ICS 07.040
CCS A 77

团 体 标 准

T/CAGIS 9—2023

遥感时空谱多维数据格式

Multi-dimensional data format for spatial-temporal-spectral remote sensing image

2023-03-13 发布　　2023-03-13 实施

中国地理信息产业协会　发布
中国标准出版社　出版

目　　次

前　言

本文件按照 GB/T 1.1—2020《标准化工作导则　第 1 部分：标准化文件的结构和起草规则》的规定起草。

本文件由中国地理信息产业协会提出并归口。

本文件起草单位：中国科学院空天信息创新研究院、天津中科谱光信息技术有限公司、中国科学院地理科学与资源研究所、国家对地观测科学数据中心、中国资源卫星应用中心、航天宏图信息技术股份有限公司、二十一世纪空间技术应用股份有限公司、生态环境部卫星环境应用中心、北京与光科技有限公司、珠海欧比特宇航科技股份有限公司、鄂尔多斯市林业和草原局综合保障中心。

本文件主要起草人：张立福、岑奕、孙雪剑、张霞、黄长平、黄瑶、张东辉、刘闯、李国庆、龙小祥、王宇翔、苏东卫、孙中平、王宇、颜军、宝孟克那顺。

引　言

随着航空、航天等遥感技术的发展和地球空间数据获取能力的不断提高，遥感数据的积累越来越多，海量的遥感数据，特别是时间序列遥感数据为研究特定区域的地表时空动态变化提供了丰富的数据保障。由于传统的遥感影像数据是三维数据，增加时间维度后，遥感数据变为时空谱多维数据，现有的遥感数据存储方式无法直接存储多维数据格式。因此，遥感多维数据的存储、共享、互操作以及多维数据可视化成为地球科学领域迫切需要解决的问题之一。

本文件定义了一种可应用于遥感时空谱多维数据存储管理的数据格式，适用于时间序列遥感数据存储、分析和管理。该数据格式可以推进遥感地学数据的存储、共享、互操作以及多维数据的可视化，是遥感数据深度开发和应用的基础。

本文件的发布机构提请注意，声明符合本文件时，可能涉及第 6、7 章中相应内容的相关专利的使用。

本文件的发布机构对于该专利的真实性、有效性和范围无任何立场。

该专利持有人已向本文件的发布机构承诺，他愿意同任何申请人在合理且无歧视的条款或条件下，就专利授权许可进行谈判。该专利持有人的声明已在本文件的发布机构备案。相关信息可以通过以下联系方式获得：

专利持有人姓名：张立福，孙雪剑，张霞

地址：北京市朝阳区大屯路甲 20 号北

请注意除上述专利外，本文件的某些内容仍可能涉及专利。本文件的发布机构不承担识别专利的责任。

遥感时空谱多维数据格式

1 范围

本文件规定了遥感时空谱多维数据的基本要求、组织结构、存储格式和数据集构建的流程。

本文件适用于时间序列遥感数据及其遥感数据产品的存储、传输、交换和共享。

2 规范性引用文件

本文件没有规范性引用文件。

3 术语和定义

下列术语和定义适用于本文件。

3.1

遥感 remote sensing

在不直接接触物体的情况下，收集、解释目标信息的技术。

[来源：ISO 19101-2：2018，3.31]

3.2

数据格式 data format

数据在传输、处理和存储过程中的编排形式。

[来源：GB/T 14950—2009，5.224]

3.3

多维数据格式 multi-dimensional data format

包含时间、空间、光谱等多个维度信息的遥感数据在传输、处理和存储过程中的编排形式。

注：多维数据格式的光谱维度也可存储通过波段运算得到的遥感信息产品。

3.4

头信息 header information

以头文件形式对快视数据进行描述的元数据。

[来源：GB/T 36300—2018，3.1.2]

3.5

数据结构 data structure

为存储、访问、传送和获得数据所使用的计算机可读的格式。

[来源：GB/T 17694—1999，3.138]

3.6

数据类型 data type

允许对域内的值进行操作的值域说明。

[来源：ISO 19103，4.14]

3.7

时谱 spectrotemporal

遥感探测器获得的地物辐射光谱信号值或经过特定处理后得到的光谱参量值按照获取时间顺序排

列组成的图形。

4 符号和缩略语

4.1 符号

下列符号适用于本文件。

$A(t,s,c,r)$：t 时间的光谱立方体 A 中第 s 波段、第 c 列、第 r 行的像元。

$B_{\times\times\times}$：表示一个波段的数据。

$F_{\times\times\times}$：表示多个波段组成的立方体数据。

$P_{\times\times\times}$：表示时空谱多维数据集，其中×××为五种数据结构中的一种。

c：某一列，取值范围$[1,C]$。

r：某一行，取值范围$[1,R]$。

s：某一个光谱波段，取值范围$[1,S]$。

t：某一个时间，取值范围$[1,T]$。

4.2 缩略语

下列缩略语适用于本文件。

API：应用程序接口（Application Program Interface）

LandSat：陆地卫星（Land Satellite）

MODIS：中分辨率成像分光辐射计（MODerate-resolution Imaging Spectroradiometer ）

MDD：多维数据（Multi-Dimensional Data）

TIB：按照时间顺序逐波段进行存储的数据格式（Temporal Interleaved by Band）

TIP：按照时间顺序逐像素进行存储的数据格式（Temporal Interleaved by Pixel）

TIS：按照时间顺序逐光谱进行存储的数据格式（Temporal Interleaved by Spectrum）

TSB：按照时间序列的顺序存储影像立方体数据，每个时间序列的影像立方体数据中，按照波段的顺序存储（Temporal Sequential in Band）

TSP：按照时间序列的顺序存储影像立方体数据，每个时间序列的影像立方体数据中，按照像素的顺序存储（Temporal Sequential in Pixel）

5 基本规定

5.1 数据格式

5.1.1 TIB 数据格式

多维数据格式数据存储模式之一，按照时间顺序逐波段进行存储的数据格式。先存完第一个波段的所有时间序列的数据，再存储第二个波段的所有时间序列的数据，以此类推。

5.1.2 TIP 数据格式

多维数据格式数据存储模式之一，按照时间顺序逐像素逐波段进行存储的数据格式。先存储第一个波段的第一个像素的所有时间序列的数据，再存储第一个波段的第二个像素所有时间序列的数据，以此类推，直至所有波段的所有像素数据存储完成。

5.1.3 TIS 数据格式

多维数据格式数据存储模式之一，按照时间顺序逐光谱进行存储的数据格式。存储完第一个像素的所有时间维度的光谱数据，再存储第二个像素的所有时间维度的光谱数据，以此类推，直至所有像素的光谱数据存储完。

5.1.4 TSB 数据格式

多维数据格式数据存储模式之一，按照时间序列的顺序存储影像立方体数据，每个时间序列的影像立方体数据中，按照波段的顺序存储。

注：影像立方体，是指多/高光谱遥感影像二维 X、Y 空间和一维 Z 光谱三个维度数据组成的影像立方集合。

5.1.5 TSP 数据格式

多维数据格式数据存储模式之一，按照时间序列的顺序存储影像立方体数据，每个时间序列的影像立方体数据中，按照像素的顺序存储。

5.2 基本数据类型

本文件涉及的基本数据类型及要求见表 1。

表 1 基本数据类型及要求

类型	字节数	取值范围	描述
byte	1	[0,255]	单字节
int16	2	[−32 768,32 767]	短整型
uint16	2	[0,65 535]	无符号短整型
int32	4	[−2 147 483 648,2 147 483 647]	整型
uint32	4	[0,4 294 967 295]	无符号整型
int64	8	$[-2^{63},(2^{63}-1)]$	长整型
uint64	8	$[0,(2^{64}-1)]$	无符号长整型
float	4	$[-3.4\times10^{38},3.4\times10^{38}]$	单精度浮点型
double	8	$[-1.7\times10^{308},1.7\times10^{308}]$	双精度浮点型
complex64	8	用 float 型表示实部、虚部	复数型
complex128	16	用 double 型表示实部、虚部	双精度复数型

5.3 字符串类型

本文件涉及的字符串数据类型用 String 对象描述，采用 Unicode 编码，字符集规定为 UTF-8。

5.4 文本格式存储

本文件涉及的文本格式存储，采用 UTF-8 编码。

6 组织结构

6.1 文件组织

本文件规定的多维数据格式由下列两部分组成：

a) 头文件：记录影像数据的属性信息，包括各个维度的大小、数据存储格式、数据类型信息，也记录关于影像数据附加的描述信息，包括坐标投影和仿射变换系数、光谱维和时间维的名称以及文件名称和类型、数据偏移等描述信息。头文件为文本格式，后缀名为.mdr。

b) 数据体文件：存储影像数据的文件，可采用 TSB、TSP、TIB、TIP 和 TIS 五种类型数据结构（详见 7.2）的其中一种进行数据存储。数据体文件为十进制数值数据格式，后缀名为.mdd。

6.2 文件说明

MDD 文件组织形式见表 2。

表 2 MDD 文件组织形式

文件类型	存储形式	存储规定	是否必须
头文件	.mdr 文件	整个数据的描述信息，文本格式。 文件名可自定义，扩展名限定为“.mdr”	是
数据体文件	.mdd 文件	影像数据信息。 文件名可自定义，扩展名限定为“.mdd”	是

7 存储格式

7.1 MDD 头文件

MDD 头文件记录用于表达和解析影像数据体文件的所有元数据信息。头文件中各字段含义见表 3。

表 3 MDD 头文件中各字段含义

字段	类型	含义
Samples	uint32	数据每个波段所包含的列数
Lines	uint32	数据每个波段所包含的行数
Bands	uint32	数据所包含的波段数
Time	uint32	数据所包含的时间数，世界同一时间（UTC）
Header Offset	uint32	文件中存在的嵌入式头信息的字节数。读取文件时，会跳过这些字节
File Type	string	文件类型，如 MDD 标准
Data Type	uint32	数据类型代码：1＝byte；2＝int16；3＝int32；4＝float；5＝double；6＝complex64；9＝complex128；12＝uint16；13＝uint32；14＝int64；15＝uint64
Interleave	string	数据存储所使用的结构，包括 TSB、TSP、TIB、TIP 和 TIS
Sensor Type	string	传感器类型，如 LandSat TM、SPOT、MODIS 等
Byte Order	uint32	字节顺序。指占内存多于一个字节类型的数据在内存中的存放顺序，Byte order ＝ 0 表示先存储低字节（Least significant bit）；byte order＝1 表示先存储高字节（Most significant bit）
Map Info	string	地理坐标信息，按此顺序列出：投影名称、参考像素 x 位置、参考像素 y 位置、像素东经、像素北纬、像素大小 x、像素大小 y、投影区域、北或南半球（仅通用横墨卡托格网坐标系统（UTM））、地理坐标系

表 3　MDD 头文件中各字段含义（续）

字段	类型	含义
Coordinate System String	string	空间参考，包括投影名称、地理坐标系名称、空间参考大地基准名称、参考椭球及参数、子午线名称、地理坐标系单位、地图投影类型、地图投影参数等
Band Names	string	数据体文件中各个波段的名称
Time Names	string	数据中各个文件获取时间

7.2　数据体文件

7.2.1　数据体文件描述

MDD 数据体文件是存储影像数据的文件，采用二进制的字节流并以 TSB、TSP、TIB、TIP 和 TIS 五种数据结构中的一种对栅格图像数据进行存储。五种数据结构之间可以相互转换。

7.2.2　TSB 数据结构

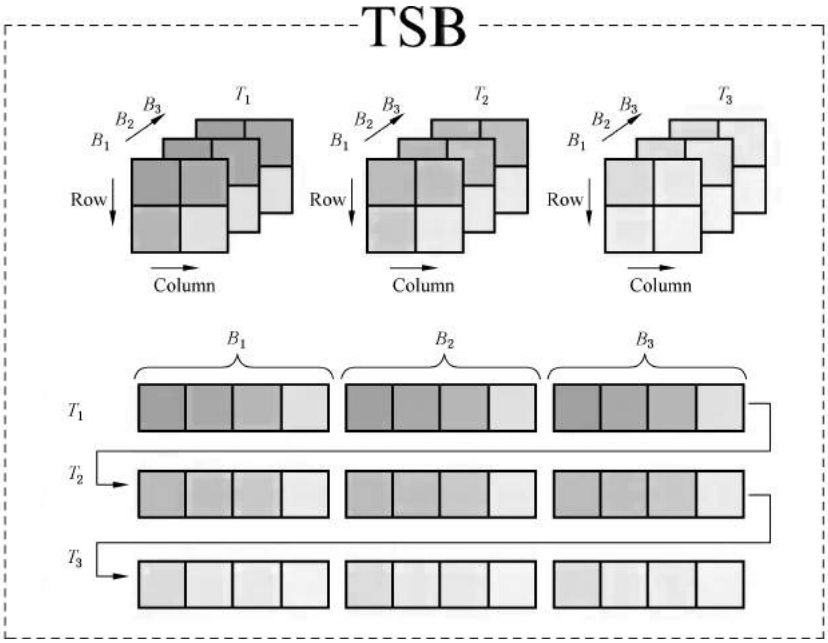
7.2.2.1　TSB 功能描述

TSB 数据结构适用于对一个时间上一个或若干个波段的数据进行空间维的处理，可应用于提取一个或多个时间所有波段组成的光谱立方体数据，方便进行光谱邻域运算和空间域滤波等。

7.2.2.2　TSB 结构描述

TSB 数据结构先将同一个时间的所有波段数据组织在一起，再按时间顺序排序存储。

假设多时相遥感数据集包含 T_1、T_2 和 T_3 三个时间的影像，每个时间的影像包含 3 个波段，每个波段 4 个像元（用 4 个小方块来表示），如图 1 所示。



注：小方块排列的顺序代表像元数据的存储顺序。

图 1　TSB 数据结构

7.2.2.3 TSB 表达式

TSB 数据结构用数学公式可表示为：

$$P_{\mathrm{TSB}}=F_{\mathrm{TSB}}(t)=F_{\mathrm{TSB}}(1)+F_{\mathrm{TSB}}(2)+\cdots+F_{\mathrm{TSB}}(T),\quad t\in[1,T]\cdots\cdots\cdots\cdots(1)$$

式中：

$F_{\mathrm{TSB}}(t)$——每个时间的立方体影像数据集。

式(1)表示数据存储按照时间 T 的顺序，依次存储不同时间的影像立方体，在每个立方体影像数据中，按照波段的顺序存储，可表示为：

$$F_{\mathrm{TSB}}(t)=B_{\mathrm{TSB}}(t,s)=B_{\mathrm{TSB}}(t,1)+B_{\mathrm{TSB}}(t,2)+\cdots+B_{\mathrm{TSB}}(t,\mathrm{S}),\quad s\in[1,\mathrm{S}]\ \cdots\cdots(2)$$

在每个波段的存储中，按照逐行的顺序存储，可表示为：

$$B_{\mathrm{TSB}}(t,s)=A(t,s,c,r)$$
$$=A(t,s,[1:C],1)+A(t,s,[1:C],2)+\cdots+A(t,s,[1:C],R),\quad r\in[1,R]\ \cdots(3)$$

式(1)、(2)、(3)中变量的含义见 4.1。

7.2.3 TSP 数据结构

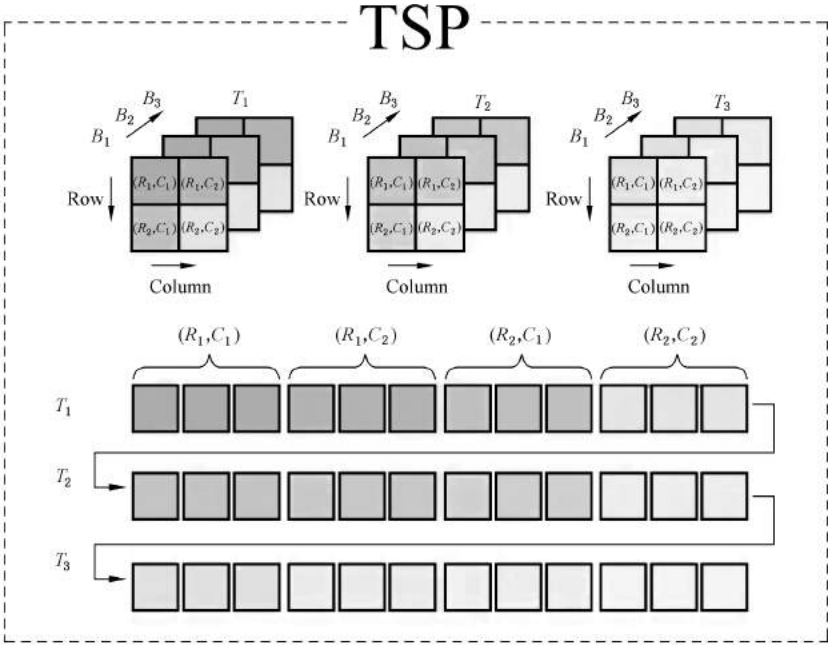
7.2.3.1 TSP 功能描述

TSP 数据结构适用于对一个时间的光谱数据进行操作，可应用于提取一个像元或者一片区域的光谱曲线或者对不同时间的影像进行光谱特征化等。

7.2.3.2 TSP 结构描述

TSP 结构格式采取光谱维优先的原则，先将同一时间影像的光谱数据组织在一起，再按时间顺序排序存储。

假设多时相遥感数据集包含 T_1、T_2和 T_3三个时间的影像，每个时间的影像包含 3 个波段，每个波段 4 个像元(用 4 个小方块来表示)，首先是将 T_1时间的立方体数据按照波段方向依次存储第一个像元的光谱信息，然后再按照“先行后列”的顺序存储各个像元点的光谱信息，最后依据上述规则将所有时间的立方体数据按时间顺序存储。



注：小方块排列的顺序代表像元数据的存储顺序。

图 2　TSP 数据结构

7.2.3.3　TSP 表达式

TSP 数据结构可表示为：

$$P_{TSP}=F_{TSP}(t)=F_{TSP}(1)+F_{TSP}(2)+\cdots+F_{TSP}(T),\quad t\in[1,T] \cdots\cdots\cdots\cdots(4)$$

式中，每个立方体 $F_{TSP}(t)$ 的存储可用式(5)表示：

$$\begin{aligned}F_{TSP}(t)=A(t,[1:S],c,r)=&A(t,[1:S],1,1)+A(t,[1:S],2,1)+\cdots+A(t,[1:S],C,1)\\&+A(t,[1:S],1,2)+A(t,[1:S],2,2)+\cdots+A(t,[1:S],C,2)\\&\vdots\\&+A(t,[1:S],1,R)+A(t,[1:S],2,R)+\cdots+A(t,[1:S],C,R)\end{aligned} \cdots\cdots\cdots\cdots(5)$$

式(4)、(5)中变量的含义见 4.1。

7.2.4　TIB 数据结构

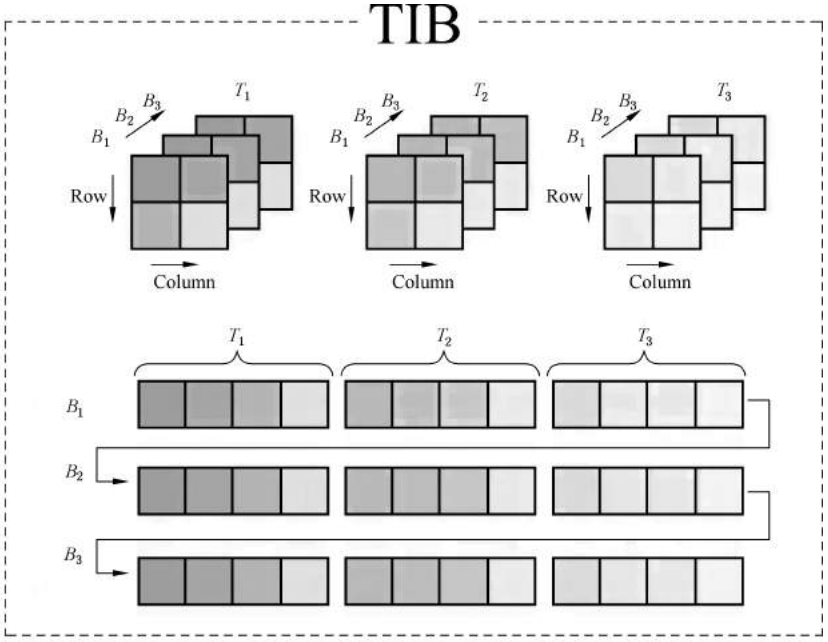
7.2.4.1　TIB 功能描述

TIB 数据结构适用于提取某个波段的时间序列立方体数据，可应用于提取一个波段的时间序列立方体，可以进行光谱时间序列分析或针对某个波段选择三个时间的数据假彩色合成显示。

7.2.4.2　TIB 结构描述

TIB 数据结构以波段优先为原则，先将一个波段所有时间数据组织在一起，再按波段顺序排序存储。

假设多时相遥感数据集包含 T_1、T_2 和 T_3 三个时间的影像，每个时间的影像包含 3 个波段，每个波段 4 个像元(用 4 个小方块来表示)，首先将第一个波段 T_1 时间的所有像元按照“先行后列”的顺序存储，然后依次将不同时间上第一个波段的所有像元数据按“先行后列”的顺序存储在一起，最后依据上述规则将波段的数据按照时间顺序进行存储。



注：小方块排列的顺序代表像元数据的存储顺序。

图 3　TIB 数据结构

7.2.4.3　TIB 表达式

TIB 数据结构可表示为：

$$P_{\mathrm{TIB}}=F_{\mathrm{TIB}}(s)=F_{\mathrm{TIB}}(1)+F_{\mathrm{TIB}}(2)+\cdots+F_{\mathrm{TIB}}(S),\quad s\in[1,S] \qquad (6)$$

式中：

$F_{\mathrm{TIB}}(s)$——不同时间序列的同一个波段组成的数据立方体，存储顺序为按照时间序列存储第一波段，然后再存储第二波段，直至存完所有波段。

$F_{\mathrm{TIB}}(s)$可表示为：

$$F_{\mathrm{TIB}}(s)=B_{\mathrm{TIB}}(t,s)=B_{\mathrm{TIB}}(1,s)+B_{\mathrm{TIB}}(2,s)+\cdots+B_{\mathrm{TIB}}(T,S) \qquad (7)$$

式(7)中，每个波段的存储顺序为：

$$\begin{aligned}B_{\mathrm{TIB}}(t,s)&=A(t,s,c,r)\\&=A(t,s,[1:C],1)+A(t,s,[1:C],2)+\cdots+A(t,s,[1:C],R)\end{aligned} \qquad (8)$$

式(7)、(8)中变量的含义见 4.1。

7.2.5　TIP 数据结构

7.2.5.1　TIP 功能描述

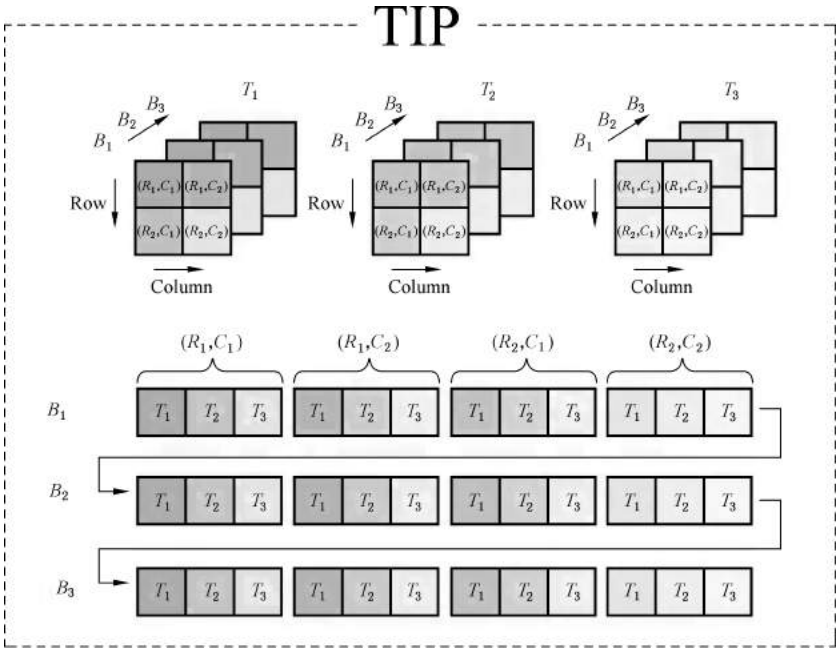
TIP 数据结构适用于对像元的时间谱进行处理与分析，可应用于提取一个像元或者一片区域在某个波段的时谱曲线，在时间维进行平滑和滤波处理，对时谱曲线进行拟合，以及进行预测分析等。

7.2.5.2　TIP 结构描述

TIP 数据结构以时序信息优先为原则，先将图像中像元的时谱数据组织在一起，再按像元“先行后列”顺序排序，最后按波段顺序存储。

假设多时相遥感数据集包含 T_1、T_2和 T_3三个时间的影像，每个时间的影像包含 3 个波段，每个波段 4 个像元(用 4 个小方块来表示)，首先将所有时间第一个波段的第一个像元点的数据按时间顺序组

织在一起，然后将所有时间第一个波段内的像元数据再按“先行后列”的原则顺序存储，最后依据上述规则将所有波段数据按照波段顺序进行存储。



注：小方块排列的顺序代表像元数据的存储顺序。

图 4　TIP 数据结构

7.2.5.3　TIP 表达式

TIP 数据结构可以表示为：

$$P_{TIP}=F_{TIP}(s)=F_{TIP}(1)+F_{TIP}(2)+\cdots+F_{TIP}(S),\quad s\in[1,S] \qquad \cdots\cdots\cdots\cdots(9)$$

式中：

$F_{TIP}(s)$——不同时间影像的同一个像元光谱组成的立方体。可表示为：

$$\begin{aligned}F_{TIP}(s)=&A([1:T],s,c,r)=A([1:T],s,1,1)+A([1:T],s,2,1)+\cdots+A([1:T],s,C,1)\\&+A([1:T],s,1,2)+A([1:T],s,2,2)+\cdots+A([1:T],s,C,2)\\&\vdots\\&+A([1:T],s,1,R)+A([1:T],s,2,R)+\cdots+A([1:T],s,C,R)\end{aligned} \qquad \cdots\cdots\cdots\cdots(10)$$

式(9)、(10)中变量的含义见 4.1。

7.2.6　TIS 数据结构

7.2.6.1　TIS 功能描述

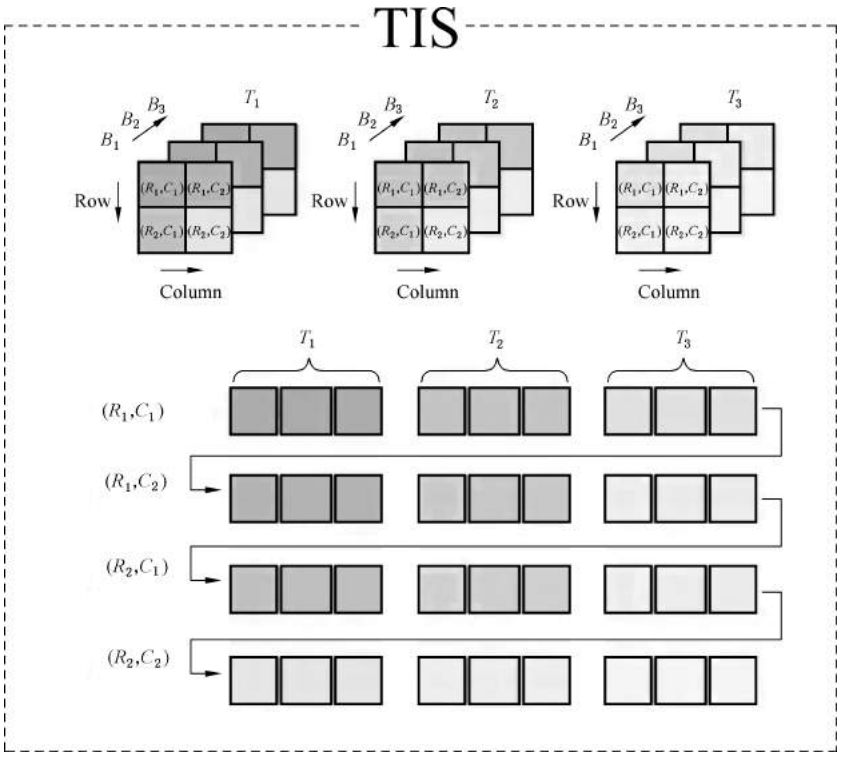
TIS 数据结构适用于提取像元光谱曲线的时间序列数据，可应用于抽取某一个像元在一个时间范围内的所有光谱曲线并对这些曲线进行三维可视化，并分析该像元的光谱随着时间变化的特征。

7.2.6.2　TIS 结构描述

TIS 数据结构以波段优先的原则，将每一个像元在整个时间序列上的光谱数据组织在一起，再依据空间“先行后列”顺序排序存储。

假设多时相遥感数据集包含 T_1、T_2和 T_3三个时间的影像，每个时间的影像包含 3 个波段，每个波段 4 个像元(用 4 个小方块来表示)，首先将所有波段第一个像元按照时间顺序进行存储，然后依据这一

规则将其他像元按照空间上“先行后列”的原则依次进行存储。



注：小方块排列的顺序代表像元数据的存储顺序。

图 5　TIS 数据结构

7.2.6.3　TIS 表达式

TIS 数据结构可以表示为：

$$\begin{aligned} P_{\mathrm{TIS}} = B_{\mathrm{TIS}}(c,r) = {} & B_{\mathrm{TIS}}(1,1) + B_{\mathrm{TIS}}(2,1) + \cdots + B_{\mathrm{TIS}}(C,1) \\ & + B_{\mathrm{TIS}}(1,2) + B_{\mathrm{TIS}}(2,2) + \cdots + B_{\mathrm{TIS}}(C,2) \\ & \vdots \\ & + B_{\mathrm{TIS}}(1,R) + B_{\mathrm{TIS}}(2,R) + \cdots + B_{\mathrm{TIS}}(C,R) \end{aligned} \quad \cdots\cdots(11)$$

式中：

$$\begin{aligned} B_{\mathrm{TIS}}(c,r) &= A(t,[1:S],c,r) \\ &= A(1,[1:S],c,r) + A(2,[1:S],c,r) + \cdots + A(T,[1:S],c,r), \quad t \in [1:T] \end{aligned} \quad \cdots\cdots(12)$$

式(11)、(12)中变量的含义见 4.1。

7.2.7　数据结构转换

五种数据结构类型按照以下步骤进行转换(代码见附录 A)：

a)　读取待转换数据结构的头文件信息，获取待转换数据结构类型；

b)　读取待转换数据结构数据体文件；

c)　按照目标数据结构存储形式写入数据体文件；

d)　将目标数据结构信息写入目标数据结构头文件。

8 多维数据集构建

8.1 一般要求

具有时间序列的数据集,可以是一个时相也可以是多个时相。

8.2 MDD 构建流程

MDD 构建流程如下(代码见附录 A):

a) 待构建的所有时相的数据存放于同一个文件夹下;

b) 输入路径选择上述总文件夹所在路径;

c) 根据需要,进行相关参数的设定;

d) 选择 MDD 的输出路径以及命名。

MDD 数据集的构建流程如图 6 所示。

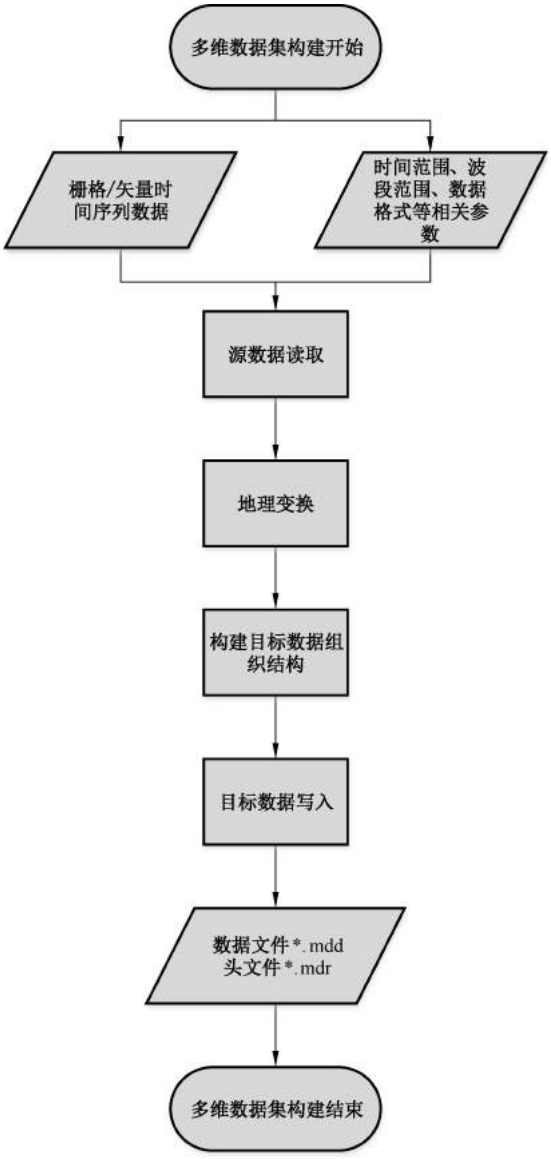


图 6 MDD 数据集构建流程

附 录 A
（资料性）
代 码 示 例

A.1 代码说明

本文件附录代码使用了 gdal220_x64 库，函数 API 使用说明见表 A.1、表 A.2 所示。

表 A.1 本文件使用函数 API

函数 API	功能
VSIFOpenL	创建文件
VSIMalloc	开辟内存
GDALOpen	打开影像
GDALGetRasterBand	获取指定波段
GDALRasterIO	读写图像矩阵数据
GDALDatasetRasterIO	读写图像矩阵数据
VSIFWriteL	写文件
CPLError	错误打印
GDALClose	关闭影像数据
VSIFree	释放内存

表 A.2 MDD 数据处理函数 API

函数 API	功能
OpenMddFile	读取 MDD 文件
CreateMddFromFolder	从 LandSat 文件创建 MDD 数据集
CreateMddFromFile	从 MODIS 文件创建 MDD 数据集
CreateMddFromEnvi	从 ENVI 的.img 文件创建 MDD 数据集
AppendMdd	从已有的 MDD 文件追加生成 MDD
MergeMdd	已有的两个 MDD 文件进行合并
OpenMddFileWithTime	指定时间读取 MDD 文件

A.2 MDD 格式数据的读取

A.2.1 OpenMddFile 函数定义

```
struct MddImage
{
```

```
    TimePhaseList                timePhaseList;
    GDALDataset                  * gdalDataset=NULL;
    GDALDataset                  * gdalOverview=NULL;
    std::string                  filePath;
    std::string                  fileName;
    std::string                  timePhaseName;
    std::string                  fileType;
    EDataFormat                  dataStorageType=E_Unknown;
    DataOpenForm                 datasetCreateMode = DOF_General;
    double                       adfGeoTransform[6];
    unsigned int                 headerOffset = 0;
    int                          xStart = 0;
    int                          yStart = 0;
};
CPL_DLL MddImage * OpenMddFile(MddImage * * mdd,const char * strFileName,bool operate=
true);
```

A.2.2 OpenMddFile 使用示例

示例数据说明:MODIS 时间序列数据
文件名:testMergeMODIS.mdd
路径:Data/testFile

```
MddImage * mdd=new MddImage();
const char * strFileName= "Data/testFile/testMergeMODIS.mdd";
OpenMddFile(&mdd, strFileName);
GDALDataset * dataser=mdd->subImageVector[0]->gdalDataset;
int len=GDALGetRasterCount(dataser);
```

A.3 从文件夹数据生成 MDD

A.3.1 CreateMddFromFolder 函数定义

```
CPLErr CPL_DLL CreateMddFromFolder(const char * srcFolder,
    const char * dstFilename,
    char * * srcSubDatasetsName,
    int count = 0,
    const char * srcExt = "*.tif",
    EDataFormat srcFormat = E_TSB,
    GDALProgressFunc pfnProgress = NULL,
    void * pProgressArg = NULL,
    char * * pDatasets = NULL);
```

A.3.2 CreateMddFromFolder 使用示例

示例数据说明:LandSat 时间序列数据
路径:Data/LandSatData/LandSat_2014

```
char * srcFolder ="Data/LandSatData/LandSat_2014/7581.7751";
char * dstFilename = "Data/testFile/testLandSat1.mdd";
char * srcSubDatasetsName[] = { "*_b1.tif","*_b2.tif", "*_b3.tif", "*_b4.tif", "*_b5.tif", "
*_b6.tif", "*_b7.tif",NULL };
int count = 3;
char * srcExt ="*.tif";
EDataFormat srcFormat = E_TSB;
CreateMddFromFolder(srcFolder, dstFilename, srcSubDatasetsName, count, srcExt, srcFormat,
pfnProgress, NULL);
```

A.4 从文件数据生成 MDD

A.4.1 CreateMddFromFile 函数定义

```
CPLErr CPL_DLL CreateMddFromFile(const char * srcFolder,
    const char * dstFilename,
    const char * srcExt = "*.hdf",
    int *srcSubIndex = NULL,
    int srcSubCount = 0,
    EDataFormat srcFormat = E_TSB,
    GDALProgressFunc pfnProgress = NULL,
    void * pProgressArg = NULL,
    const char * dstWkt = NULL,
    char * * pdatasets = NULL);
```

A.4.2 CreateMddFromFile 使用示例

示例数据说明:MODIS 时间序列数据
文件名:2011001
路径:Data/MODISData/MODIS

```
    char *srcFolder1 = "Data/MODISData/MODIS/2011001";
    char * dstFilename1 = "Data/testFile/testMODIS111111.mdd";
    char * srcExt1 = "*.hdf";
    int srcSubIndex[] ={0,1,2,3,4,5,6};
    int srcSubCount = ARRAYSIZE(srcSubIndex);
    EDataFormat srcFormat1 = E_TSB;
    char * dstWkt = "+proj=UTM +zone=5 +datum=WGS84 +units=m +no_defs";
    CreateMddFromFile ( srcFolder1, dstFilename1, srcExt1, srcSubIndex , srcSubCount,
srcFormat1, pfnProgress, NULL, dstWkt);
```

A.5 从 ENVI img 数据生成 MDD

A.5.1 CreateMddFromEnvi 函数定义

```
CPLErr CPL_DLL CreateMddFromEnvi(char * * pSrcPathList,
```

```
    char * dstFilename,
    std::vector<int>  v_subBandIndex,
    EDataFormat srcFormat,
    GDALProgressFunc pfnProgress = NULL,
                              void * pProgressArg = NULL);
```

A.5.2 CreateMddFromEnvi 使用示例

示例数据说明:ENVI 时序数据

路径:Data/enviData/NEW

```
char * pSrcPathList[] = { "Data/enviData/NEW/20010322_ref",
        "Data/enviData/NEW/20010407_ref",
        "Data/enviData/NEW/20010610_ref",
        "Data/enviData/NEW/20010626_ref",
        "Data/enviData/NEW/20010829_ref",NULL  };
char * dstFilename = "Data/testFile/testENVI.mdd";
std::vector<int>  subBandIndex;
subBandIndex.push_back(0);
subBandIndex.push_back(1);
subBandIndex.push_back(2);
EDataFormat srcFormat = E_TSB;
CreateMddFromEnvi( pSrcPathList,dstFilename,subBandIndex, srcFormat, pfnProgress , NULL);
```

A.6 MDD 数据追加

A.6.1 AppendMdd 函数定义

```
CPLErr CPL_DLL AppendMdd(const char *  pszMddFile,
    const char *  pszSrcFile,
    const char *  pszDstFile,
    EDataType appendType,
    EDataFormat pszFormat = E_TSB,
    GDALProgressFunc pfnProgress = NULL,
    void *  pProgressArg = NULL,
    const char *  dstWkt = NULL,
    char * *  pdatasets = NULL);
```

A.6.2 AppendMdd 使用示例

示例数据说明:LandSat 多维数据,LandSat 时间序列数据

```
const char *  pszMddFile = "Data/testFile/testLandSat.mdd";
const char *  pszSrcFile = "Data/LandSatData/LandSat_2014/7581.7751";
const char *  pszDstFile = "Data/testFile/testAppendLandSat.mdd";
EDataType appendType = LandSat;
EDataFormat pszFormat = E_TSB;
const char *  dstWkt = NULL;
```

```
char * pdatasets[] = {"1.LC80310362014018LGN00_b1.tif",
    "2.LC80310362014018LGN00_b2.tif",
    "3.LC80310362014018LGN00_b3.tif",
    "4.LC80310362014018LGN00_b4.tif",
    "5.LC80310362014018LGN00_b5.tif",
    "6.LC80310362014018LGN00_b6.tif",
    "7.LC80310362014018LGN00_b7.tif",NULL };
AppendMdd ( pszMddFile, pszSrcFile, pszDstFile, appendType, pszFormat, pfnProgress ,
NULL, dstWkt, pdatasets);
```

A.6.3 MODIS 数据 AppendMdd 使用示例

```
const char * pszMddFile = "Data/testFile/testMODIS.mdd";
const char * pszSrcFile = "Data/MODISData/MODIS/2011001";
const char * pszDstFile = "Data/testFile/testAppendMODIS.mdd";
EDataType appendType = MODIS;
EDataFormat pszFormat = E_TSB;
const char * dstWkt = NULL;
char * pdatasets[] = { "1.sur_refl_b01 MOD_Grid_500m_Surface_Reflectance (16-bit integer)",
    "2.sur_refl_b02 MOD_Grid_500m_Surface_Reflectance (16-bit integer)",
    "3.sur_refl_b03 MOD_Grid_500m_Surface_Reflectance (16-bit integer)",
    "4.sur_refl_b04 MOD_Grid_500m_Surface_Reflectance (16-bit integer)",
    "5.sur_refl_b05 MOD_Grid_500m_Surface_Reflectance (16-bit integer)",
    "6.sur_refl_b06 MOD_Grid_500m_Surface_Reflectance (16-bit integer)",
    "7.sur_refl_b07 MOD_Grid_500m_Surface_Reflectance (16-bit integer)",NULL };
AppendMdd ( pszMddFile, pszSrcFile, pszDstFile, appendType, pszFormat, pfnProgress,
NULL, dstWkt, pdatasets);
```

A.7 合并 MDD

A.7.1 MergeMdd 函数定义

```
CPLErr CPL_DLL MergeMdd(const char * pszSrcFile1,
    const char * pszSrcFile2,
    const char * pszDstFile,
    char * pPhaseList1[],
    char * pPhaseList2[],
    EDataFormat pszFormat = E_TSB,
    GDALProgressFunc pfnProgress = NULL,
    void * pProgressArg = NULL );
```

A.7.2 MergeMDD 使用示例

示例数据说明:LandSat 时间序列数据
路径:Data/testFile

```
const char * pszSrcFile1 = "Data/testFile/testLandSat.mdd";
```

```
const char * pszSrcFile2 = "Data/testFile/testLandSat1.mdd";
const char * pszDstFile = "Data/testFile/testMergeLandSat.mdd";
char * pPhaseList1[] = {
        "MDD:0:Data/testFile/testLandSat.mdd",
        "MDD:1.LC80310362014002LGN00",
        "MDD:1:Data/testFile/testLandSat.mdd",
        "MDD:2.LC80310362014034LGN00", NULL };
char * pPhaseList2[] = {
        "MDD:1:Data/testFile/testLandSat1.mdd",
        "MDD:2.LC80310362014066LGN00", NULL };
EDataFormat pszFormat = E_TSB;
MergeMdd ( pszSrcFile1, pszSrcFile2, pszDstFile, pPhaseList1, pPhaseList2, pszFormat,
pfnProgress, pProgressArg);
```

A.7.3 MODIS 数据 MergeMDD 使用示例

```
const char * pszSrcFile1 = "Data/testFile/testMODIS.mdd";
const char * pszSrcFile2 = "Data/testFile/testMODIS1.mdd";
const char * pszDstFile = "Data/testFile/testMergeMODIS.mdd";
char * pPhaseList1[] = {
        "MDD:0:Data/testFile/testModis.mdd",
        "MDD:1.MOD09A1.A2011177.h08v05.005.2011202122415",
        "MDD:1:Data/testFile/testModis.mdd",
        "MDD:2.MOD09A1.A2011185.h08v05.005.2011210221828", NULL};
char * pPhaseList2[] = {
        "MDD:0:Data/testFile/testMODIS1.mdd",
        "MDD:1.MOD09A1.A2011001.h08v05.005.2011018175331",
        "MDD:1:Data/testFile/testMODIS1.mdd",
        "MDD:2.MOD09A1.A2011009.h08v05.005.2011025121701", NULL };
EDataFormat pszFormat = E_TSB;
MergeMdd ( pszSrcFile1, pszSrcFile2, pszDstFile, pPhaseList1, pPhaseList2, pszFormat,
pfnProgress, pProgressArg);
```

A.8 MDD 格式数据的指定时间读取

A.8.1 OpenMddFileWithTime 函数定义

```
CPL _ DLL MddImage * OpenMddFileWithTime ( MddImage * * mdd, const char *
strFileName, const char * timePhaseName, bool operate = true);
```

A.8.2 OpenMddFileWithTime 使用示例

```
示例数据说明:MODIS 时间序列数据
文件名:testMergeMODIS.mdd
路径:Data/testFile/testMergeMODIS.mdd
MddImage *mdd=new MddImage();
```

```
const char * strFileName= "Data/testFile/testMergeMODIS.mdd";
const char * timePhaseName = "2.2002035";
OpenMddFileWithTime(&mdd, strFileName, timePhaseName);
GDALDataset *dataser=mdd->subImageVector[0]->gdalDataset;
int len=GDALGetRasterCount(dataser);
```

A.9 数据格式生成

A.9.1 TSB 格式数据生成

```
void *pBuffer = VSIMalloc(nXSize * nPixelSize * nYSize);
for (int i = 0; i<nTimes && eErr == CE_None; i++)
{
    hDS = GDALOpen(papszFileNames[i], GA_ReadOnly);
    for (int b = 0; b<nBands; b++)
    {
        GDALRasterBandH hBand = GDALGetRasterBand(hDS, b + 1);
        eErr = GDALRasterIO(hBand, GF_Read, 0, 0, nXSize, nYSize, pBuffer, nXSize,
nYSize, eDT, 0, 0);
        VSIFWriteL(pBuffer, nPixelSize * nXSize * nYSize, 1, pMddFile);
        if (eErr == CE_None && ! pfnProgress(((b + 1.0) + i * nBands) / (nBands *
nTimes), "", pProgressArg))
        {
          CPLError(CE_Failure, CPLE_UserInterrupt, "User terminated");
          eErr = CE_Failure;
        }
      }
      GDALClose(hDS);
}
VSIFree(pBuffer);
```

A.9.2 TSP 格式数据生成

```
int *panBandCount = new int[nBands];
for (int i=0; i<nBands; i++)
    panBandCount[i] = i+1;
void *pBuffer = VSIMalloc(nXSize * nBands * nPixelSize);
int nPixelSpace = nPixelSize * nBands;
int nLineSpace = nPixelSpace * nXSize;
int nBandSpace = nPixelSize;
for(int i=0; i<nTimes && eErr == CE_None; i++)
{
      hDS = GDALOpen(papszFileNames[i], GA_ReadOnly);
      for (int j=0; j<nYSize && eErr == CE_None; j++)
      {
```

```
        eErr = GDALDatasetRasterIO(hDS, GF_Read, 0, j, nXSize, 1, pBuffer, nXSize,
1, eDT,
        nBands, panBandCount, nPixelSpace, nLineSpace, nBandSpace);
        VSIFWriteL(pBuffer, nPixelSize * nXSize * nBands, 1, pMddFile);
        if( eErr == CE_None && ! pfnProgress( ((j + 1.0) + i * nYSize) / (nYSize *
nTimes) , "", pProgressArg ) )
        {
          CPLError( CE_Failure, CPLE_UserInterrupt, "User terminated");
          eErr = CE_Failure;
        }
      }
      GDALClose(hDS);
}
VSIFree(pBuffer);
delete []panBandCount;
```

A.9.3 TIB 格式数据生成

```
void * pBuffer = VSIMalloc(nXSize * nYSize * nPixelSize);
int index = 0, mCount = 0;
for (int b = 0; b<nTimes * nBands && eErr == CE_None; b++)
{
      if (b%nTimes == 0)
          index++;
      hDS = GDALOpen(papszFileNames[mCount], GA_ReadOnly);
       eErr = GDALRasterIO(GDALGetRasterBand(hDS, index), GF_Read, 0, 0, nXSize,
nYSize, pBuffer, nXSize, nYSize, eDT, 0, 0);
      VSIFWriteL(pBuffer, nPixelSize * nXSize * nYSize, 1, pMddFile);
      mCount++;
      if (mCount >= nTimes)
      {
          mCount = 0;
      }
      if (eErr == CE_None && ! pfnProgress((b + 1.0) / (nBands * nTimes), "", pProgres-
sArg))
      {
          CPLError(CE_Failure, CPLE_UserInterrupt, "User terminated");
          eErr = CE_Failure;
      }
      GDALClose(hDS);
}
VSIFree(pBuffer);
```

A.9.4 TIP 格式数据生成

```
int nLines = nXSize * nPixelSize;
```

```
void * pBuffer = VSIMalloc(nTimes * nLines * nYSize);
void * pBufferOut = VSIMalloc(nTimes * nLines * nYSize);
void * * ppBufIndex = new void * [nTimes];
for (int t=0; t<nTimes; t++)
    ppBufIndex[t] = (char * )pBuffer + nLines * t * nYSize;
for (int b=0; b<nBands && eErr == CE_None; b++)
{
    for (int t=0; t<nTimes && eErr == CE_None; t++)
    {
        hDS = GDALOpen(papszFileNames[t], GA_ReadOnly);
        GDALRasterBandH hBand = GDALGetRasterBand(hDS, b+1);
         eErr = GDALRasterIO(hBand, GF_Read, 0, 0, nXSize, nYSize, ppBufIndex[t],
nXSize, nYSize, eDT, 0, 0);
        GDALClose(hDS);
    }
    int nIndex = 0;
    for (int i=0; i<nXSize * nYSize; i++)
    {
        for (int t=0; t<nTimes; t++)
        {
            void * pOut = (char * )pBufferOut + nIndex;
            memcpy(pOut, (char * )ppBufIndex[t] + i * nPixelSize, nPixelSize);
            nIndex += nPixelSize;
        }
    }
    VSIFWriteL(pBufferOut, nTimes * nLines * nYSize, 1, pMddFile);
    if( eErr == CE_None && ! pfnProgress( (b +1.0) / (nBands) , "", pProgressArg ) )
    {
        CPLError( CE_Failure, CPLE_UserInterrupt, "User terminated");
        eErr = CE_Failure;
    }
}
VSIFree(pBuffer);
VSIFree(pBufferOut);
delete []ppBufIndex;
```

A.9.5 TIS 数据格式生成

```
int nLines = nXSize * nBands * nPixelSize;
void * pBuffer = VSIMalloc(nTimes * nLines * nYSize);
void * pBufferOut = VSIMalloc(nTimes * nLines * nYSize);
void * * ppBufIndex = new void * [nTimes];
for (int t=0; t<nTimes; t++)
    ppBufIndex[t] = (char * )pBuffer + nLines * t * nYSize;
```

```
int * panBandCount = new int[nBands];
for (int i=0; i<nBands; i++)
    panBandCount[i] = i+1;
int nPixelSpace = nPixelSize * nBands;
int nLineSpace = nPixelSize * nXSize * nBands;
int nBandSpace = nPixelSize;
for (int t = 0; t<nTimes && eErr == CE_None; t++)
{
    hDS = GDALOpen(papszFileNames[t], GA_ReadOnly);
    eErr = GDALDatasetRasterIO(hDS, GF_Read, 0, 0, nXSize, nYSize, ppBufIndex[t],
nXSize, nYSize, eDT, nBands, panBandCount, nPixelSpace, nLineSpace, nBandSpace);
    if (eErr == CE_None && ! pfnProgress(((t+0.01) / nTimes) * 0.8, "", pProgressArg))
    {
        CPLError(CE_Failure, CPLE_UserInterrupt, "User terminated");
        eErr = CE_Failure;
    }
    GDALClose(hDS);
}
int nIndex = 0;
for (int i = 0; i<nXSize * nYSize; i++)
{
    for (int t = 0; t<nTimes; t++)
    {
        void * pOut = (char * )pBufferOut + nIndex;
        memcpy(pOut, (char * )ppBufIndex[t] + i * nPixelSize * nBands, nPixelSize * nBands);
        nIndex += nPixelSize * nBands;
    }
}
VSIFWriteL(pBufferOut, nTimes * nLines * nYSize, 1, pMddFile);
pfnProgress(1.0, "", pProgressArg);
VSIFree(pBuffer);
VSIFree(pBufferOut);
delete []ppBufIndex;
    delete []panBandCount;
```

A.10 数据格式转换

```
GDALAllRegister();
GDALDataset * pDS = (GDALDataset * )GDALOpen(m_strInput.toLocal8Bit(), GA_ReadOnly);
if(pDS == NULL)
{
    QMessageBox::information(this, tr("Information"),  tr("Can't open input file"));
    return;
```

```
}

char * * papszSubdatasets = pDS->GetMetadata("SUBDATASETS");
if(papszSubdatasets == NULL)
{
    GDALClose((GDALDatasetH)pDS);
    QMessageBox::information(this, tr("Information"),  tr("Can't find subdatasets in input
file"));
    return;
}

int iCount = CSLCount(papszSubdatasets);
if (iCount<1)
{
    GDALClose((GDALDatasetH)pDS);
    QMessageBox::information(this, tr("Information"),  tr("Can't find subdatasets in input
file"));
    return;
}

char * * papszSubFiles = NULL;
char * * papszSubDescs = NULL;
for (int i=0; papszSubdatasets[i]! =NULL; i++)
{
    string strSubDatasetName = papszSubdatasets[i];
    strSubDatasetName = strSubDatasetName.substr(strSubDatasetName.find_first_of("=") + 1);

    if(i%2 == 0)
    papszSubFiles = CSLAddString(papszSubFiles, strSubDatasetName.c_str());
    else
    {
        strSubDatasetName = strSubDatasetName.substr(4);
        papszSubDescs = CSLAddString(papszSubDescs, strSubDatasetName.c_str());
    }
}

GDALClose((GDALDatasetH)pDS);

CPLSetConfigOption("GDAL_FILENAME_IS_UTF8", "NO");
if( pfnProgress == NULL )
pfnProgress = GDALDummyProgress;

if( ! pfnProgress( 0.0, "Building MDD file...", pProgressArg ) )
```

```
{
    CPLError( CE_Failure, CPLE_UserInterrupt, "User terminated");
    return CE_Failure;
}

if(papszFileNames == NULL)
{
    CPLError( CE_Failure, CPLE_IllegalArg, "Input data file list is NULL");
    return CE_Failure;
}

int nTimes = CSLCount(papszFileNames);
if(nTimes <= 0)
{
    CPLError( CE_Failure, CPLE_IllegalArg, "Input data file list is NULL");
    return CE_Failure;
}

if(papszTimeDescs ! = NULL)
{
    if(CSLCount(papszTimeDescs) ! = nTimes)
    {
        CPLError( CE_Failure, CPLE_IllegalArg, "Input time description list count is not equal file
size");
        return CE_Failure;
    }
}

if(pszFilename == NULL)
{
    CPLError( CE_Failure, CPLE_IllegalArg, "Output filename is NULL");
    return CE_Failure;
}

GDALAllRegister();
GDALDatasetH parDataH = NULL;;
GDALDatasetH tempDataH = NULL;;
GDALDatasetH hDS = NULL;
// 定义一些变量用于判断所有的数据是否大小等信息一样
int nYSize = 0;
int nXSize = 0;
int nBands = 0;
double adfGeoTransform[6] = {0};
```

```
CPLString strWkt = "";
GDALDataType eDT = GDT_Unknown;

char * * papszWavelength = NULL;
string strWavelengthUnit = "Unknown";
string strSensorType = "Unknown";

bool bError = false;
for(int i=0; i<nTimes; i++)
{
    if(bError)
        break;

    if(i == 0)
    {
        parDataH = GDALOpen(papszFileNames[i], GA_ReadOnly);
        nXSize = GDALGetRasterXSize(parDataH);
        nYSize = GDALGetRasterYSize(parDataH);
        nBands = GDALGetRasterCount(parDataH);
        if(nBands == 0 || nXSize==0 || nYSize==0)
        {
            CPLDebug("MDDDataset", "File [%s] can not get raster information.", papsz-
FileNames[i]);
            bError = true;
            break;
        }

        GDALGetGeoTransform(parDataH, adfGeoTransform);
        strWkt = GDALGetProjectionRef(parDataH);
        GDALRasterBandH hBand = GDALGetRasterBand(parDataH, 1);
        if(hBand == NULL)
        {
            CPLDebug("MDDDataset", "File [%s] can not get raster band 1.", papszFileNames
[i]);
            bError = true;
            break;
        }

        const char * pszShortName = GDALGetDriverShortName(GDALGetDatasetDriver(parDa-
taH));  //TODO ??
        const char * pszSensorName = GDALGetMetadataItem(parDataH, "sensor_type",
pszShortName);
        if(pszSensorName != NULL)
```

```
    strSensorType = pszSensorName;

  eDT = GDALGetRasterDataType(hBand);
  char ** papszMetadata = GDALGetMetadata( hBand, NULL );
  if(CSLCount(papszMetadata) > 0)
  {
    for(int j = 0; papszMetadata[j] != NULL; j++ )
    {
      if(EQUALN(papszMetadata[j], "wavelength_units=", 17))
      {
        strWavelengthUnit = papszMetadata[j] + 17;
        break;
      }
    }
  }

  for (int b=1; b<=nBands; b++)
  {
    hBand = GDALGetRasterBand(parDataH, b);

    papszMetadata = GDALGetMetadata( hBand,"SUBDATASETS");
    if(CSLCount(papszMetadata) > 0)
    {
      for( i = 0; papszMetadata[i] != NULL; i++ )
      {
        if(EQUALN(papszMetadata[i], "wavelength=", 11))
        {
          string strWavelength = papszMetadata[i] + 11;
          papszWavelength = CSLAddString(papszWavelength, strWavelength.c_str());
          break;
        }
      }
    }
  }
}
else
{
  tempDataH = GDALOpen(papszFileNames[i], GA_ReadOnly);  //每次打开
  if (tempDataH == NULL)
  {
    CPLDebug("MDDDataset", "File [%s] can not open.", papszFileNames[i]);
    bError = true;
    break;
```

```
        }

        int testX = GDALGetRasterXSize(tempDataH);
        if (nXSize ! = testX)
        {
          bError = true;
          CPLError(CE_Failure, CPLE_IllegalArg, "Image width is not consistent.\nPlease try to
select the band.");
          break;
        }

        if (nYSize ! = GDALGetRasterYSize(tempDataH))
        {
          bError = true;
          CPLError(CE_Failure, CPLE_IllegalArg, "Image height is not consistent.\nPlease try to
select the band.");
          break;
        }

        if (nBands > GDALGetRasterCount(tempDataH))
        {
          bError = true;
          CPLError(CE_Failure, CPLE_IllegalArg, "Image Band is not consistent.\nPlease try to
select the band.");
          break;
        }

        if (eDT ! = GDALGetRasterDataType(GDALGetRasterBand(tempDataH, 1)))
        {
          bError = true;
          CPLError(CE_Failure, CPLE_IllegalArg, "Image data type is not consistent.\nPlease try
to select the band.");
          break;
        }

        double adfGeoTransform1[6] = { 0 };
        GDALGetGeoTransform(tempDataH, adfGeoTransform1);
        for (int j = 0; j<6; j++)
        {
          if (adfGeoTransform[j] ! = adfGeoTransform1[j]) {
            CPLError(CE_Failure, CPLE_IllegalArg, "Image Range of four to four(adfGeoTrans-
form) is not consistent.\nPlease try to select the band.");
            bError = true;
```

```
            break;
          }
        }
        if (bError)
          break;
        GDALClose(tempDataH);//每次销毁
      }
}

if(bError)
{

      if(papszWavelength ! = NULL)
        CSLDestroy(papszWavelength);
      if (tempDataH ! = NULL)
      {
        GDALClose(tempDataH);
      }
      if (parDataH ! = NULL)
      {
        GDALClose(parDataH); parDataH = NULL;
      }
      return CE_Failure;
    }

    if (pszFormat == NULL)
    {
      CPLDebug("MDDDataset", "Output format is NULL, assgin to TSB format");
    }

    char * eType = "TSB";
    getTDTType(pszFormat, &eType);

    /* -------------------------------------------------------------------- */
    /*      Verify input options.                                           */
    /* -------------------------------------------------------------------- */
    int  iENVIType = GetEnviType(eDT);
    if (0 == iENVIType)
    {
      GDALClose(parDataH);
      if (papszWavelength ! = NULL)
        CSLDestroy(papszWavelength);
```

```
        CPLError( CE_Failure, CPLE_IllegalArg, "Input data type is not support");
        return CE_Failure;
}

/* -------------------------------------------------------------------- */
/*      Try to create the mdr file.                                     */
/* -------------------------------------------------------------------- */
const char   * pszHDRFilename;
pszHDRFilename = CPLResetExtension(pszFilename, "mdr");

VSILFILE * pHdrFile;
pHdrFile = VSIFOpenL( pszHDRFilename, FILE_RESERVATION_WT);
if( pHdrFile == NULL )
{
    GDALClose(parDataH);
    if (papszWavelength ! = NULL)
        CSLDestroy(papszWavelength);

    CPLError( CE_Failure, CPLE_OpenFailed, "Attempt to create file `%s' failed.\n", pszFilename );
    return CE_Failure;
}

/* -------------------------------------------------------------------- */
/*      Write out the header.                                           */
/* -------------------------------------------------------------------- */
int     iBigEndian;
CPLString sDescription = "MDD Dataset";

#ifdef CPL_LSB
    iBigEndian = 0;
#else
    iBigEndian = 1;
#endif

    VSIFPrintfL( pHdrFile, "MDD\n");
    if (""! = sDescription)
        VSIFPrintfL( pHdrFile, "description = {\n%s}\n", sDescription.c_str());

    VSIFPrintfL( pHdrFile, "samples = %d\nlines = %d\nbands = %d\ntimes = %d\
n", nXSize, nYSize, nBands, nTimes );
    VSIFPrintfL( pHdrFile, "header offset = 0\nfile type = MDD Standard\n");
    VSIFPrintfL( pHdrFile, "data type = %d\n", iENVIType );
    VSIFPrintfL( pHdrFile, "interleave = %s\n", eType);
```

```
VSIFPrintfL( pHdrFile, "sensor type = %s\n", strSensorType.c_str());
VSIFPrintfL( pHdrFile, "byte order = %d\n", iBigEndian );

WriteProjectionInfo(pHdrFile, adfGeoTransform, strWkt.c_str());

/* -------------------------------------------------------------------- */
/*      Write the band names header.                                    */
/* -------------------------------------------------------------------- */
VSIFPrintfL( pHdrFile, "band names = {\n");
for ( int i = 1; i <= nBands; i++ )
{
    CPLString sBandDesc = GDALGetDescription(GDALGetRasterBand(parDataH, i));

    if (sBandDesc == "")
    {
      sBandDesc = CPLSPrintf("Band %d", i);
      if (papszSubBands! =NULL&&papszSubBands[i-1] ! = "")
        sBandDesc =papszSubBands[i-1];
    }

    VSIFPrintfL( pHdrFile, "%s", sBandDesc.c_str() );
    if ( i ! = nBands )
      VSIFPrintfL( pHdrFile, ",\n");
  }
  VSIFPrintfL( pHdrFile, "}\n");

  /* -------------------------------------------------------------------- */
  /*      Write the wavelength header.                                    */
  /* -------------------------------------------------------------------- */
  if(papszWavelength ! = NULL)
  {
    VSIFPrintfL( pHdrFile, "wavelength = {\n");
    for ( int i = 1; i <= nBands; i++ )
    {
      VSIFPrintfL( pHdrFile, "%s", papszWavelength[i-1] );
      if ( i ! = nBands )
        VSIFPrintfL( pHdrFile, ",\n");
    }
    VSIFPrintfL( pHdrFile, "}\n");
    if(strWavelengthUnit.length() >0)
      VSIFPrintfL( pHdrFile, "wavelength units=%s\n", strWavelengthUnit.c_str());
    CSLDestroy(papszWavelength);
  }
```

```
        /* -------------------------------------------------------------------- */
        /*      Write the time names header.                                    */
        /* -------------------------------------------------------------------- */
        VSIFPrintfL( pHdrFile, "time names = {\n");
        for ( int i = 1; i <= nTimes; i++ )
        {
          CPLString sTimeDesc = CPLGetBasename(papszFileNames[i-1]);

          if(papszTimeDescs ! = NULL)
            sTimeDesc = papszTimeDescs[i-1];

          //如果之前没有加序号,这里添加序号。
          if (! checkSerialNumber(sTimeDesc))
          {
            std::string tempStr,tempChar;
            stringstream stream;
            stream << i;
            tempStr = stream.str();
            tempStr += ".";
            tempChar = sTimeDesc;
            tempStr += tempChar;
          sTimeDesc = tempStr;
        }

        if ( sTimeDesc == "")
          sTimeDesc = CPLSPrintf( "Time %d", i );
        VSIFPrintfL( pHdrFile, "%s", sTimeDesc.c_str() );
        if ( i ! = nTimes )
          VSIFPrintfL( pHdrFile, ",\n");
}
VSIFPrintfL( pHdrFile, "}\n");

VSIFCloseL( pHdrFile );

/* -------------------------------------------------------------------- */
/*      Try to create the mdd file.                                     */
/* -------------------------------------------------------------------- */

VSILFILE *pMddFile;//创建一个指向准 MDD 文件。
pMddFile = VSIFOpenL( pszFilename, "wb");
if( pMddFile == NULL )
{
```

```
    GDALClose(parDataH);
    CPLError(CE_Failure, CPLE_OpenFailed, "Attempt to create file `%s' failed.\n", pszHDR-
Filename);
    return CE_Failure;
}

CPLErr eErr = CE_None;
int nPixelSize = GDALGetDataTypeSize(eDT) / 8;
    switch(pszFormat)
    {
    case TDT_TSB:
    default:
        {
          void * pBuffer = VSIMalloc(nXSize * nPixelSize * nYSize);//每次读取一行

          for (int i = 0; i<nTimes && eErr == CE_None; i++)
          {
            hDS = GDALOpen(papszFileNames[i], GA_ReadOnly);  //假设所有的数据都可以
打开
            for (int b = 0; b<nBands; b++)
            {
              GDALRasterBandH hBand = GDALGetRasterBand(hDS, b + 1);
              //读取影像数据
                eErr = GDALRasterIO(hBand, GF_Read, 0, 0, nXSize, nYSize, pBuffer,
nXSize, nYSize, eDT, 0, 0);
              //影像数据写入准 MDD 文件 pMddFile
              VSIFWriteL(pBuffer, nPixelSize * nXSize * nYSize, 1, pMddFile);
              if (eErr == CE_None
                && ! pfnProgress(((b + 1.0) + i * nBands) / (nBands * nTimes), "", pPro-
gressArg))
              {
                CPLError(CE_Failure, CPLE_UserInterrupt, "User terminated");
                eErr = CE_Failure;
              }
            }
            GDALClose(hDS);
          }
          VSIFree(pBuffer);
        }
        break;
      case TDT_TSP:
        {
          int * panBandCount = new int[nBands];
```

```
            for (int i=0; i<nBands; i++)
              panBandCount[i] = i+1;

            void *pBuffer = VSIMalloc(nXSize * nBands * nPixelSize);//每次读取一行

            int nPixelSpace = nPixelSize * nBands;
            int nLineSpace = nPixelSize * nXSize * nBands;
            int nBandSpace = nPixelSize;

            for(int i=0; i<nTimes && eErr == CE_None; i++)
            {
              hDS = GDALOpen(papszFileNames[i], GA_ReadOnly);  //假设所有的数据都可以
打开
              for (int j=0; j<nYSize && eErr == CE_None; j++)
              {
                  eErr = GDALDatasetRasterIO(hDS, GF_Read, 0, j, nXSize, 1, pBuffer,
nXSize, 1, eDT,
                  nBands, panBandCount, nPixelSpace, nLineSpace, nBandSpace);

                VSIFWriteL(pBuffer, nPixelSize * nXSize * nBands, 1, pMddFile);

                if( eErr == CE_None
                  && ! pfnProgress( ((j+1.0) + i * nYSize) / (nYSize * nTimes) , "", pProgres-
sArg ) )
                {
                  CPLError( CE_Failure, CPLE_UserInterrupt, "User terminated");
                  eErr = CE_Failure;
                }
              }
              GDALClose(hDS);
            }

            VSIFree(pBuffer);
            delete []panBandCount;
          }
          break;
        case TDT_TIB:
          {
            void *pBuffer = VSIMalloc(nXSize * nYSize * nPixelSize);//每次读取一波段
            int index = 0, mCount = 0;
            for (int b = 0; b<nTimes * nBands && eErr == CE_None; b++)
            {
              if (b%nTimes == 0)
```

```
            index++;
          //DatasetH 对象需要一次性使用,不能连续 open。所以针对需要的 DatasetH 对象获取需要的 paBands。
          hDS = GDALOpen(papszFileNames[mCount], GA_ReadOnly);
            eErr = GDALRasterIO(GDALGetRasterBand(hDS, index), GF_Read, 0, 0, nXSize, nYSize, pBuffer, nXSize, nYSize, eDT, 0, 0);
          VSIFWriteL(pBuffer, nPixelSize * nXSize * nYSize, 1, pMddFile);
          mCount++;
          if (mCount >= nTimes)
          {
            mCount = 0;
          }
          if (eErr == CE_None
            && ! pfnProgress((b + 1.0) / (nBands * nTimes), "", pProgressArg))
          {
            CPLError(CE_Failure, CPLE_UserInterrupt, "User terminated");
            eErr = CE_Failure;
          }
          GDALClose(hDS);
        }
        VSIFree(pBuffer);
      }
      break;
    case TDT_TIP:
      {
        int nLines = nXSize * nPixelSize;
        void * pBuffer = VSIMalloc(nTimes * nLines * nYSize);//每次读取一行,共 nTimes 个数据
        void * pBufferOut = VSIMalloc(nTimes * nLines * nYSize);

        void * * ppBufIndex = new void * [nTimes];
        for (int t=0; t<nTimes; t++)
          ppBufIndex[t] = (char * )pBuffer + nLines * t * nYSize;

        for (int b=0; b<nBands && eErr == CE_None; b++)
        {
          /* for (int h=0; h<nYSize && eErr == CE_None; h++)
          { */
            for (int t=0; t<nTimes && eErr == CE_None; t++)
            {
              hDS = GDALOpen(papszFileNames[t], GA_ReadOnly);  //假设所有的数据都可以打开

              GDALRasterBandH hBand = GDALGetRasterBand(hDS, b+1);
```

```
                eErr = GDALRasterIO(hBand, GF_Read, 0, 0, nXSize, nYSize, ppBufIndex
[t], nXSize, nYSize, eDT, 0, 0);
                GDALClose(hDS);
              }

              //重新排列
              int nIndex = 0;
              for (int i=0; i<nXSize * nYSize; i++)
              {
                for (int t=0; t<nTimes; t++)
                {
                  void * pOut = (char * )pBufferOut + nIndex;
                  memcpy(pOut, (char * )ppBufIndex[t] + i * nPixelSize, nPixelSize);
                  nIndex += nPixelSize;
                }
              }

              VSIFWriteL(pBufferOut, nTimes * nLines * nYSize, 1, pMddFile);

              if( eErr == CE_None
                && ! pfnProgress( (b +1.0) / (nBands) , "", pProgressArg ) )
              {
                CPLError( CE_Failure, CPLE_UserInterrupt, "User terminated");
                eErr = CE_Failure;
              }
            //}
          }

          VSIFree(pBuffer);
          VSIFree(pBufferOut);
          delete []ppBufIndex;
        }
        break;
      case TDT_TIS:
        {
          int nLines = nXSize * nBands * nPixelSize;//每次读取一行,共 nTimes 个数据
          void * pBuffer = VSIMalloc(nTimes * nLines * nYSize);
          void * pBufferOut = VSIMalloc(nTimes * nLines * nYSize);

          void * * ppBufIndex = new void * [nTimes];
          for (int t=0; t<nTimes; t++)
            ppBufIndex[t] = (char * )pBuffer + nLines * t * nYSize;
```

```
            int * panBandCount = new int[nBands];
            for (int i=0; i<nBands; i++)
              panBandCount[i] = i+1;

            int nPixelSpace = nPixelSize * nBands;
            int nLineSpace = nPixelSize * nXSize * nBands;
            int nBandSpace = nPixelSize;

            // 读取所有时间段的数据,每次读取一行数据
            for (int t = 0; t<nTimes && eErr == CE_None; t++)
            {
              hDS = GDALOpen(papszFileNames[t], GA_ReadOnly);
               eErr = GDALDatasetRasterIO(hDS, GF_Read, 0, 0, nXSize, nYSize, ppBufIndex
[t], nXSize, nYSize, eDT,
                nBands, panBandCount, nPixelSpace, nLineSpace, nBandSpace);

              if (eErr == CE_None
                && ! pfnProgress(((t+0.01) / nTimes) * 0.8, "", pProgressArg))
              {
                CPLError(CE_Failure, CPLE_UserInterrupt, "User terminated");
                eErr = CE_Failure;
              }
              GDALClose(hDS);
            }

            //重新排列
            int nIndex = 0;
            for (int i = 0; i<nXSize * nYSize; i++)
            {
              for (int t = 0; t<nTimes; t++)
              {
                void * pOut = (char * )pBufferOut + nIndex;
                 memcpy(pOut, (char * ) ppBufIndex[t] + i * nPixelSize * nBands, nPixelSize *
nBands);
                nIndex += nPixelSize * nBands;
              }
            }

            VSIFWriteL(pBufferOut, nTimes * nLines * nYSize, 1, pMddFile);
            pfnProgress(1.0, "", pProgressArg);

            VSIFree(pBuffer);
            VSIFree(pBufferOut);
```

```
            delete []ppBufIndex;
            delete []panBandCount;
        }
        break;
    }
    GDALClose(parDataH);
    VSIFCloseL( pMddFile );

    if( ! pfnProgress( 1.0, "", pProgressArg ) )
    {
        CPLError( CE_Failure, CPLE_UserInterrupt, "User terminated");
        return CE_Failure;
    }
```

附 录 B
（资料性）
栅格与矢量数据构建 MDD 流程

B.1 栅格数据构建 MDD

栅格数据构建 MDD 之前需要有投影坐标信息，以高分一号数据为例进行说明。

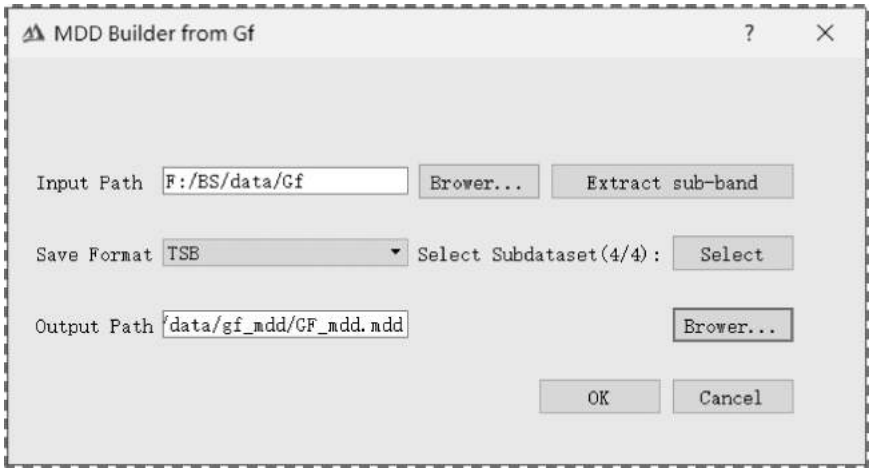


图 B.1 高分数据 MDD 构建主界面

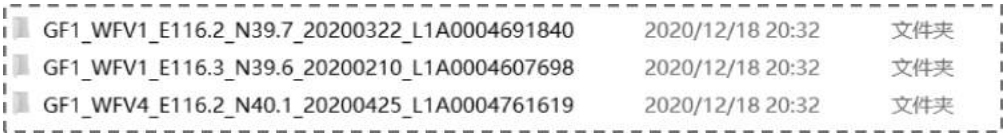


图 B.2 样例数据

a) 如图 B.1 所示，每一时相的高分数据用一个文件夹保存，所有时相的高分数据存放于同一文件夹下，该文件夹的路径即为图 B.2 中的数据输入路径。
b) 点击“Extract sub-band”进行多光谱数据子波段的提取；这一步主要将子波段数据提取出来；
c) 选择保存格式，共有五种格式可供选择：TSB、TSP、TIB、TIP 与 TIS；
d) 选择子波段数据集，本次试验数据每个多光谱数据包含了四个子波段，此处全选了子波段数据集，如图 B.1 中的“Select Subdataset(4/4)”；
e) 最后一步选择 MDD 的输出路径；上述操作完成后点击“OK”即可进行高分 MDD 的构建了；
f) 如图 B.3 所示，为构建的高分数据 MDD 的头文件.mdr 与数据体文件.mdd。

图 B.3 高分数据 MDD 构建结果

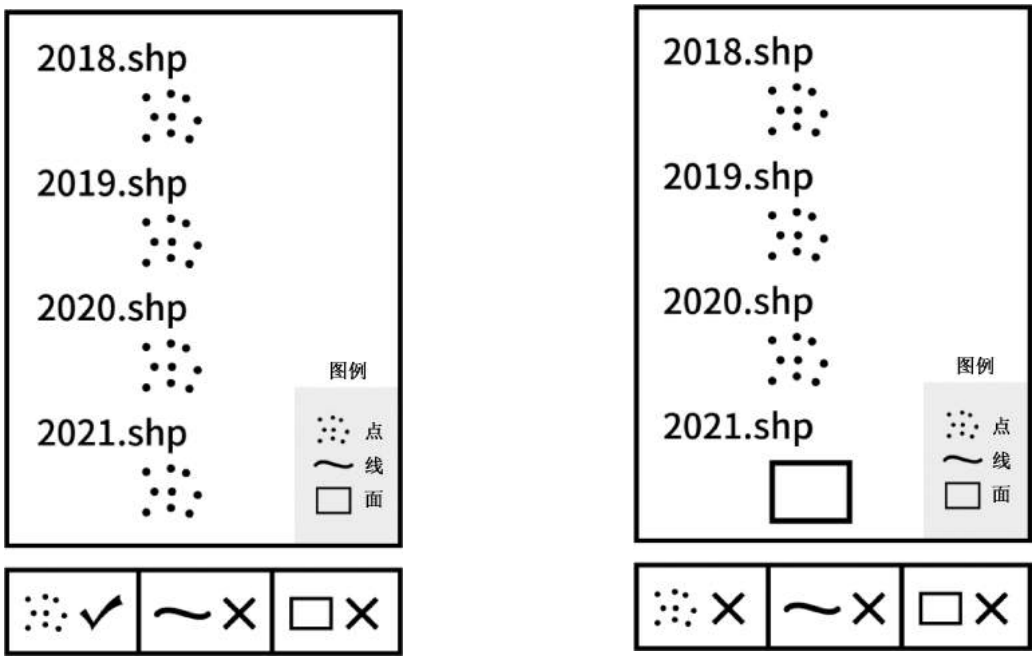
B.2 矢量数据构建 MDD

B.2.1 概述

矢量数据与栅格数据不同，它没有光谱维的概念。但 Shp 矢量数据分为点、线、面 3 种类型，每种类型可以看作遥感影像产品的一个波段，因此，可以借鉴遥感数据构建 MDD 的方式去进行矢量数据 MDD 的构建。

B.2.2 构建规则

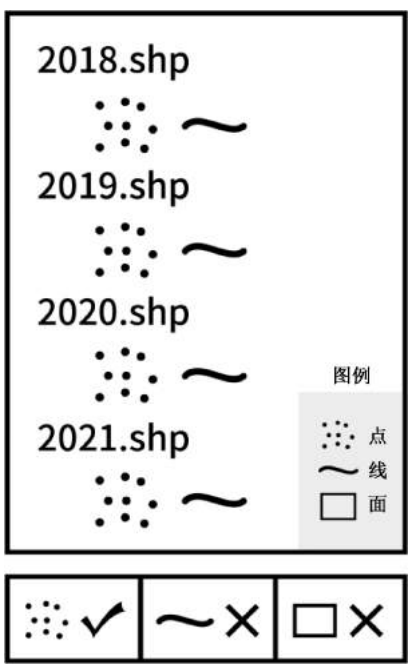
矢量数据在构建 MDD 时以第一个时相数据所包含波段为标准；参与构建的波段在每个时相上都须存在，否则该波段不能参与 MDD 构建。图 B.4、图 B.5、图 B.6 列举了矢量数据构建 MDD 时输入数据的几种情况（此处只是对输入数据情况的三种举例，并不是三种构建格式）：



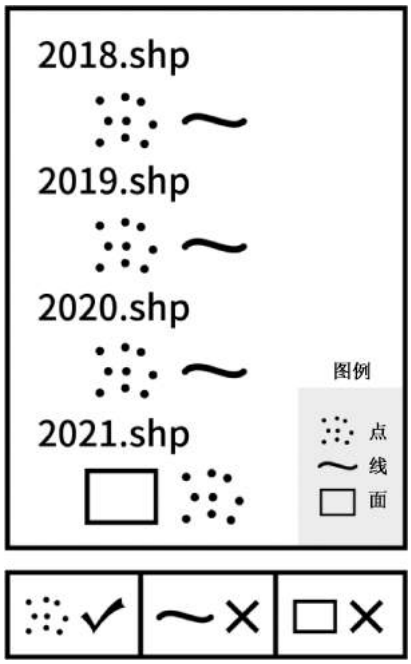
a） 每个时相上均包含点要素，可以构建 MDD

b） 不能构建 MDD

图 B.4 构建 MDD 数据集的 shp 格式类型 A

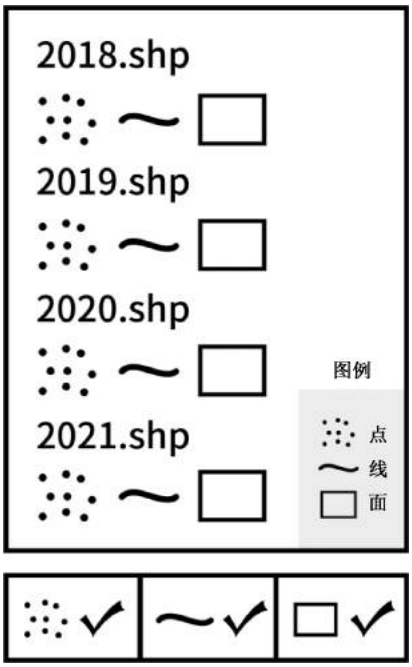


a） 每个时相上均包含点、线要素，可以构建 MDD

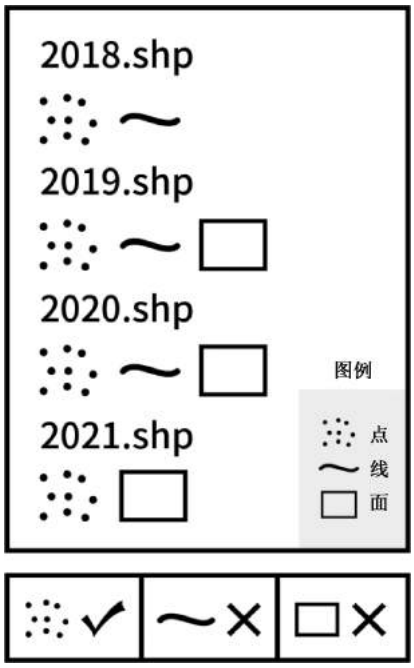


b） 只能构建点要素 MDD

图 B.5 构建 MDD 数据集的 shp 格式类型 B



a） 每个时相上均包含点、线、面要素，可以构建 MDD



b） 只能构建点要素 MDD

图 B.6 构建 MDD 数据集的 shp 格式类型 C

B.2.3 注意事项

矢量数据构建 MDD 时，应注意以下事项：

a) 矢量数据构建 MDD 只支持 TSB 格式排列。因为矢量数据没有规则行列及像元的概念，所以数据大小不统一导致其他规则排列会出现数据移位的问题；

b) 矢量数据构建 MDD 后的展示规则遵循 Shp 数据展示标准，可以参考 ENVI 工具。代码库以 GDAL 为标准，数据操作以 GDAL 库内 API 为准。

参 考 文 献

[1] 张立福,陈浩,孙雪剑,等.多维遥感数据时空谱一体化存储结构设计[J].遥感学报,2017,21(1):62-73.

[2] 张立福,王飒,刘华亮,等.从光谱到时谱——遥感时间序列变化检测研究进展[J].武汉大学学报,2021,46(4):451-468.

[3] 张立福,孙雪剑,张霞,等.时空谱多维数据格式(MDD)结构与计算机配套系统[J].全球变化数据学报,2017,1(2):121-135.

[4] 张立福,孙雪剑,张霞,等.遥感多维数据格式互操作分析软件系统(MARS 2.03)[DB/OL].全球变化科学研究数据出版系统,2017.DOI:10.3974/geodb.2017.02.20.V1.

[5] GB/T 14950—2009 摄影测量与遥感术语

[6] GB/T 17694—2009 地理信息 术语

[7] GB/T 36300—2018 遥感卫星快视数据格式规范

[8] ISO/TS 19101-2:2018 Geographic information—Reference model—Part 2:Imagery