acessar todos os dados da resolução total da imagem e depois reduzir-la por amostragem.

Bandas podem também a propriedadde HasArbitraryOverviews que indica se, caso seja verdadeira, o raster pode ser lido a qualquer resolução mas sem nenhuma distinção de overviews. Isto se aplica a algumas images codificadas tipo FFT, ou imagens capturadas via servidores (como OGDI) onde a amostragem pode ser executada mais eficentemente do que na máquina remota.

Capítulo 3

Tutorial da API do GDAL

3.1 Abrindo um Arquivo

Antes de abrir um dataset raster suportado por GDAL é necessário registar os drivers, existe um driver para cada formato suportado e o registro dos driver é realizado normalmente com a função GDALAllRegister(). GDALAllRegister() registar todos os drivers conhecidos including os "plug-in", que são bilioteca dinâmicas, carregadas pelo método GDALDriverManager::AutoLoadDrivers(). Se por algum motivo uma aplicações necessetita limitar o conjunto de drivers seria útil verificar o código de `gdalallregister.cpp`.

Uma vez que os drivers são registados, a aplicação deve chamar a função GDALOpen() para abrir dataset, passando o nome do dataset e a forma de acesso (GA_ReadOnly ou GA_Update).

Em C++:

```
#include "gdal_priv.h"

int main()
{
    GDALDataset  *poDataset;

    GDALAllRegister();

    poDataset = (GDALDataset *) GDALOpen( pszFilename, GA_ReadOnly );
    if( poDataset == NULL )
    {
        ...;
    }
```

Em C:

```
#include "gdal.h"

int main()
{
    GDALDatasetH  hDataset;

    GDALAllRegister();

    hDataset = GDALOpen( pszFilename, GA_ReadOnly );
    if( hDataset == NULL )
    {
        ...;
    }
```

Em Python:

```
    import gdal
    from gdalconst import *

    dataset = gdal.Open( filename, GA_ReadOnly )
    if dataset is None:
        ...
```

Note que se GDALOpen() retornar NULL significa que ocorreu uma falhada, e que as mensagens de erro deverão ter sido emitidas através de CPLError(). Se você quiser controlar como os erros estão relatados revise a a documentação do usuário de função CPLError(). Em geral, todo o GDAL usa CPLError() para o relatório de erro. Note também que o pszFilename não necessita realmente ser o nome de uma arquivo físico (no entando geralmente é). A interpretação é dependente do driver, e pôde ser um URL, um nome de arquivo com os parâmetros adicionais adicionados na string para controlar a abertura do arquivo ou qualquer outra coisa. Tente por favor não limitar diálogos da seleção da arquivo de GDAL somente a selecionar arquivos físicos.

3.2 Extraindo Informacoes do Arquivo

Como descrita em `GDAL Data Model`, um GDALDataset contem uma lista de bandas raster, todas pertencendo à uma mesma área, e tendo a mesma definição. Possui também um metadata, um sistema coordenado, uma transformação geográfica, tamanho do raster e várias outras informações.

```
adfGeoTransform[0] /* top left x */
adfGeoTransform[1] /* w-e pixel resolution */
adfGeoTransform[2] /* rotation, 0 if image is "north up" */
adfGeoTransform[3] /* top left y */
adfGeoTransform[4] /* rotation, 0 if image is "north up" */
adfGeoTransform[5] /* n-s pixel resolution */
```

Se nós quiséssemos imprimir alguma informação geral sobre a série de dados nós pudemos fazer o seguinte:

Em C++:

```
double        adfGeoTransform[6];

printf( "Driver: %s/%s\n",
        poDataset->GetDriver()->GetDescription(),
        poDataset->GetDriver()->GetMetadataItem( GDAL_DMD_LONGNAME ) );

printf( "Size is %dx%dx%d\n",
        poDataset->GetRasterXSize(), poDataset->GetRasterYSize(),
        poDataset->GetRasterCount() );

if( poDataset->GetProjectionRef()  != NULL )
    printf( "Projection is `%s'\n", poDataset->GetProjectionRef() );

if( poDataset->GetGeoTransform( adfGeoTransform ) == CE_None )
{
    printf( "Origin = (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );

    printf( "Pixel Size = (%.6f,%.6f)\n",
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

Em C:

```
GDALDriverH   hDriver;
double        adfGeoTransform[6];

hDriver = GDALGetDatasetDriver( hDataset );
printf( "Driver: %s/%s\n",
        GDALGetDriverShortName( hDriver ),
        GDALGetDriverLongName( hDriver ) );

printf( "Size is %dx%dx%d\n",
        GDALGetRasterXSize( hDataset ),
        GDALGetRasterYSize( hDataset ),
        GDALGetRasterCount( hDataset ) );

if( GDALGetProjectionRef( hDataset ) != NULL )
    printf( "Projection is `%s'\n", GDALGetProjectionRef( hDataset ) );

if( GDALGetGeoTransform( hDataset, adfGeoTransform ) == CE_None )
{
    printf( "Origin = (%.6f,%.6f)\n",
            adfGeoTransform[0], adfGeoTransform[3] );

    printf( "Pixel Size = (%.6f,%.6f)\n",
```

```
            adfGeoTransform[1], adfGeoTransform[5] );
}
```

Em Python:

```
print 'Driver: ', dataset.GetDriver().ShortName,'/', \
      dataset.GetDriver().LongName
print 'Size is ',dataset.RasterXSize,'x',dataset.RasterYSize, \
      'x',dataset.RasterCount
print 'Projection is ',dataset.GetProjection()

geotransform = dataset.GetGeoTransform()
if not geotransform is None:
    print 'Origin = (',geotransform[0], ',',geotransform[3],')'
    print 'Pixel Size = (',geotransform[1], ',',geotransform[5],')'
```

3.3 Extraindo uma Banda Raster

Neste ponto o acesso aos dados da raster através de GDAL pode ser feito uma banda de cada vez. A Band também possui metadata, tamanho de block, tabelas da cor, e vário a outra informação disponível na classe GDALRasterBand. Os seguintes códigos buscam um objeto de GDALRasterBand da série de dados (numerada a partir de 1 em GetRasterCount()) e a exposições de algums informações sobre ela.

Em C++:

```
GDALRasterBand  *poBand;
int             nBlockXSize, nBlockYSize;
int             bGotMin, bGotMax;
double          adfMinMax[2];

poBand = poDataset->GetRasterBand( 1 );
poBand->GetBlockSize( &nBlockXSize, &nBlockYSize );
printf( "Block=%dx%d Type=%s, ColorInterp=%s\n",
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(poBand->GetRasterDataType()),
        GDALGetColorInterpretationName(
            poBand->GetColorInterpretation()) );

adfMinMax[0] = poBand->GetMinimum( &bGotMin );
adfMinMax[1] = poBand->GetMaximum( &bGotMax );
if( ! (bGotMin && bGotMax) )
    GDALComputeRasterMinMax((GDALRasterBandH)poBand, TRUE, adfMinMax);

printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );

if( poBand->GetOverviewCount() > 0 )
    printf( "Band has %d overviews.\n", poBand->GetOverviewCount() );

if( poBand->GetColorTable() != NULL )
    printf( "Band has a color table with %d entries.\n",
             poBand->GetColorTable()->GetColorEntryCount() );
```

Em C:

```
GDALRasterBandH hBand;
int             nBlockXSize, nBlockYSize;
int             bGotMin, bGotMax;
double          adfMinMax[2];

hBand = GDALGetRasterBand( hDataset, 1 );
GDALGetBlockSize( hBand, &nBlockXSize, &nBlockYSize );
```

```
printf( "Block=%dx%d Type=%s, ColorInterp=%s\n",
        nBlockXSize, nBlockYSize,
        GDALGetDataTypeName(GDALGetRasterDataType(hBand)),
        GDALGetColorInterpretationName(
            GDALGetRasterColorInterpretation(hBand)) );

adfMinMax[0] = GDALGetRasterMinimum( hBand, &bGotMin );
adfMinMax[1] = GDALGetRasterMaximum( hBand, &bGotMax );
if( ! (bGotMin && bGotMax) )
    GDALComputeRasterMinMax( hBand, TRUE, adfMinMax );

printf( "Min=%.3fd, Max=%.3f\n", adfMinMax[0], adfMinMax[1] );

if( GDALGetOverviewCount(hBand) > 0 )
    printf( "Band has %d overviews.\n", GDALGetOverviewCount(hBand));

if( GDALGetRasterColorTable( hBand ) != NULL )
    printf( "Band has a color table with %d entries.\n",
             GDALGetColorEntryCount(
                 GDALGetRasterColorTable( hBand ) ) );
```

In Python (note several bindings are missing):

```
band = dataset.GetRasterBand(1)

print 'Band Type=',gdal.GetDataTypeName(band.DataType)

min = band.GetMinimum()
max = band.GetMaximum()
if min is not None and max is not None:
    (min,max) = ComputeRasterMinMax(1)
print 'Min=%.3f, Max=%.3f' % (min,max)

if band.GetOverviewCount() > 0:
    print 'Band has ', band.GetOverviewCount(), ' overviews.'

if not band.GetRasterColorTable() is None:
    print 'Band has a color table with ', \
    band.GetRasterColorTable().GetCount(), ' entries.'
```

3.4 Lendo dato Raster

Há algumas maneiras diferentes de ler dados da raster, mas o mais comum é através do Método GDALRasterBand::RasterIO(). Este método tomará automaticamente cuidado da conversão do tipo de dados, amostragem e janela de dados requerida. O seguinte código lerá o primeiro scanline dos dados em um buffer em tamanho similar à quantidade lida, convertendo os valores para ponto flutuando como parte da operação:

Em C++:

```
float *pafScanline;
int   nXSize = poBand->GetXSize();

pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
poBand->RasterIO( GF_Read, 0, 0, nXSize, 1,
                  pafScanline, nXSize, 1, GDT_Float32,
                  0, 0 );
```

Em C:

```
float *pafScanline;
```

```
int   nXSize = GDALGetRasterBandXSize( hBand );

pafScanline = (float *) CPLMalloc(sizeof(float)*nXSize);
    GDALRasterIO( hBand, GF_Read, 0, 0, nXSize, 1,
              pafScanline, nXSize, 1, GDT_Float32,
              0, 0 );
```

Em Python:

```
scanline = band.ReadRaster( 0, 0, band.XSize, 1, \
                            band.XSize, 1, GDT_Float32 )
```

Note que o scanline retornado é do tipo char∗, e contem os bytes xsize∗4 de dados binários brutos de ponto flutuando. Isto pode ser convertido em Python usando o módulo do **struct** da biblioteca padrão:

```
import struct

tuple_of_floats = struct.unpack('f' * b2.XSize, scanline)
```

A chamada de RasterIO espera os seguintes argumentos.

```
CPLErr GDALRasterBand::RasterIO( GDALRWFlag eRWFlag,
                                 int nXOff, int nYOff, int nXSize, int nYSize,
                                 void * pData, int nBufXSize, int nBufYSize,
                                 GDALDataType eBufType,
                                 int nPixelSpace,
                                 int nLineSpace )
```

Note que a mesma chamada de RasterIO() poderá ler, ou gravar dependendo do valor de eRWFlag (GF_-Read ou GF_Write). Os argumentos nXOff, nYOff, nXSize, nYSize descreve a janela de dados da raster para ler (ou para gravar). Não necessita ser coincidente com os limites da image embora o acesso pode ser mais eficiente se for.

O pData é o buffer de memória para os dados que serão lidos ou gravados. O verdadeiro tipo de dado &eacute aquele passado por eBufType, tal como GDT_Float32, ou GDT_Byte. A chamada de RasterIO() cuidará de converter entre o tipo de dados do buffer e o tipo de dados da banda. Anotar que ao converter dados do ponto flutuando para o inteiro RasterIO arredonda para baixo, e ao converter de para fora dos limites de valores válidos para o tipo de saída, será escolhido o mais próximo valor possível. Isto implica, por exemplo, que os dados 16bit lidos em um buffer de GDT_Byte converterão todos os valores maiores de 255 para 255, os **dados não estão escalados!**

Os valores nBufXSize e nBufYSize descrevem o tamanho do buffer. Ao carregar dados na resolução completa os valores seria o mesmo que o tamanho da janela. Entretanto, para carregar uma vista de solução reduzida (overview) os valores podiam ser ajustado para menos do que a janela no arquivo. Neste caso o RasterIO() utilizará overview para fazer mais eficientemente o IO se as overview forem apropriadas.

O nPixelSpace, e o nLineSpace são normalmente zero indicando que os valores default devem ser usados. Entretanto, podem ser usados controlar o acesso à dados da memória, permitindo a leitura em um buffer que contem dados intercalados (interleave) pixel por exemplo.

3.5 Fechando o Dataset

Por favor tenha em mente que os objetos de GDALRasterBand estão possuídos por sua dataset, e nunca devem ser destruídos com o operador delete de C++. GDALDataset podem ser fechado chamando GDAL-Close() ou usando o operador delete no GDALDataset. Um ou outro resultado na finalização apropriada, e resolver gravações pendentes.

3.6 Tecnicas para criar arquivos

As arquivos em formatos suportados GDAL podem ser criadas se o driver do formato suportar a criação. Há duas técnicas gerais para criar arquivos, usando CreateCopy() e Create(). O método de CreateCopy chama o método CreateCopy() do driver do formato, e passar como parâmetro dataset que será copiado. O método criar chama o método Create() do driver, e então explicitamente grava todos os metadata, e dados da raster com as chamadas separadas. Todos os drivers que suportam criar arquivos novos suportam o método de CreateCopy(), mas somente algum sustentação o método Create().

Para determinar se o driver de um formato suporta Create ou CreateCopy é necessário verificar o DCAP_-CREATE e os metadata de DCAP_CREATECOPY no driver do formato objetam. Assegurar-se de que GDALAllRegister() tenha sido chamado antes de chamar GetDriverByName().

Em C++:

```
#include "cpl_string.h"
...
    const char *pszFormat = "GTiff";
    GDALDriver *poDriver;
    char **papszMetadata;

    poDriver = GetGDALDriverManager()->GetDriverByName(pszFormat);

    if( poDriver == NULL )
        exit( 1 );

    papszMetadata = poDriver->GetMetadata();
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
        printf( "Driver %s supports Create() method.\n", pszFormat );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
        printf( "Driver %s supports CreateCopy() method.\n", pszFormat );
```

Em C:

```
#include "cpl_string.h"
...
    const char *pszFormat = "GTiff";
    GDALDriver hDriver = GDALGetDriverByName( pszFormat );
    char **papszMetadata;

    if( hDriver == NULL )
        exit( 1 );

    papszMetadata = GDALGetMetadata( hDriver, NULL );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATE, FALSE ) )
        printf( "Driver %s supports Create() method.\n", pszFormat );
    if( CSLFetchBoolean( papszMetadata, GDAL_DCAP_CREATECOPY, FALSE ) )
        printf( "Driver %s supports CreateCopy() method.\n", pszFormat );
```

Em Python:

```
    format = "GTiff"
    driver = gdal.GetDriverByName( format )
    metadata = driver.GetMetadata()
    if metadata.has_key(gdal.DCAP_CREATE) \
       and metadata[gdal.DCAP_CREATE] == 'YES':
        print 'Driver %s supports Create() method.' % format
    if metadata.has_key(gdal.DCAP_CREATECOPY) \
       and metadata[gdal.DCAP_CREATECOPY] == 'YES':
        print 'Driver %s supports CreateCopy() method.' % format
```

Note que um número de drivers são de leitura apenas e não suportarão Create() ou CreateCopy ().

3.7 Usando CreateCopy()

O GDALDriver:: O método de CreateCopy() pode ser usado razoavelmente simples enquanto a maioria de informação é coletada do dataset de entrada. Entretanto, inclui opções para passar a formato opções específicas da criação, e para relatar o progresso ao usuário enquanto uma cópia longa da série de dados ocorre. Uma cópia simples de uma arquivo nomeou o pszSrcFilename, a uma arquivo nova nomeada pszDstFilename usando opções de defeito em um formato cujo o driver fosse buscado previamente pudesse olhar como este:

Em C++:

```
GDALDataset *poSrcDS =
   (GDALDataset *) GDALOpen( pszSrcFilename, GA_ReadOnly );
GDALDataset *poDstDS;

poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                NULL, NULL, NULL );
if( poDstDS != NULL )
    delete poDstDS;
```

Em C:

```
GDALDatasetH hSrcDS = GDALOpen( pszSrcFilename, GA_ReadOnly );
GDALDatasetH hDstDS;

hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                         NULL, NULL, NULL );
if( hDstDS != NULL )
    GDALClose( hDstDS );
```

Em Python:

```
src_ds = gdal.Open( src_filename )
dst_ds = driver.CreateCopy( dst_filename, src_ds, 0 )
```

Note que o método de CreateCopy() retorna um dataset writeable, e que deve ser fechado corretamente à escrita completa e a nivelar a série de dados ao disco. No Python encaixotar isto ocorre automaticamente quando os "dst_ds" saem do espaço. O valor FALSO (ou 0) usado para a opção do bStrict imediatamente depois que o nome de arquivo do destino na chamada de CreateCopy() indica que a chamada de CreateCopy() deve proseguir sem um erro fatal mesmo se a série de dados do destino não puder ser criada para combinar exatamente a série de dados da entrada. Isto pôde ser porque o formato da saída não suporta o datatype do pixel do dataset de entrada, ou porque o destino não pode suportar a escrita que georeferencing por exemplo.

Casos mais complexo pôdem envolver passar opções da criação, e usar um monitor predefinido do progresso como este:

Em C++:

```
#include "cpl_string.h"
...
    char **papszOptions = NULL;

    papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
    papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
    poDstDS = poDriver->CreateCopy( pszDstFilename, poSrcDS, FALSE,
                                    papszOptions, GDALTermProgress, NULL );
    if( poDstDS != NULL )
        delete poDstDS;
    CSLDestroy( papszOptions );
```

Em C:

```
#include "cpl_string.h"
...
    char **papszOptions = NULL;

    papszOptions = CSLSetNameValue( papszOptions, "TILED", "YES" );
    papszOptions = CSLSetNameValue( papszOptions, "COMPRESS", "PACKBITS" );
    hDstDS = GDALCreateCopy( hDriver, pszDstFilename, hSrcDS, FALSE,
                             papszOptions, GDALTermProgres, NULL );
    if( hDstDS != NULL )
        GDALClose( hDstDS );
    CSLDestroy( papszOptions );
```

Em Python:

```
    src_ds = gdal.Open( src_filename )
    dst_ds = driver.CreateCopy( dst_filename, src_ds, 0,
                                [ 'TILED=YES', 'COMPRESS=PACKBITS' ] )
```

3.8 Usando Create()

Em situações em que não se quer somente exportar um arquivo existente para uma arquivo novo, geralmente usa-se o método GDALDriver::Criar() (embora algumas opções interessantes são possíveis com o uso de arquivos virtuais ou de arquivos da em-memória). O método Create() examina uma lista de opções bem como o CreateCopy(), mas o tamanho da imagem, número das bandas e o tipo da banda deve ser fornecido explicitamente.

Em C++:

```
    GDALDataset *poDstDS;
    char **papszOptions = NULL;

    poDstDS = poDriver->Create( pszDstFilename, 512, 512, 1, GDT_Byte,
                                papszOptions );
```

Em C:

```
    GDALDatasetH hDstDS;
    char **papszOptions = NULL;

    hDstDS = GDALCreate( hDriver, pszDstFilename, 512, 512, 1, GDT_Byte,
                         papszOptions );
```

Em Python:

```
    dst_ds = driver.Create( dst_filename, 512, 512, 1, gdal.GDT_Byte )
```

Uma vez que o dataset é criado com sucesso, todos os metadata apropriados devem ser gravados no arquivo. O que variará de acordo com o uso, mas um caso simples com projeção, do geotransform e da raster é mostrado a seguir:

Em C++:

```
    double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
    OGRSpatialReference oSRS;
```

```
char *pszSRS_WKT = NULL;
GDALRasterBand *poBand;
GByte abyRaster[512*512];

poDstDS->SetGeoTransform( adfGeoTransform );

oSRS.SetUTM( 11, TRUE );
oSRS.SetWellKnownGeogCS( "NAD27" );
oSRS.exportToWkt( &pszSRS_WKT );
poDstDS->SetProjection( pszSRS_WKT );
CPLFree( pszSRS_WKT );

poBand = poDstDS->GetRasterBand(1);
poBand->RasterIO( GF_Write, 0, 0, 512, 512,
                  abyRaster, 512, 512, GDT_Byte, 0, 0 );

delete poDstDS;
```

Em C:

```
double adfGeoTransform[6] = { 444720, 30, 0, 3751320, 0, -30 };
OGRSpatialReferenceH hSRS;
char *pszSRS_WKT = NULL;
GDALRasterBandH hBand;
GByte abyRaster[512*512];

GDALSetGeoTransform( hDstDS, adfGeoTransform );

hSRS = OSRNewSpatialReference( NULL );
OSRSetUTM( hSRS, 11, TRUE );
OSRSetWellKnownGeogCS( hSRS, "NAD27" );
OSRExportToWkt( hSRS, &pszSRS_WKT );
OSRDestroySpatialReference( hSRS );

GDALSetProjection( hDstDS, pszSRS_WKT );
CPLFree( pszSRS_WKT );

hBand = GDALGetRasterBand( hDstDS, 1 );
GDALRasterIO( hBand, GF_Write, 0, 0, 512, 512,
              abyRaster, 512, 512, GDT_Byte, 0, 0 );

GDALClose( hDstDS );
```

Em Python:

```
import Numeric, osr

dst_ds.SetGeoTransform( [ 444720, 30, 0, 3751320, 0, -30 ] )

srs = osr.SpatialReference()
srs.SetUTM( 11, 1 )
srs.SetWellKnownGeogCS( 'NAD27' )
dst_ds.SetProjection( srs.ExportToWkt() )

raster = Numeric.zeros( (512, 512) )
dst_ds.GetRasterBand(1).WriteArray( raster )
```

Capítulo 4

GDAL - Biblioteca de Abstracoes de Dados Geo-Espaciais

Selecione o idioma: [English] [Russian] [Portuguese]

GDAL é uma bibliotéca para tradução de formatos de dados geográficos distribuída pela Open Source Geospatial Foundation sob a licença X/MIT estilo Open Source. As aplicações que à utilizam acessam todos os formats supportados pela biblioteca através de **um único modelo de dados abstrato** (pag. 3). A bilioteca GDAL também conta com uma variadade de programas **utilitários de linha de comando** (pag. **??**) para a tradução de formatos bem como uma série de outras funções. A página novidades descreve o lançamento em October de 2008 da release 1.5.3 da biblioteca GDAL/OGR.

A bilioteca OGR (cujo código fonte esta incluído na GDAL) possue funcionalidades semelhantes às da GDAL para datos vetorias (Simple Features).

Página Principal: http://www.gdal.org

Download: ftp at remotesensing.org, http at download.osgeo.org

4.1 Documentacao para usuario

- Wiki - Documentação e dicas mantidas por vários desenvolvedores e contribuidores
- Downloads - Programas executáveis prontos para usar
- Formatos Suportados
- Programs Utilitários
- Perguntas Mais Frequentes
- Modelo de dados **(em português)**
- Governança e participação comunitária
- Patrocinadores, Agradecimentos e Créditos
- Software que usam GDAL

4.2 Documentacao para desenvolvedor

- Como compilar GDAL
- Downloads - código fonte
- Documentação de referencia da API
- Tutorial da API do GDAL **(em português)**
- Tutorial de implementação de drivers
- Tutorial de transformação geométrica
- Tutorial de Referência Espacial
- GDAL API para C
- GDALDataset
- GDALRasterBand
- GDAL para Windows CE

4.3 Lista de discussao por correio eletronico

A lista gdal-announce é uma lista de pequeno volume de mensagens usada somente para anunciar lançamento de versões e desenvolvimentos significativos.

A lista gdal-dev@lists.maptools.org pode ser usada para a discussão sobre o desenvolvimento e uso da GDAL e tecnologias relaciadas. A inscrição e o acesso aos arquivo da lista podem ser feitos nesta página. Os arquivos da lista também estão disponíveis (somente para leitura) através do news://news.gmane.org/gmane.comp.gis.gdal.devel ou na página http://news.gmane.org/gmane.comp.gis.gdal.devel.

Usuários e desenvolvedores da GDAL/OGR podem constantemente ser achados no canal IRC gdal do irc.freenode.net.

4.4 Como Reportar Problemas (Bugs)

Bugs em GDAL podem ser reportados e também listados através do software Trac.

4.5 GDAL em outras linguagens

Lista de linguagens de programção supportadas por GDAL:

- Perl
- Python
- VB6 Bindings (sem o uso de SWIG)
- GDAL interface para R by Timothy H. Keitt.
- Ruby
- Java
- C# / .Net