附录 C
用 ParaView 进行后处理

ParaView[118] 是基于 VTK (Visualization Toolkit) 开发的对二维和三维数据进行分析和可视化的程序。它既是一个应用程序框架，也可以直接使用。ParaView 不仅可以运行于单处理器的工作站，也可以基于 MPI 运行于分布式存储器的大型计算集群。因此它可以处理极大规模的数据，且功能强大、操作简单灵活。此外，ParaView 是开源、跨平台软件，用户可直接从其网站上 http://www.paraview.org 下载源代码或预编译的可执行文件，依据安装说明进行安装后即可使用。本附录只着重介绍基于单处理器模式的 ParaView 进行有限元法和无网格法后处理的功能，和两类 ParaView 可以读取的数据文件格式 (vtk 和 vtu)。相比于 TECPLOT，ParaView 除了对导入的数据可视化，还可以基于原始数据做多种多样的数据提取和再分析。

ParaView 的界面如图C.1所示，由菜单栏、工具栏、工作区组成。工作区可由任意个模块组成，由菜单栏 View 中的选项控制。在图C.1中，工作区由可视化管道列表区、对象属性区和视图区组成。

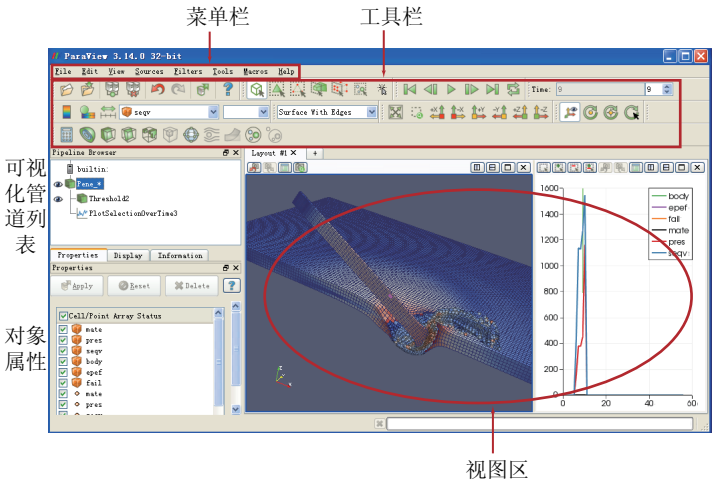


图 C.1　ParaView 界面

可视化管道列表区 (Pipeline Browser) 是以树状结构的形式列出程序导入和提取的所有子数据对象的名称浏览表 (子数据对象按其创建的先后顺序列在母数据对象的下边，并

缩进)。每个数据对象前有眼形图标，用于激活当前的数据对象。

对象属性 (Object Inspector) 区是可视化管道列表中处于激活状态的数据对象各类属性，包括三个属性页：Properties、Diaplay 和 Information。其中 Properties 显示该数据对象所具有的变量名称，数据的导入和删除以及重置均可在此实现；Display 则提供了该数据可视化方面的设置；Information 显示了该该数据对象的一些状态和统计信息。

视图区 (Display Area) 是可视化管道列表中数据对象的可视化区。用户在此可以查看、探测数据，并能实现对数据的交互操作。该区域可以进一步切分成多个不同类型的视图 (View) ，并在不同视图中显示不同的计算结果。每个视图仅显示处于激活状态的数据对象，也可同时显示多个处于激活状态的数据对象。

更加详细的功能介绍和操作流程请参考《ParaView 指南》[118] 和程序帮助文档。下面先从一个实例介绍 ParaView 作为有限元法和无网格法后处理软件时的常用功能。

C.1　一个实例

本实例是自适应物质点有限元法的一个算例结果文件[119]。该算例为一个长杆弹斜侵彻薄钢板的问题，初始采用有限元离散各个物体，在计算过程中将发生畸变的单元转化为质点并采用物质点法模拟。因此该算例文件结果中既包含有限元法的网格数据，也包含无网格法的质点数据。与 TecoPlot 的后处理文件不同的是，ParaView 的每个文件仅包含一个 Zone，即一个单步的计算结构数据。ParaView 可一次性读入所有文件，从而形成一桢帧的动画。

下面分别从对导入数据观察物理量云图和对导入的数据进行再分析两个方面介绍 ParaView 的后处理功能。导入数据观察物理量云图，可分为 4 步操作：

第 1 步，读入数据。在菜单栏中单击 File - Open 读入一系列数据文件 Pene_*.vtu。此时，在可视化管道列表区会出现一个数据对象 Pene_*，也可以通过鼠标双击重新命名。此时，该数据对象处于激活状态，单击其对象属性 Properties 页的 Apply 按钮，即可完成数据载入过程。同时，在视图区按照默认的显示方式显示该数据模型。

第 2 步，选择显示方式。该项操作在 Representation Toolbar 工具栏中完成。如果没有找到 Representation Toolbar 工具栏，可单击菜单栏 View - Representation Toolbar 加以显示。该工具栏的下拉菜单中有多种显示方式可选，如果后处理文件仅是质点类型的数据，则选择 Points；如果是网格类型的数据，则选择 Surface With Edges。此外，用户还可以添加额外的显示方式，如可为质点类型的数据添加 Point Sprite 的显示方式。使用 Point Sprite 时需要先载入相应的插件，可在菜单栏中单击 Tools - Manage Plugins 中设置。对于本算例，在该工具图标下拉菜单中选择 Surface With Edges，可以得到图C.2所示的结果。

第 3 步，调整视图视角。调整视图视角有多种方式，一是通过鼠标，将鼠标放在视图区，按住鼠标右键可缩放 (滚轮亦可) ，按住左键可旋转图形；二是通过 Camera Control 工具栏，可通过单击菜单栏 View - Toolbars - Camera Controls 显示该工具栏。此外，视图区左上角有四个图标，可对显示的图层进行各种视图及背景颜色的调整等；视图区的右

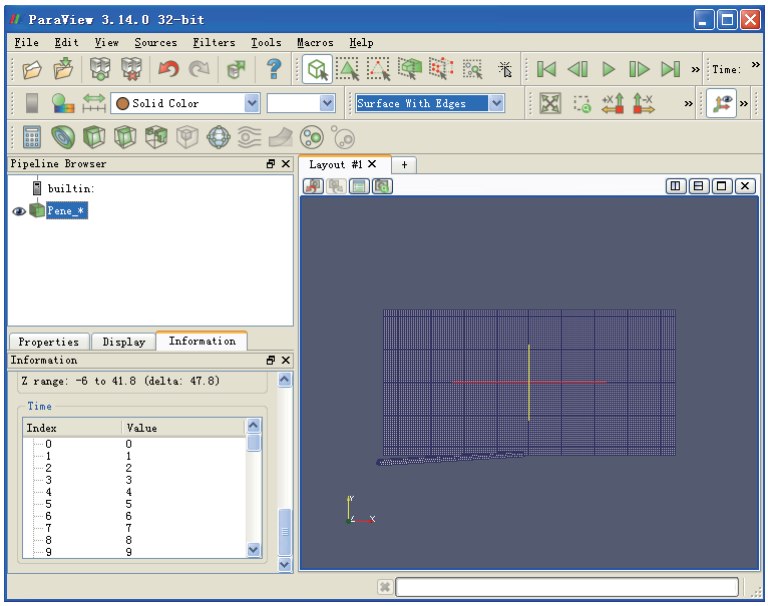


图 C.2 三维有限元离散图

上角有四个图标，可对显示区进行分割、放大和删除。

第 4 步，选择显示的变量。该项操作可通过 Active Variable Controls 工具栏完成，可通过单击菜单栏 View - Toolbars - Active Variable Controls 显示该工具栏。在该图标的下拉菜单中可选择显示的物理量，其中带六面体图标的物理量是属于单元的，带实点图标的物理量是属于质点的。通过该工具栏中的其他按钮可为当前显示的物理量选择颜色图例。此时，可通过单击动画播放控制工具栏的播放按钮，观察整个侵彻过程的计算结果。用户可单击菜单栏 File 下的相关按钮，保存动画或图片。图C.3为输出的第 6 帧的计算结果图片，其中显示的是单元等效塑性应变 epef 的云图。

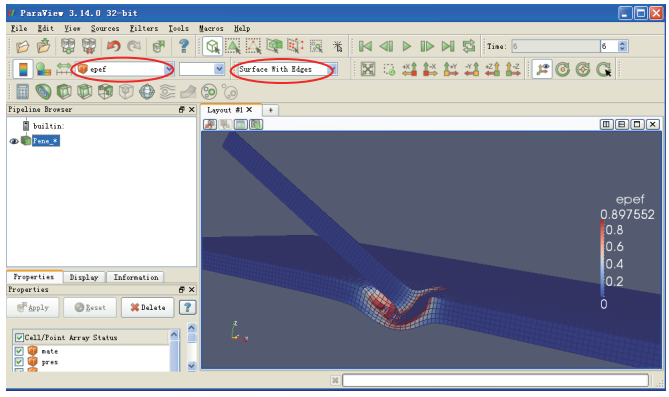


图 C.3 Pene* 数据对象第 6 帧显示结果

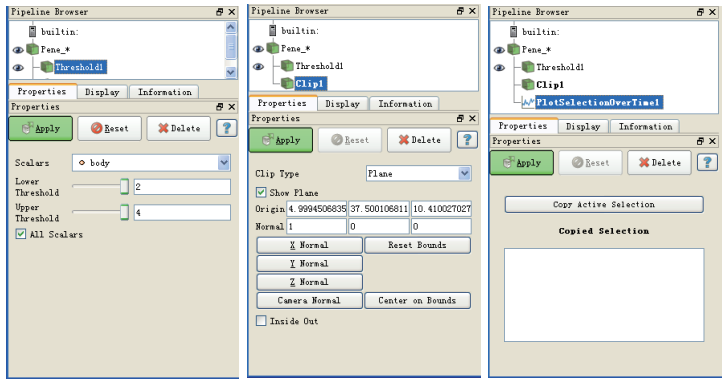
除了对导入的数据进行云图显示和动画观察，ParaView 还可以对数据对象进行数据

分析，该项功能集中在 Filters 菜单栏中。该菜单提供了多种过滤器算法，可以对原始数据进行多种分析和再加工，方便观察各种物理现象和提取指定点物理量的时间历程曲线等。因此，过滤器操作需要数据对象源 (一般为含有质点/单元的数据对象) ，通过过滤算法得到一个新的子数据对象，在可视化管道列表区中位于母数据对象下缩进显示。下面主要介绍 Threshold、Clip 和 Plot Selection Over Time 等 3 种常用的过滤算法，对于其他的过滤器操作，请参考 ParaView 帮助文档。

1. Threshold

以某物理量的阀值范围来过滤指定数据对象中的质点和/或单元，位于阀值范围内的质点和/或单元组成一个新的子数据对象。物理量的选择及其阀值范围的设定位于该数据对象 Properties 属性页的参数区，如图C.4 (a) 所示，各参数的意义如下：

- Scalars：通过其下拉菜单可选择变量，以此变量的值来做判据。
- Lower Threshold：变量值的下限。
- Upper Threshold：变量值的上限。



(a) Threshold　(b) Clip　(c) PlotSelectionOverTime

图 C.4　Filters 产生的数据对象属性页

因为不能同时显示同一数据对象的质点变量和单元变量云图，所以在图C.3中，显示了有限单元的等效塑性应变云图，但没有显示转化的质点云图。我们可以采用此功能将质点筛选出来形成一个新的数据对象并与母对象数据一起显示，则可以达到双重显示的目的。流程如下：

(1) 选择管道列表中的相应的数据对象，单击其眼形图标激活其所在的 3D View。

(2) 单击 Filters 菜单下 Common 中的 Threshold 选项或者单击工具栏的快捷图标，此时会在管道列表中新增一个对象名称为 Threshold1，如图C.5所示。

(3) 单击 Threshold 对象的属性页 Properties，此处变量选择 body，设置完成参数设置后，单击该属性页的 Apply 按钮完成过滤器操作。

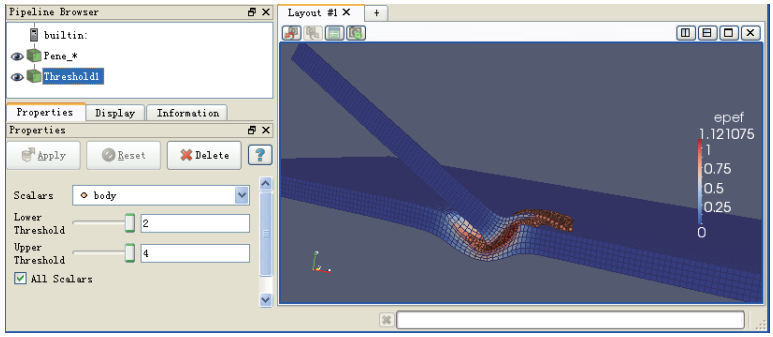


图 C.5 Threshold 使用演示

(4) 单击母数据对象的眼形图标，关闭其可见状态。此时，在当前的 3D View 中出现的子数据对象的物理图像，选择 Point Sprite 显示方式，在该对象属性页 Display 中选择渲染方式为 Sphere (Texture) ，size 设为 5。

(5) 对数据对象 Threshold1 进行相应的操作以观察各物理量的云图。

(6) 同时激活母数据对象和子数据对象，则如图C.5所示。

2. Clip

通过定义的几何图形或体，对数据对象进行裁减，获得剩余部分的物理模型，形成一个新的子数据对象。裁减所采用的几何图形及其相应的参数在该对象属性的 Properties 参数区中设置，如图C.4 (b) 所示。各参数的意义如下：

- Clip Type：几何体类型选择，支持四种类型：Plane、Box、Sphere 和 Scalar，常用类型为 Plane。
- Show Plane：控制几何体是否在视图中可见，若在视图中可见则可通过鼠标拖动该几何体完成相应的参数定义。
- 具体参数设置区：该部分会依据 Clip Type 选择的类型做相应的变化。以 Plane 类型为例，Origin 用来设置中心点坐标，Normal 用来设置平面法线方向。也可通过在按钮区单击相应按钮来定义法线方向和中心点。
- Inside Out：选择保留的区域。

在此我们对 Pene_* 数据对象进行平面切割，以演示该过滤器的使用方法。流程如下：

(1) 选择管道列表中相应的数据对象，单击其眼形图标激活其所在的 3D View。

(2) 单击 Filters 菜单下 Common 中的 clip 选项或者单击工具栏的快捷图标，如图C.6红圈所示，此时会在管道列表中新增一个数据对象名称为 Clip1。

(3) 单击数据对象 Clip1 的属性页 Properties，进行裁剪几何体的定义。

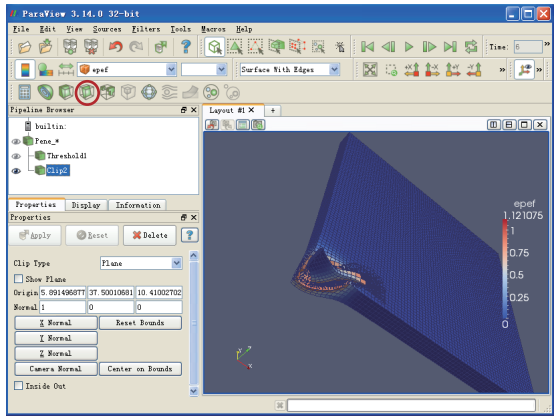


图 C.6　Clip 使用演示

(4) 完成属性页 Properties 参数设置后，单击该属性页的 Apply 按钮完成过滤器操作。

(5) 单击母数据对象的眼形图标，关闭其可见状态。此时，在当前的 3D View 中出现子数据对象的物理图像。

(6) 对数据对象 Clip 进行相应的操作以观察各物理量在剖面上的分布，如图C.6所示。

(7) 可继续调整相应的参数设置，单击 Apply 重新获得物理图像。

3. PlotSelectionOverTime1

对选定的质点/结点或者单元，从数据对象中提取数据，绘制物理量的时间历程曲线。

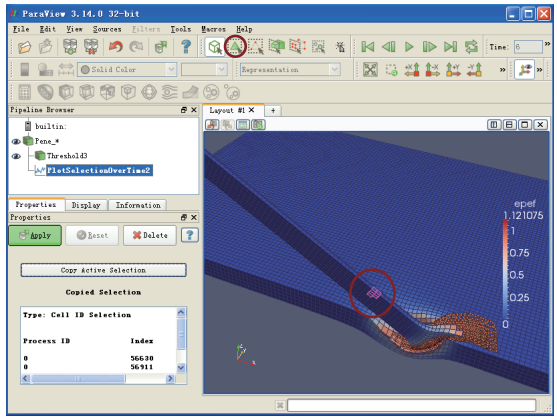


图 C.7　PlotSelectionOverTime 使用演示

在此我们以图C.3所示的数据对象为例，演示该过滤器的使用流程。首先选取要观察的点或单元，选取方式参见工具栏 Selection Controls 的图标，依据不同的需求选择不同的工具。流程如下：

(1) 单击管道列表中数据对象，激活该数据对象所在的 3D View。

(2) 单击工具栏中选择单元选择工具，如图C.7工具栏区红色圈中所示。选中的质点/结点或单元会用不同的颜色与周围区分，如图C.7所示。

(3) 单击菜单栏中 Filters - Data Analysis - Plot Selection Over Time，此时会在管道列表中新增一个对象名称为 PlotSelectionOverTime，同时列出其属性页，如图C.7所示。

(4) 在属性页中单击“Copy Active Selection”，然后单击“Apply”，完成该过滤器操作，生成的曲线如图C.8所示，此时可在 Display 属性页，选择要显示的单元和变量。

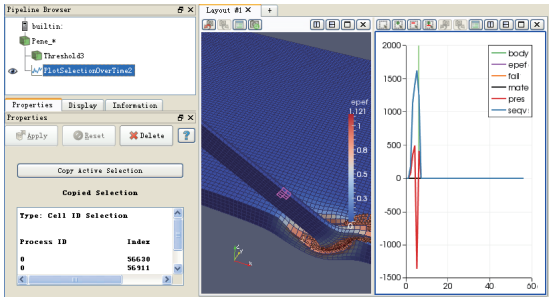


图 C.8 PlotSelectionOverTime 提取数据结果

C.2 数据格式

ParaView 可读取的数据文件格式有多种，用户也可在 ParaView 框架内加入特定的读入插件来读取定制的数据文件格式。ParaView 基于 VTK[120, 121] 开发，因此可以读取 VTK 数据文件格式。在 VTK 数据文件中，支持的数据结构有结构化点 (structured point)、结构化网格 (structured grid) 、非结构化网格 (unstructured grid) 等。早期的 VTK 数据文件是不区分数据结构的，统一后缀为.vtk，并且数据文件组织格式简单明了，方便人工手写也可编程输出。为了组织和管理更加复杂的数据结构，现在 VTK 数据文件采用了 XML 语言描述，并且根据不同的数据结构，采用不同的后缀加以区分，如结构化网格数据文件的后缀为.vts，非结构化网格数据文件则为.vtu。本附录仅介绍针对非结构化网格数据结构的.vtk 文件和.vtu 文件。

C.2.1 .vtk 文件

1. 数据文件组织格式总览

该数据文件组织格式如图C.9所示，包括 5 部分内容：

第 1 部分是数据文件的第 1 行，用于指定文件的版本号，即 # vtk DataFile Version x.x。其中，x.x 是发布的 VTK 版本号。

第 2 部分是数据文件的第 2 行，用于对本数据文件的内容进行简单描述，不超过 256 个字符。

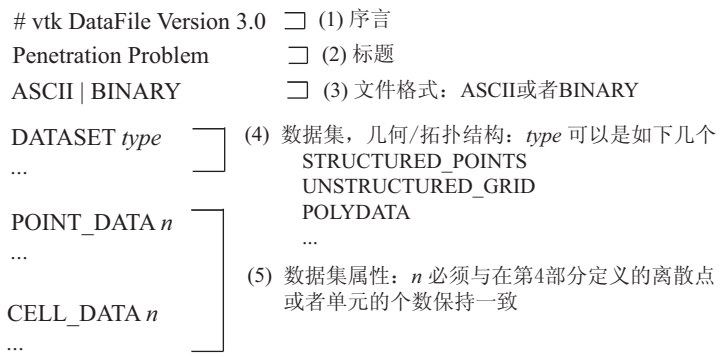


图 C.9　后缀为.vtk 的 VTK 文件格式

第 3 部分是数据文件的第 3 行，说明本数据文件存储的除关键字外的数据的格式，支持 ASCII 码和二进制 (binary) 码两种，分别采用关键字 ASCII 或 BINARY 指定。通常指定为 ASCII，便于跨系统使用。

第 4 部分是该数据文件描述的物理模型，包括结点信息和单元信息，称为数据集 (Dataset) 。

第 5 部分是数据集的属性 (Dataset Attribute) ，即结点或者单元的变量值。

在 VTK 文件格式中有几点需要注意：

- 某些关键字下的具体数值需要指定其数据类型 (dataType) ，目前 VTK 支持的数据类型关键字有 bit，unsigned_char，char，unsigned_short，short，unsigned_int，int，unsigned_long，long，float 和 double。
- VTK 中结点和单元按照其排列顺序自动编号，并且编号从 0 开始，因此第一个点的 ID 号是 0。
- 第 4 部分内容须位于第 5 部分之前。
- 关键字不区分大小写。

2. 数据集 (Dataset) 关键字

数据集用来描述物理模型，包括结点的坐标值，单元的拓扑结构及其类型。VTK 支持多种单元类型，图C.10给出了几种常用的单元类型及其在 VTK 数据文件中的索引号。

针对非结构化网格数据，数据集的关键字包括 4 个：UNSTRUCTURED_GRID、POINTS、CELLS 和 CELL_TYPES。使用形式如下：

- DATASET UNSTRUCTURED_GRID：声明该数据集为非结构化网格数据
- POINTS n dataType：声明结点总数 n 及其数据类型 dataType
  p_{0x} p_{0y} p_{0z}：0 号结点的坐标
  p_{1x} p_{1y} p_{1z}

...

$p_{(n-1)x}$ $p_{(n-2)y}$ $p_{(n-3)z}$：$n-1$ 号结点的坐标

- CELLS n size：声明单元总数 n 及其列表中的数据总数 size
  numPoints$_0$ i j k l ...：0 号单元的结点总数 numPoints$_0$ 及各结点的编号
  numPoints$_1$ i j k l ...
  numPoints$_2$ i j k l ...
  ...
  numPoints$_{n-1}$ i j k l ...

- CELL_TYPES n：声明在 CELLS 中定义的 n 个单元的类型
  type$_0$ ：0 号单元的类型 (见图C.10中各单元的索引号)
  type$_1$
  ...
  type$_{n-1}$

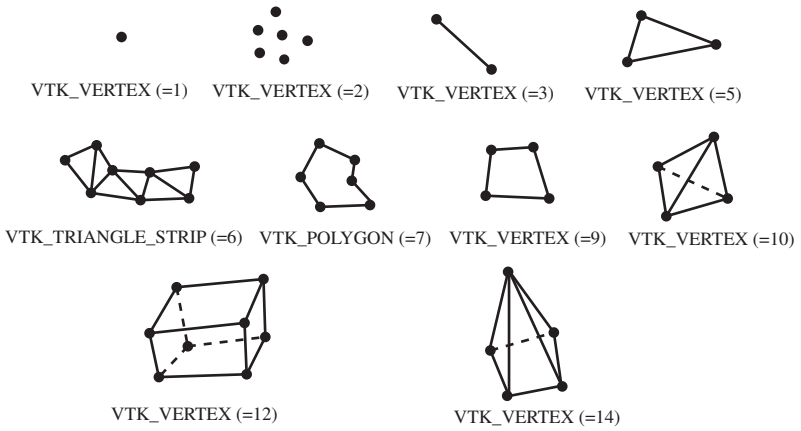


图 C.10 VTK 文件支持的线性单元类型

POINTS 关键字用于定义结点的位置，参数 n 表示结点总数，dataType 用于指定其后数据的类型，在此应为 double。此后各行为各个单元结点的坐标值，数据间采用空格分割。

CELLS 关键字用于定义单元的拓扑结构，可同时定义不同类型的单元。参数 n 表示单元总数；size 为单元拓扑结构列表中参数的个数，即各单元结点数与 n 的和。每一行为一个单元的拓扑结构，第一个数字表示该单元具有的总结点数，后面依次为该单元各结点的编号，并按照图C.10中单元结点序列排列。

CELL_TYPES 用于指定每个单元的具体类型，参数 n 表示定义的单元总数，此后各行数据表示各单元的类型，具体数值为图C.10中的单元索引号。

3. 数据集属性关键字

结点和单元的变量值称为数据集的属性，依据变量类型的不同，可采用不同的关键字。VTK 可以支持的变量类型包括：标量 (SCALARS，最多可由 4 个元素组成) 、矢量 (VECTORS) 、3 × 3 的张量 (TENSORS) 等。每个关键字的参数至少包括名称 (dataName) 、数据类型 (dataType) ，其中名称一般取为该变量的物理含义。下面依次介绍。

(1) 标量型变量关键字使用形式

SCALARS dataName dataType numComp

LOOKUP_TABLE default

s_0

s_1

...

s_{n-1}

其中数据名称 dataName 为字符串，数据类型 dataType 为 VTK 支持的数据类型关键字，numComp 用于指定该标量组的元素个数。此外，关键字 LOOKUP_TABLE 可为该标量数据指定图例，此处采用默认值。

(2) 矢量型变量关键字使用形式

VECTORS dataName dataType

$v_{0x}\ v_{0y}\ v_{0z}$

$v_{1x}\ v_{1y}\ v_{1z}$

...

$v_{(n-1)x}\ v_{(n-1)y}\ v_{(n-1)z}$

(3) 张量型变量关键字使用形式

TENSORS dataName dataType

$t^0_{00}\ t^0_{01}\ t^0_{02}$

$t^0_{10}\ t^0_{11}\ t^0_{12}$

$t^0_{20}\ t^0_{21}\ t^0_{22}$

$t^1_{00}\ t^1_{01}\ t^1_{02}$

$t^1_{10}\ t^1_{11}\ t^1_{12}$

$t^1_{20}\ t^1_{21}\ t^1_{22}$

...

$t^{n-1}_{00}\ t^{n-1}_{01}\ t^{n-1}_{02}$

$t^{n-1}_{10}\ t^{n-1}_{11}\ t^{n-1}_{12}$

$t^{n-1}_{20}\ t^{n-1}_{21}\ t^{n-1}_{22}$

4. 一个范例

图C.11中显示了有多个类型单元组成的一个物理模型，对应的输入文件如下：

```
#vtk DataFile Version 3.0
```

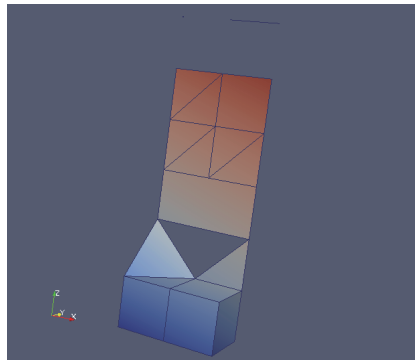


图 C.11　VTK 文件范例图示

```
Unstructured Grid Example
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 27 float
0 0 0   1 0 0   2 0 0   0 1 0   1 1 0   2 1 0
0 0 1   1 0 1   2 0 1   0 1 1   1 1 1   2 1 1
0 1 2   1 1 2   2 1 2   0 1 3   1 1 3   2 1 3
0 1 4   1 1 4   2 1 4   0 1 5   1 1 5   2 1 5
0 1 6   1 1 6   2 1 6

CELLS 11 60
8 0 1 4 3 6 7 10 9
8 1 2 5 4 7 8 11 10
4 6 10 9 12
4 5 11 10 14
6 15 16 17 14 13 12
6 18 15 19 16 20 17
4 22 23 20 19
3 21 22 18
3 22 19 18
2 26 25
1 24

CELL_TYPES 11
12
12
10
10
7
6
9
5
5
```

```
3
1

POINT_DATA 27
SCALARS scalars float 1

LOOKUP_TABLE default
0.0 1.0 2.0 3.0 4.0 5.0
6.0 7.0 8.0 9.0 10.0 11.0
12.0 13.0 14.0 15.0 16.0 17.0
18.0 19.0 20.0 21.0 22.0 23.0
24.0 25.0 26.0

VECTORS vectors float
1 0 0   1 1 0   0 2 0   1 0 0   1 1 0   0 2 0
1 0 0   1 1 0   0 2 0   1 0 0   1 1 0   0 2 0
0 0 1   0 0 1   0 0 1   0 0 1   0 0 1   0 0 1
0 0 1   0 0 1   0 0 1   0 0 1   0 0 1   0 0 1
0 0 1   0 0 1   0 0 1
```

C.2.2　.vtu 文件

vtu 文件描述的数据结构同 vtk 文件是相同的，但采用标准的 XML 语言描述。

1. XML 语言规范

XML 是 eXtensible Markup Language 的简写，意为可扩展的标记语言，其存储形式为树形结构，适合表示复杂的数据结构。XML 文件有着严格的书写规范，下面简要介绍其书写的方式。

每个 XML 文件都有 XML 序言开始，用来声明版本号，例如

```
<?xml version="1.0" ?>
```

XML 文件通常是由一个根元素 (root element) 和多个一级元素 (element) 组合而成。元素由起始标签开始 (start-tag，如 <Heading>) 开始，由结束标签 (end-tag，如 </Heading>) 结束，它们之间的部分为元素描述的具体内容 (content) ，例如

```
<Heading>Taylor bar impact</Heading>
```

就是一个元素。一个元素也可仅由空元素标签组成，如 <line_break />。元素的内容也可以包含其他元素，称为子元素。通常，一个元素的内容由元素的属性 (attributes) 及其属性值和子元素构成。属性的值 (数值或字符串) 用引号括起来，用等号“=”与属性相连。同一个元素下的属性与属性之间用空格分割，排序不分先后，例如

```
<Piece NumberOfPoints="#" NumberOfCells="#" />
```

一个 XML 文件只有一个根元素，其他所有元素可作为根元素的子元素。一个元素可以包含多个子元素。XML 文件中可以包含注释，以增加文件的可读性。注释以<!-- 开头，以 --> 结束，例如

```
<!-- catalog last updated 2014-1-19 -->
```

2. VTU 文件格式总览

```
<?xml version="1.0" ?>
<VTKFile type="UnstructureGrid" version="0.1"
   byte_order="LittleEndian" compressor="vtkZLibDataCompressor">
   <UnstructureGrid>
        <Piece NumberOfPoints="#" NumberOfCells="#">
            <PointData> ... </PointData>
            <CellData> ...  </CellData>
            <Points> ... </Points>
            <Cells> ... </Cells>
        </Piece>
   </UnstructureGrid>
</VTKFile>
```

本例即为 vtu 文件的 XML 格式，包括根元素 VTKFile，一级元素 UnstructureGrid 和多个子元素 Piece、Pointdata、CellData、Points、Cells。下面分别介绍。

3. 根元素 VTKFile

根元素关键字为 VTKFile，具有 4 个属性关键字，如下：

- type：用于指定文件描述的数据类型，此处取值为 UnstructureGrid，即非结构化网格数据。
- version：用于指定文件的版本号。
- byte_order：用于指定二进制文件中数据存储的字节顺序，可取为 BigEndian (大端序) 或 LittleEndian (小端序)。在二进制文件中，多字节数据被存储为连续的字节序列。如果最低有效字节在最高有效字节的前面，称为小端序，反之称为大端序。例如，十六进制数据 0x01234567 用 4 个连续字节存储。小端序的 4 个连续字节依次存储 67、45、23 和 01，而大端序的 4 个连续字节依次存储 01、23、45 和 67。x86 系列 CPU 采用 little endian 方式存储数据。
- compressor：该属性为可选项，由 VTK 库文件的输出类指定。

4. 一级元素 UnstructuredGrid

一级元素关键字为根元素的 type 属性值。由于本 vtu 文件为非结构化网格数据，type 应为 UnstructuredGrid，所以一级元素关键字为 UnstructuredGrid。该元素包含子元素 Piece，用于定义数据集及其属性。子元素 Piece 可在一级元素内重复出现。

5. 子元素 Piece

子元素关键字 Piece 用于定义一个具体的数据集，即 VTK 文件格式的第 4 和第 5 部分。该元素关键字具有两个属性，NumberOfPoints 和 NumberOfCells，其取值分别为单元结点数和单元总数。该元素关键具有 4 个子元素：Points、Cells、PointData 和 CellData。其中前两个分别定义单元结点的坐标和单元的拓扑结构及类型，即数据集；后两个定义结点和单元的变量，即数据集的属性。

6. 描述数据集的元素关键字

子元素关键字 Points，用于定义每个结点的坐标值，包含 1 个子元素 DataArray。示例如下：

```
<Points>
    <DataArray NumberOfComponents="3" ... />
</Points>
```

子元素关键字 Cells 用于定义单元的拓扑结构和单元类型，包含 3 个 DataArray 子元素。第一个 DataArray 定义单元的拓扑结构，第二个 DataArray 定义每个单元的结点列表在第一个 DataArray 中的终止点位置，第三个 DataArray 定义单元的类型。示例如下：

```
<Cells>
    <DataArray type="Int32" Name="connectivity" ... />
    <DataArray type="Int32" Name="offsets" ... />
    <DataArray type="UInt8" Name="types" ... />
</Cells>
```

7. 描述数据集属性的元素关键字

子元素关键字 PointData 用于定义单元结点的变量值，包含 1 个子元素 DataArray。子元素用于定义所有结点的某个变量值，可重复出现以定义不同的变量值。示例如下：

```
<PointData>
    <DataArray Name="Velocity" .... />
    <DataArray Name="Temperature" .... />
    <DataArray Name="Pressure" .... />
</PointData>
```

子元素关键字 CellData 用于定义单元的变量值，形式同 PointData，不再赘述。

8. 子元素 DataArray、AppendedData

从上面可知，数据集及其属性的具体值均是通过 DataArray 元素定义的。DataArray 元素用于定义同一类型的数据，依据不同的需求，具有不同的属性和形式。示例如下：

```
<DataArray type="Float32" Name="vectors" NumberOfComponents="3"
           format="appended"  offset="0" />
```

```
<DataArray type="Float32" Name="scalars" format="binary">
          bAAAAAAAAAAAAIA/AAAAQAAAQEAAAIBA ... </DataArray>
<DataArray type="Int32" Name="offsets" format="ascii">
          10 20 30 ... </DataArray>
```

DataArray 元素具有如下属性：

- type：指定数组的数据类型，可选范围为 Int8、UInt8、Int16、UInt16、Int32、UInt32、Int64、UInt64、Float32 和 Float64。
- Name：数组的名称，通常取为变量的名称。
- NumberOfComponents：每个变量的元素个数。不论是标量还是矢量，DataArray 中的数据均按照一维数组的形式连续存储，因此需要指定变量包含的数据个数。
- format：数据在文件存储的形式，可选“ascii”，“binary”，“appended”。其中，ascii 表示数据采用 ASCII 形式存在 DataArray 数组中，用空格分割数据；binary 表示数据二进制存储并采用 base64 编码形式表示；appended 表示数据存储在 AppendedData 元素内而非 DataArray 元素中。
- offset：当选择 appended 格式存储数据时，所有数据均连续存储在 AppendedData 元素中。属性 offset 的值定义了数组在 AppendedData 元素内数据中的开始位置。
- RangeMin/RangeMax：成对出现，为变量的最小值与最大值。

AppendedData 元素用于连续存储所有采用 format=“appended” 的 DataArray 中的数据，并采用 base64 编码形式存储。示例如下：

```
<VTKFile>
    ...
    <AppendedData encoding="base64">
        _QMwEAAAAAA...
    </AppendedData>
</VTKFile>
```

9. 一个范例

采用 vtu 格式，可将上节中的 vtk 范例文件改写为：

```
<?xml version="1.0"?>
<VTKFile type="UnstructuredGrid" version="0.1"
               byte_order="LittleEndian" >
  <UnstructuredGrid>
    <Piece NumberOfPoints="27" NumberOfCells="11" >
      <Points>
        <DataArray type="Float32" Name="Points" NumberOfComponents="3"
                   format="ascii" >
```

```
      0 0 0   1 0 0   2 0 0   0 1 0   1 1 0   2 1 0
      0 0 1   1 0 1   2 0 1   0 1 1   1 1 1   2 1 1
      0 1 2   1 1 2   2 1 2   0 1 3   1 1 3   2 1 3
      0 1 4   1 1 4   2 1 4   0 1 5   1 1 5   2 1 5
      0 1 6   1 1 6   2 1 6
  </DataArray>
</Points>

<PointData Vectors="vector">
  <DataArray type="Float32" Name="scalars" NumberOfComponents="1"
             format="ascii" RangeMin="0" RangeMax="26.0" >
      0.0 1.0 2.0 3.0 4.0 5.0
      6.0 7.0 8.0 9.0 10.0 11.0
      12.0 13.0 14.0 15.0 16.0 17.0
      18.0 19.0 20.0 21.0 22.0 23.0
      24.0 25.0 26.0
  </DataArray>

  <DataArray type="Float32" Name="vector" NumberOfComponents="3"
             format="ascii" >
      1 0 0   1 1 0   0 2 0   1 0 0   1 1 0   0 2 0
      1 0 0   1 1 0   0 2 0   1 0 0   1 1 0   0 2 0
      0 0 1   0 0 1   0 0 1   0 0 1   0 0 1   0 0 1
      0 0 1   0 0 1   0 0 1   0 0 1   0 0 1   0 0 1
      0 0 1   0 0 1   0 0 1
  </DataArray>
</PointData>

<Cells>
  <DataArray type="Int32" Name="connectivity" format="ascii" >
      0 1 4 3 6 7 10 9
      1 2 5 4 7 8 11 10
      6 10 9 12
      5 11 10 14
      15 16 17 14 13 12
      18 15 19 16 20 17
      22 23 20 19
      21 22 18
      22 19 18
      26 25
      24
  </DataArray>

  <DataArray type="Int32" Name="offsets" format="ascii" >
       8 16 20 24 30 36 40 43 46 48 49
  </DataArray>
```

```
        <DataArray type="UInt8" Name="types" format="ascii" >
            12 12 10 10 7 6 9 5 5 3 1
        </DataArray>
      </Cells>
    </Piece>
  </UnstructuredGrid>
</VTKFile>
```

C.2.3 VTK 的输出类 (Writer)

VTK 提供了输出 VTK 格式数据文件的输出类，用户可以调用相应的输出类输出后处理文件。例如，在输出非结构化网格数据时，可以先创建 vtkUnstructuredGridWrite 类型对象 writer，然后依次调用成员函数 SetInput、SetFileName 和 Write 指定要输出的 vtkUnstructuredGrid 对象、指定输出数据文件名和执行输出操作。

vtk 和 vtu 文件需要调用的两个输出类分别为 vtkUnstructuredGridWrite 和 vtkXMLUnstructuredGridWrite。这两个类都需要 vtkUnstructuredGrid 类声明的对象作为输入参数。为此，需要将要输出的数据依照 Point、Cell、PointData 和 CellData 的形式组装到 vtkUnstructured 声明的对象中。vtkUnstructuredGrid 的数据结构形式如图C.12所示。

下面给出采用 C++ 语言编写的一段 VTK 后处理输出代码：

```
#include <vtkUnstructuredGrid.h>
#include <vtkXMLUnstructuredGridWriter.h>
#include <vtkIdList.h>
#include <vtkFloatArray.h>
#include <vtkPointData.h>
#include <vtkCellData.h>
```

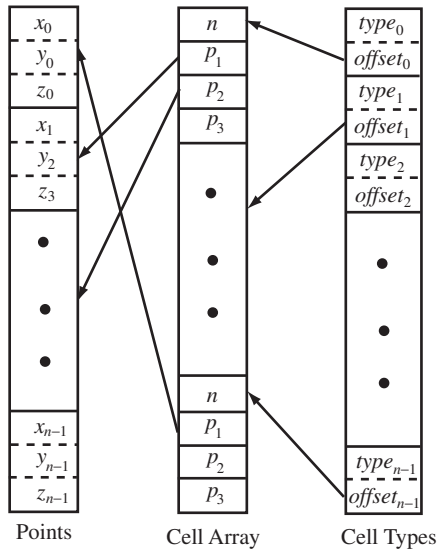


图 C.12 vtkUnstructuredGrid 数据结构图示

```
#include "vtkDoubleArray.h"
#include "vtkSmartPointer.h"

vtkPoints *newPoints = vtkPoints::New();  // 存储结点坐标
newPoints->Allocate(nb_point);    // 依据总结点数，分配内存
vtkCellArray *newCells = vtkCellArray::New();   // 存储单元的拓扑结构
newCells->Allocate(nb_ele);   // 依据总的单元数，分配内存

//Velocity vectors 定义vector属性的变量
vtkDoubleArray* vectors = vtkDoubleArray::New();
vectors->SetNumberOfComponents(3);
vectors->SetNumberOfTuples(nb_point);

// 定义用来指定各个单元类型的数组
int* types = new int[nb_ele];

// 声明用来存储PointData和CellData的指针型数组
double **H5ptOption = new double*[aninum];
double **EH5ptOption = new double*[aninum];

// 依据要输出的物理量总数，给两个数组分配内存
for(unsigned char i=0;i<aninum;i++)
{
    H5ptOption[i]=new double[nb_point];
    EH5ptOption[i]= new double[nb_ele];
}

// 依据具体情况，填充 newPoints、newCells、vectors、types、
// H5ptOption、EH5ptOption数组
....

// ---------- 组装vtkUnstructured数据 ----------------------
// 声明vtkUnstructuredGrid对象
vtkUnstructuredGrid *ugrid = vtkUnstructuredGrid::New();

// 在对象ugrid中填充Cells数据
ugrid->SetCells(types,newCells);

// 在对象ugrid中填充Points数据
ugrid->SetPoints(newPoints);

// 在对象ugrid中填充PointData
if(velocity_output) {
    vectors->SetName("velocity");
    ugrid->GetPointData()->SetVectors(vectors);
}
```

```
// 将存储在H5ptOption、EH5ptOption中的PointData和CellData
// 填充到ugrid对象中
for(unsigned char i=0;i<aninum;i++)
{
    vtkDoubleArray *value = vtkDoubleArray::New();
    value->SetName(MPM::OutputName[animoption[i]]);
    for (int j=0; j<nb_point; j++)
       value->InsertNextValue(H5ptOption[i][j]);
    ugrid->GetPointData()->AddArray(value);
    value->Delete();

    vtkDoubleArray *values = vtkDoubleArray::New();
    values->SetName(MPM::OutputName[animoption[i]]);
    for (unsigned int j=0; j<nb_ele; j++)
       values->InsertNextValue(EH5ptOption[i][j]);
    ugrid->GetCellData()->AddArray(values);
    values->Delete();
}

// 删除指针型数组
delete [] animoption;
for(unsigned char i=0;i<aninum;i++)
{
    delete  [] H5ptOption[i];
    delete  [] EH5ptOption[i];
}
delete [] H5ptOption;
delete [] EH5ptOption;
delete [] types;

//-------------- 调用VTK的输出类写出 vtu文件 --------------

const string suffix = ".vtu";
string fileName =  JobTitle + "_" + InttoString(iStep) + suffix;
// 声明写出类对象
vtkXMLUnstructuredGridWriter *Writer =
           vtkXMLUnstructuredGridWriter::New();
Writer->SetInput(ugrid);   // 将ugrid数据导入该类中
Writer->SetFileName(fileName.c_str());  // 给数据文件命名
Writer->Write();    // 写出vtu文件

// 删除指针型对象和数组
ugrid->Delete();
newPoints->Delete();
newCells->Delete();
vectors->Delete();
dataWriter->Delete();
```