**Peter A. Jacobs and Rowan J. Gollan and Ingo Jahn**

# Eilmer4.0 流动仿真程序

## ----------几何工具包指南

（可用于构建流动路径）

## 2020 年 4 月 20 日更新

译者：卢顺、齐建荟

Technical Report 2017/25

School of Mechanical & Mining Engineering

The University of Queensland

# Geometry user guide Translation

**摘要：**

在 Eilmer4 可压缩流动模拟程序中，几何工具包支持几何元素的构建，如表面和三维体。工具包的元素可以用来建立气体流动和固体域的描述，它们可以被离散，然后传递到流动求解器。流域的描述被构造为一个 Lua 脚本，它定义了包围域元素的位置和范围。对于二维流域，您将使用边来限定(x,y)平面上的小块;对于三维流域，您将在边处满足定义(x,y,z)空间中的参数体来定义曲面。

流动求解器计划将气体流动和固体域指定为有限体积单元的网格，因此您将需要离散二维块或三维体。StructuredGrid 类可以通过在块和体积内插值点来构建这些网格。还有一个 UnstructuredGrid 类，当结构化网格难以生成时可以使用它。

Eilmer4 的源代码可在 https://bitbucket.org/cfcfd/dgd/找到，与其相关的可压缩仿真代码收录在 http://cfcfd.mechmining.uq.edu.au/网站上。

本用户指南的英文原版以及几何、气体等配套文件可在以下网站下载：https://gdtk.uqcloud.net/docs/eilmer/user-guide/。

**致谢：**

非常感谢 PAJ 在牛津大学 Special Studies Program 时准备了这份几何文件。

# 目录

# 1. 介绍

几何工具包支持几何元素的构建，如 Eilmer 流动模拟程序[1]中的表面块和的三维体。流动求解器希望将气体流动和固体域指定为有限体积单元网格。您可以在您最喜欢的网格生成程序中准备这些网格,然后将它们导入到您的仿真中，或者可能使用在报告中描述的元素来准备需要描述的域,然后使用一个包含几何工具的离散化网格生成器。这里所描述的过程对于相对简单的区域来说是最方便的，但是，只要付出足够的努力，也可以应用于任意复杂的情况。它们的优点是您的仿真描述是完全自包含的，您不需要依赖外部程序来进行模拟。这个报告是整体模拟程序[2]的一个工具包。

对于结构化网格，网格生成器的顶层几何元素是二维流动的块和三维流动的参数体。这些空间区域可以被一组参数坐标 0≤r<1, 0≤s<1(二维)以及第三个参数 0≤t<1(三维)所遍历。这些块或体可以作为 VTK 结构网格导入，也可以从低维度的几何实体(如路径和点)构建为边界。

对于非结构化网格[1]，在二维或作为一个表面网格在三维系统中，将边界的形式提供给网格生成器作为一组离散的边界。也可以从结构化网格构建非结构化网格，或者导入 VTK 或 SU2 格式的非结构化网格。

## 1.1 一些建议

在描述用于构建流域几何元素的细节之前，我们想提供一些关于构建描述过程的建议。这个过程是对几何体构建程序进行编程，以构建流域的编码进行描述。考虑到这一点，我们建议采取以下步骤：

1. 首先在纸上画出流域的草图，标记关键特性。
2. 从小处着手，构建一个脚本来描述整个域中的一些非常简单的元素。
3. 使用 e4shared 选项(--prep 或--custom-post)处理此脚本，生成渲染图或网格
4. 查看该图是否是您想要的，然后进行必要的调试。
5. 以小步骤继续完成流域的描述。

[1] 用于生成非结构化网格的函数正在进行中。

我们相信这个程序将会带来一个远比一次性编码您的整个描述域更令人满意的体验。您可能是幸运的，但发生的错误可能会改变您的运气。

# 2. 几何元素

使用几何工具包中的元素的最终目标是构造一个可以传递给流动求解器的流域表示。对于二维流动模拟，这个区域将被定义为 x,y 平面上的一个或多个块。在程序代码中，这些块被表示为表面对象，这些对象是拓扑四边形的，被标记为北、东、南和西。这些块与整体结构相适应，这意味着主边界将呈现一个或多个包含气体流动的形状。对于一个三维流动模拟，气体流动域将使用相适应的三维体对象进行描述。这些体和表面是由低维的几何物体，特别是点和路径构成。

## 2.1 点

最基本的一类几何对象是 Vector3，它表示三维空间中的一个点，具有常规的几何向量特征。创建和操作这些对象的代码在底层 Vector3 类中的 D 语言模块。这段 D 代码被包装成可以从 Lua 脚本访问的形式。

在 Lua 脚本中，您可以构造一个新的 Vector3 对象:

```
Vector3:new{x=0.1, y=0.2, z=0.3}
```

当构建二维区域的模型时，您可以省略 z 分量，它将默认为零。注意，我们使用的是 Lua 编程[3]中描述的面向对象协议。在我们的程序设置的 Lua 环境中，Vector3 是一个表，而 new 是该表中的一个函数。在这个特殊的例子中，我们将把 new 函数当作 Vector3 对象的构造函数。在单词 new 之前使用冒号而不是圆点，这是在告诉解释器，我们希望在传递给被调用函数的参数中包含对象本身。最后，请注意使用大括号构造函数的参数表。在 new 函数的绑定代码中，我们希望在堆栈上传递的所有参数都包含在一个表中。在输入脚本中构造其它对象时，我们通常选择一种将所有元素作为命名属性放在一个表中的表示法。我们认为这有利于将参数的顺序变得不重要。它还使您的脚本更加自文档化，但代价是更加繁杂冗长。

可以在几何元素中“获取”和“设置”属性值。例如，要创建一个节点，提取该节点的 x 分量，更改 y 分量，或者使用这些分量构造一个新点，以下脚本可供您参考。

```
1 -- example-1.lua
2
3 a = Vector3:new{x=0.5, y=0.8}
4 x_value = a.x
5 a.y = 0.4
6 b = Vector3:new{x=a.x+0.4, y=a.y+0.2}
7 print("a=", a, "b=", b)
8 print("x_value=", x_value)
9
10 dofile("sketch-example-1.lua")
```

在第 3 行,我们构造了一个点,并将其绑定到名称 a。第 4 和 5 行显示的示例可以获取和设置组件，第 6 行构造一个涉及点 a 表达式的新点。第 10 行调用另一个文件让 SVG 显示这些点。我们将在后面的部分讨论该文件的内容。目前来看，它有些复杂，但并不重要。

通过 Eilmer4 程序的自定义后处理模式运行此脚本的记录如下:

```
1 $ e4shared --custom-post --script-file="example-1.lua"
2 Eilmer4 compressible-flow simulation code.
3 Revision: 0c2021bb8eb9 713 default tip
4 Begin custom post-processing using user-supplied script.
5 Start lua connection.
6 a= Vector3([0.5, 0.4, 0]) b= Vector3([0.9, 0.6, 0])
7 x_value= 0.5
8 Done custom postprocessing.
```

生成的点在图 2.1 中以图形方式显示。请注意 print 函数需要显示的内容在文本的第 6 行和第 7 行。显示的 Vector3 对象的格式是底层 D-language 模块格式，而不是输入脚本的第 3 行使用的 Lua 构造格式。在内部，坐标存储在固定大小的数组中，因此输出显示格式为方括号分隔的坐标值列表。

如果您查看 dgd/src/geom/geom.d 文件，您会看到 Vector3 对象支持通常的加法、减法等运算。此外，您可以从另一个对象构造一个 Vector3 对象，并且有少量的内置转换可能对以编程方式构造的流域有用。例如，要在(y,z)平面上通过 x=0.5 创建一个点及其镜像，可以使用:

```
1 -- example-2.lua
2
3 a = Vector3:new{x=0.2, y=0.5}
4 b = Vector3:new{a}
5
6 mi_point = Vector3:new{x=0.5} -- mirror-image plane through this point
```

```
7 mi_normal = Vector3:new{x=1.0} -- normal of the mirror-image plane
8 b:mirrorImage(mi_point, mi_normal)
9
10 print("a=", a, "b=", b)
11 print("abs(b)=", b:abs())
12
13 c = a + b
14 print("c=", c)
```

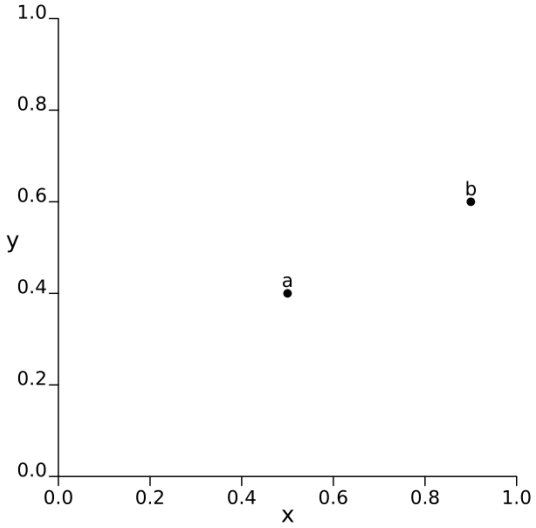



图 2.1：定义为 Vector3 对象并呈现给 SVG 的几个点。

相应的输出显示为：

```
1 $ e4shared --custom-post --script-file="example-2.lua"
2 Eilmer4 compressible-flow simulation code.
3 Revision: 0c2021bb8eb9 713 default tip
4 Begin custom post-processing using user-supplied script.
5 Start lua connection.
6 a= Vector3([0.2, 0.5, 0]) b= Vector3([0.8, 0.5, 1.11022e-16])
7 abs(b)= 0.94339811320566
8 c= Vector3([1, 1, 1.11022e-16])
9 Done custom postprocessing.
```

在示例脚本的第 8 行，mirrorImage 方法将调用镜像-图像平面的定义点作为未命名但有序的参数进行传递。这些参数由括号而不是大括号分隔，就像在构建表时使用的那样。还请注意，在脚本的第 11 行中，对 abs 的方法调用使用了一对空括号，使调用方法没有歧义。最后，在终端显示的第 6 行，注意 b 点的 z 分量中出现的舍入误差。这个误差被传递给了 c 点。

在表 2.1 中给出了为 Lua 环境提供的操作符和方法列表。

表 2.1:Vector3 对象的操作符和方法。在表中，Vector3 量表示为$\vec{a}$。标量是双精度(或

64 位)浮点值。在全局名称空间中，有些方法也以未绑定函数的形式出现。

| Lua 表达形式 | 表示结果 |
|---|---|
| -$\vec{a}$ | 带有负号的 Vector3 对象。$\vec{a}$不变。 |
| $\vec{a}+\vec{b}$ | 带有求和分量的 Vector3 对象 |
| $\vec{a}-\vec{b}$ | 带有减法分量的 Vector3 对象 |
| $\vec{a}*s$ | 带有 s 比例缩放分量的 Vector3 对象 |
| $s*\vec{b}$ | 带有 s 比例缩放分量的 Vector3 对象 |
| $\vec{a}/s$ | 带有除以 s 分量的 Vector3 对象 |
| $\vec{a}$：$normalize()$ | 使$\vec{a}$变为单位向量，并改变了$\vec{a}$的值 |
| $\vec{a}$：$dot(\vec{b})$ | 两个向量的点积 |
| $\vec{a}$：abs() | 向量的标量大小，不会改变$\vec{a}$的值 |
| $\vec{a}$：unit() | $\vec{a}$方向上单位大小的 Vector3 对象，不会改变$\vec{a}$的值 |
| $\vec{a}$：mirrorImage($\vec{p}$，$\vec{n}$) | $\vec{a}$向量在平面上通过$\vec{p}$与法向$\vec{n}$的镜像，注意改变了$\vec{a}$的值 |
| $\vec{a}$：rotateAboutZAxis(θ) | 将$\vec{a}$向量绕着 z 轴旋转θ角度，注意改变了$\vec{a}$的值 |
| $dot(\vec{a}，\vec{b})$ | 两个向量的点积，如上所述 |
| vabs($\vec{a}$) | 向量的标量大小，如上所述 |
| unit($\vec{a}$) | 方向为$\vec{a}$的单位大小 Vector3 对象，$\vec{a}$值不变 |
| $cross(\vec{a}，\vec{b})$ | 向量的乘法$\vec{a}\times\vec{b}$ |
| quadProperties{p0=$\vec{p}_0$，p1=$\vec{p}_1$，p2=$\vec{p}_2$，p3=$\vec{p}_3$} | 返回表 t，包含命名元素 centroid,n, t1, t2 以及 area |
| hexCellProperties{p0=$\vec{p}_0$，p1=$\vec{p}_1$， | 返回表 t，包含命名元素 centroid， |

| | |
|---|---|
| p2=$\vec{p}_2$，p3=$\vec{p}_3$，p4=$\vec{p}_4$，p5=$\vec{p}_5$<br>p6=$\vec{p}_6$，p7=$\vec{p}_7$} | volume, iLen, jLen 以及 kLen |

## 2.2 路径

下一层维度是 Path 类。路径对象是空间中的参数曲线，通过单个参数 0≤t <1 可以指定其上的点。Path 是 D 语言域中的一个基类，有许多派生类型的路径可用。对参数值 t 的路径求值会得到相应的 Vector3 结果。路径也可以作为字符串进行复制或打印。

### 2.2.1 Vector3 对象或 Lua 表

在接下来的章节中，将路径面和体的构造函数接受它们相应参数的 Points。如果您碰巧不需要 Vector3 对象的全部功能，您可以提供简单的带有命名条目的 Lua 表。例如:

a = {x=0.2, y=0.3}

现在可以在任何需要 Point 的地方提供绑定到 a 的 Lua 表。

### 2.2.2 简单路径

有几个简单的路径对象可以由点构造。构造函数通常接受它们在单个表中的参数，该表中的项目已被命名，如下所示:

• Line:new{p0=$\vec{a}$, p1=$\vec{b}$}：在$\vec{a}$和$\vec{b}$之间的一条直线；

• Arc:new{ p0=$\vec{a}$, p1=$\vec{b}$，centre=$\vec{c}$}：绕圆心$\vec{c}$，从$\vec{a}$到$\vec{b}$的圆弧。注意不要试图做出包含角为 180 度或更大的圆弧。对于这种情况，可以创建两个圆弧并作为一条折线路径进行连接；

• Arc3:new{ p0=$\vec{a}$, pmid=$\vec{b}$，centre=$\vec{c}$}：从$\vec{a}$到$\vec{b}$到$\vec{c}$的圆弧，三个点都在圆弧上；

• Bezier:new{points={$\vec{b}_0$，$\vec{b}_1$， ...，$\vec{b}_n$}}:从$\vec{b}_0$到$\vec{b}_n$的 n-1 阶 Bezier 曲线。您可以为曲线指定很多点，然而，对于三阶曲线来说，4 个点是非常常用的。贝塞尔曲线有一个非常方便的特点：您可以直接控制端点和这些端点上的切线。注意，点的表被标记并包含在传递给构造函数的单个表中。单个点没有被标记，尽管显示的下标从 0 开始(与核心 D 语言代码中的表示形式保持一致)，但 Lua 表中项目的默认编号从 1 开始。我们在这里回避了通过提供点表的文字构造问题，但是，有时需要在索引中考虑这个差异。

以下是构建这些简单路径的一些示例:

```
1 -- path-example-1.lua
2
3 -- An arc with a specified pair of end-points and a centre.
4 c = Vector3:new{x=0.4, y=0.1}
5 radius = 0.2
6 a0 = Vector3:new{x=c.x-radius, y=c.y}
7 a1 = Vector3:new{x=c.x, y=c.y+radius}
8 my_arc = Arc:new{p0=a0, p1=a1, centre=c}
9
10 -- A Bezier curve of order 3.
11 b0 = Vector3:new{x=0.1, y=0.1}
12 b3 = Vector3:new{x=0.4, y=0.7}
13 b1 = b0 + Vector3:new{y=0.2}
14 b2 = b3 - Vector3:new{x=0.2}
15 my_bez = Bezier:new{points={b0, b1, b2, b3}}
16
17 -- A line between the Bezier and the arc.
18 my_line = Line:new{p0=b0, p1=a0}
19
20 -- Put an arc through three points.
21 a2 = Vector3:new{x=0.8, y=0.2}
22 a3 = Vector3:new{x=1.0, y=0.3}
23 my_other_arc = Arc3:new{p0=a1, pmid=a2, p1=a3}
24
25 dofile("sketch-path-example-1.lua")
```

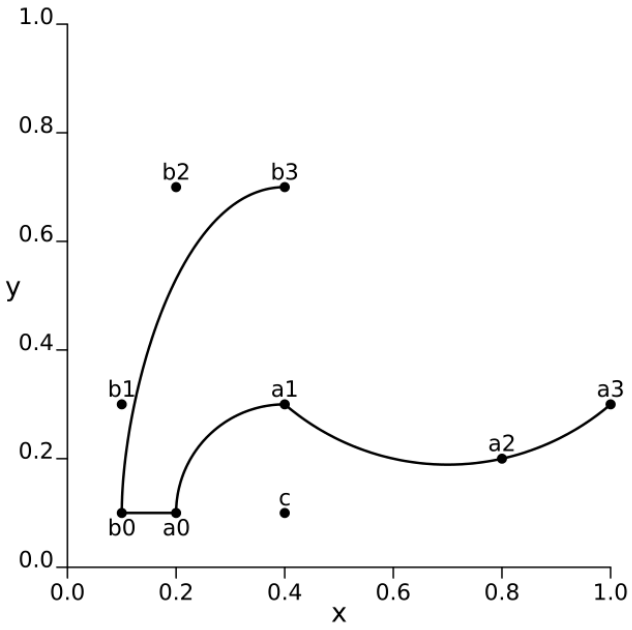



图 2.2：一些从 Vector3 对象定义并渲染为 SVG 图像的简单路径。

### 2.2.3 用户自定义函数路径

为了实现路径定义的最大灵活性，您可以提供一个 Lua 函数来将参数 t 转换到物理空间。下面的脚本，从代码中提取的尖头体算例的几何定义，展示了如何在二维情况下执行此操作。在第 23 行，通过提供 Lua 函数的名称作为字符串来构造 LuaFnPath。从第 13 行开始定义的 xypath 函数接受参数 t 的值(范围从 0.0 到 1.0)，并返回一个有 x、y(可能还有 z)命名组件的表，这些组件表示参数值 t 在空间中的位置。从第 4 行开始定义的 y(x)是一个帮助函数，用于使脚本看起来更像教科书[4]中描述的公式。但是没有必要以这种方式分割功能。图 2.3 显示了路径的图像。

```
1 -- path-example-2.lua
2 -- Extracted from the 2D/sharp-body example.
3
4 function y(x)
5 -- (x,y)-space path for x>=0
6 if x <= 3.291 then
7 return -0.008333 + 0.609425 * x - 0.092593 * x * x
8 else
9 return 1.0
10 end
11 end
```

```
12
13 function xypath(t)
14 -- Parametric path with 0<=t<=1.
15 local x = 10.0*t
16 local yval = y(x)
17 if yval < 0.0 then
18 yval = 0.0
19 end
20 return {x=x, y=yval}
21 end
22
23 my_path = LuaFnPath:new{luaFnName="xypath"}
24
25 dofile("sketch-path-example-2.lua")
```

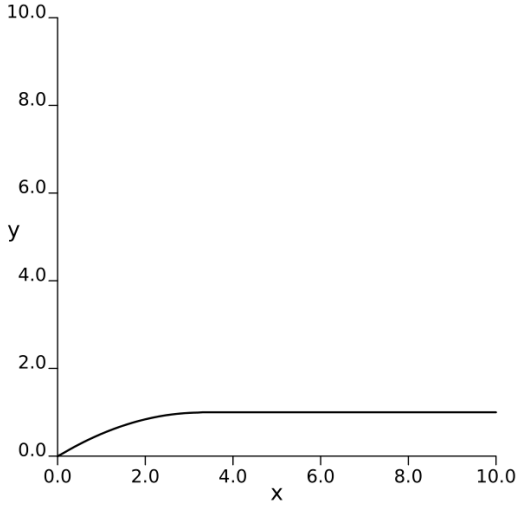



图 2.3:通过 Lua 函数定义并以 SVG 格式显示的路径。在 Eilmer4 的二维/锐体仿真练习中，这一特定路径用于表示锐体的轮廓。

### 2.2.4 复合路径

有时您希望有单一的参数路径对象传递给网格生成函数，但是定义较小的部分是非常方便的。可以构造以下路径子类型:

- Polyline:new{segments={$p_0$ ,$p_1$ ,...,$p_n$ }}:由 $p_0$ 到 $p_n$ 组成的复合路径，每一段都是之前定义的 path 对象。根据弧长对单独的段重新参数化，使复合曲线参数为 0≤t < 1。
- Spline:new{points={ $\vec{b}_0$，$\vec{b}_1$， ...，$\vec{b}_n$}}:一条从$\vec{b}_0$到$\vec{b}_1$再到$\vec{b}_n$的样条曲线。

Spline 实际上是包含 n-1 个三维贝塞尔曲线段的特殊折线。

- Spline2:new{filename="something.dat"}:由包含插值点 xyz 坐标的文件中构造的一种样条曲线，每条线一个点。坐标值应该用空格分隔。如果缺少 y 或 z 值，则默认它们为零。

下面的脚本构建了一个类似于第一个示例的组合路径对象:

```
1 -- path-example-3.lua
2
3 -- An arc with a specified pair of end-points and a centre.
4 c = Vector3:new{x=0.4, y=0.1}
5 radius = 0.2
6 a0 = Vector3:new{x=c.x-radius, y=c.y}
7 a1 = Vector3:new{x=c.x, y=c.y+radius}
8 my_arc = Arc:new{p0=a0, p1=a1, centre=c}
9
10 -- A Bezier curve of order 3.
11 b3 = Vector3:new{x=0.1, y=0.1}
12 b0 = Vector3:new{x=0.4, y=0.7}
13 b1 = b0 - Vector3:new{x=0.2}
14 b2 = b3 + Vector3:new{y=0.2}
15 my_bez = Bezier:new{points={b0, b1, b2, b3}}
16
17 -- A line between the Bezier and the arc.
18 my_line = Line:new{p0=b3, p1=a0}
19
20 -- Put an arc through three points.
21 a2 = Vector3:new{x=0.8, y=0.2}
22 a3 = Vector3:new{x=1.0, y=0.3}
23 my_other_arc = Arc3:new{p0=a1, pmid=a2, p1=a3}
24
25 one_big_path = Polyline:new{segments={my_bez, my_line, my_arc,
26                                       my_other_arc}}
27
28 dofile("sketch-path-example-3.lua")
```

注意，想要创建一条可以遍历的路径，我们需要使用与前面示例相反顺序的点定义 Bezier 曲线。

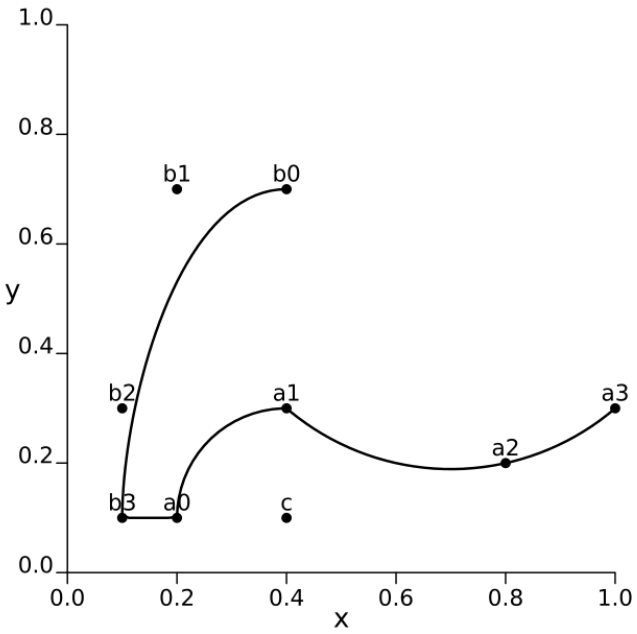



图 2.4:由一组路径对象定义并以 SVG 格式呈现的单一复合路径。

如果我们使用与构造 Polyline 相同的数据点，我们可以在图 2.5 中看到使用 Spline 构造函数所产生的差异。有时，它就是你想要的，但其它时候并不如此。

```
1 -- path-example-4.lua
2 -- A single spline through the same data points.
3 c = Vector3:new{x=0.4, y=0.1}
4 radius = 0.2
5 a0 = Vector3:new{x=c.x-radius, y=c.y}
6 a1 = Vector3:new{x=c.x, y=c.y+radius}
7
8 b3 = Vector3:new{x=0.1, y=0.1}
9 b0 = Vector3:new{x=0.4, y=0.7}
10 b1 = b0 - Vector3:new{x=0.2}
11 b2 = b3 + Vector3:new{y=0.2}
12
13 a2 = Vector3:new{x=0.8, y=0.2}
14 a3 = Vector3:new{x=1.0, y=0.3}
15
16 my_spline = Spline:new{points={b0,b1,b2,b3,a0,a1,a2,a3}}
17
18 dofile("sketch-path-example-4.lua")
```

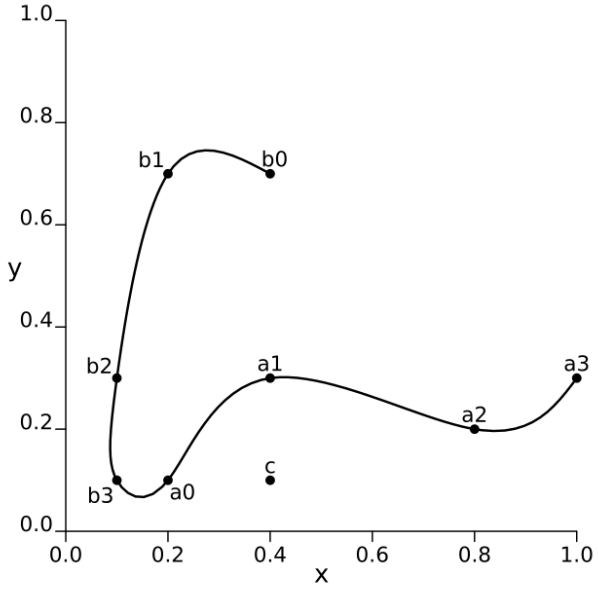



图 2.5:使用 Polyline 定义在同一组点上的样条路径段线。

### 2.2.5 导出路径

我们可以构造新的路径对象作为之前定义的路径的导数。下面列出了许多具有不同转换的构造函数:

- ArcLengthParameterizedPath:new{underlying_path=*pth*}:有时，Bezier 曲线、Spline 曲线或 Polyline 可能有它的定义点分布，这样构建在其上的网格就不能以较好的方式聚集。为了解决这个问题，指定用弧长参数化的曲线可能是有用的。
- SubRangedPath:new{underlying_path=*pth*, t0=t 0 , t1=t 1 }:新路径是原始路径的一个子集。如果 t0 > t1，则新路径以相反的方向遍历。
- ReversedPath:new{underlying_path=*pth*}:用于构建相同边缘的面。例如，如果第一个面的东边缘与第二个面的南边缘相同，那么移动的方向就会颠倒。
- TranslatedPath:new{original_path=*pth*, shift=$\vec{a}$}:新路径上的点用$\vec{a}$从原路径上对应的点平移。
- MirrorImagePath:new{original_path=*pth*, point=$\vec{p}$, normal=$\vec{n}$}:点$\vec{p}$是构成图像平面的锚点，$\vec{n}$是该平面的单位法线。
- RotatedAboutZAxisPath:new{original_path=pth, angle=θ}:在三维中对构建

旋转体有用。

下面是一些构建派生路径的例子，结果如图 2.6 所示:

```
1 -- path-example-5.lua
2 -- Transformed Paths
3
4 b0 = Vector3:new{x=0.1, y=0.1}
5 b1 = b0 + Vector3:new{x=0.0, y=0.05}
6 b2 = b1 + Vector3:new{x=0.05, y=0.1}
7 b3 = Vector3:new{x=0.8, y=0.1}
8 path0 = Bezier:new{points={b0,b1,b2,b3}}
9
10 path1 = TranslatedPath:new{original_path=path0, shift=Vector3:new{y=0.5}}
11 path1b = MirrorImagePath:new{original_path=path1,
12                               point=Vector3:new{x=0.5, y=0.6},
13                               normal=Vector3:new{y=1.0}}
14
15 path2 = TranslatedPath:new{original_path=path0, shift=Vector3:new{y=0.5}}
16 path2b = ArcLengthParameterizedPath:new{underlying_path=path2}
17
18 dofile("sketch-path-example-5.lua")
```

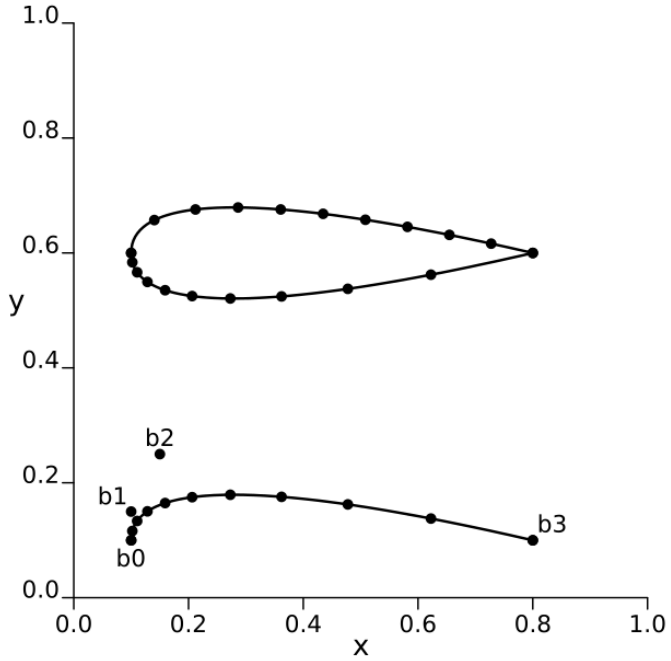



图 2.6：用图下方 b0、b1、b2、b3 四个点定义的原始贝塞尔曲线。图上方显示了一对派生路径。所有路径上的点表示参数 t 的相同增量。请注意在 x,y 空间中，ArcLengthParameterizedPath 的分布才更加均匀。

## 2.3 面

ParametricSurface 类表示可以由 Path 对象构造的二维对象。它们可以作为 ParametricSurface 对象传递给 StructuredGrid 构造函数(在 3.2 节)，也可以用来形成三维 ParametricVolume 对象(章节 2.4)的边界曲面。ParametricSurface 对象可以评估为参数对(r,s)的函数，并产生相应的 Vector3 值，可表示建模空间中的一点。在这里 $0 \le r \le 1$ 以及 $0 \le s \le 1$。

常用的面函数如下：

- CoonsPatch:new{south=$path_S$ ,north=$path_N$ ,west=$path_W$ ,east=$path_E$ }:
在四种路径之间的超限插值面。预计路径会在面的各个角点连接，例如：$path_S(0) = path_W(0) = p_{00}$, $path_S(1) = path_E(0) = p_{10}$, $path_N(0) = path_W(1) = p_{01}$ and $path_N(1) = path_E(1) = p_{11}$。这个表面的布局见图 2.7 的左侧部分。尽管指定已命名边的顺序并不重要，但是要注意构成面边界的 Path 元素的方向。南北边界由西向东推进，参数 r 从 0 增加到 1。西、东边界由南向北推进，参数 s 从 0 增加到 1。如果程序的准备阶段出现报错：面的边角是“开放的”。这可能是因为有一个或多个边界路径的方向有问题而导致的。
- CoonsPatch:new{p00=$\vec{p}_{00}$ , p10=$\vec{p}_{10}$ , p11=$\vec{p}_{11}$ , p01=$\vec{p}_{01}$}:由它的角点定义的四边形面。直线段可以(隐式地)连接角。这对于构建可用直边面平铺的简单区域非常方便，因为您不需要显式地生成 Line 对象来形成每个面的边缘。请注意，指定角的顺序并不重要。
- AOPatch:new{south=$path_S$ ,north=$path_N$ ,west=$path_W$ ,east=$path_E$ ,nx=10, ny=10}:由四条路径限定的内表面。在构造时，这个表面会建立一个分辨率为 nx 和 ny 的背景网格。，这是基于椭圆网格发生器[5]的构造方法，它试图保持背景网格在边缘附近的正交性，同时也试图保持整个表面的单元面积相等。背景网格被保留在 AOPatch 对象中，如果稍后将 AOPatch 传递给网格生成器，则通过在背景网格内插值生成最终的网格。10×10 的默认背景网格在简单的情况下可能工作得相当好。如果边界路径有较大的曲率，则可能有利于提高背景网格的分辨率，从而使最终插值的网格不穿越边界路径。如果最后的网格线聚集在边界附近，尤其要注意这一点。设置更高分辨率的背景网格需要更多的迭代过程来达到收敛，您可能会看到一个警告信息：迭代没有收

敛。所以，非收敛的背景网格通常也适合使用，因为从 CoonsPatch 网格开始，每次迭代都应该改进正交性标度和单元区域的分布。

- AOPatch:new{ p00=$\vec{p}_{00}$ , p10=$\vec{p}_{10}$ , p11=$\vec{p}_{11}$ , p01=$\vec{p}_{01}$ , nx=10, ny=10}：由它的角定义的四边形表面。直线段可以(隐式地)连接角。如图 2.7 所示，与相的 CoonsPatch 的不同之处在于，背景网格在这里试图与边缘正交，并在表面上保持相同的单元格面积。

下面是两个设置路径的简单例子：

```
1 -- surface-example-1.lua
2
3 -- Transfinite interpolated surface
4 a = Vector3:new{x=0.1, y=0.1}; b = Vector3:new{x=0.4, y=0.3}
5 c = Vector3:new{x=0.4, y=0.8}; d = Vector3:new{x=0.1, y=0.8}
6 surf_tfi = CoonsPatch:new{north=Line:new{p0=d, p1=c},
7                           south=Line:new{p0=a, p1=b},
8                            west=Line:new{p0=a, p1=d},
9                            east=Line:new{p0=b, p1=c}}
10
11 -- Knupp's area-orthogonality surface
12 xshift = Vector3:new{x=0.5}
13 p00 = a + xshift; p10 = b + xshift;
14 p11 = c + xshift; p01 = d + xshift;
15 surf_ao = AOPatch:new{p00=p00, p10=p10, p11=p11, p01=p01}
16
17 dofile("sketch-surface-example-1.lua")
```

除了上面提到的构造函数外，还有一个比较方便的函数：makePatch{south=$path_S$, north=$path_N$, west=$path_W$, east=$path_E$,gridType="TFI"}，它可以返回一个插值表面。即返回一个 CoonsPatch (默认是 TFI)或者 AOPatch(gridType="AO")对象。如果您碰巧要将一个旧示例移植到 Eilmer4，那么这种方便性实际上仅限于允许您的脚本更类似于 Eilmer3 的输入脚本。

还有更多的表面构造函数，如下所示。但是，您需要参考它们的源代码以获取文档。

- ChannelPatch:new{south=$path_S$, north=$path_N$, ruled=false,pure2D=false}:两条路径之间的内插面。默认情况下,三维贝塞尔曲线用于连接定义曲线之间的区域，形成一个与这些曲线正交的网格。为 ruled 参数提供 true 值将导致连接曲线为直线段。如果你将这个面集成到一个更大的结构中，可以通过分别调

用它从 0.0 和 1.0 的参数的 make_bridgeing_path 方法来获得东边和西边的边界曲线。在二维结构中，pure2D 的参数可能会设置为 true，来使得 z 轴的值为零。

• SweptPathPatch:new{west=$path_W$, south=$path_S$}:一个面，由南路径固定，并由扫过南路径的西路径生成。

• LuaFnSurface:new{luaFnName="*myLuaFnName*"}:由用户提供的函数 f(r,s)定义的曲面。用户函数返回一个包含三个标记坐标的表，代表参数值 r 和 s 在三维空间中的点。如果您试图构建一个二维模拟，只需将 z 坐标设置为零。

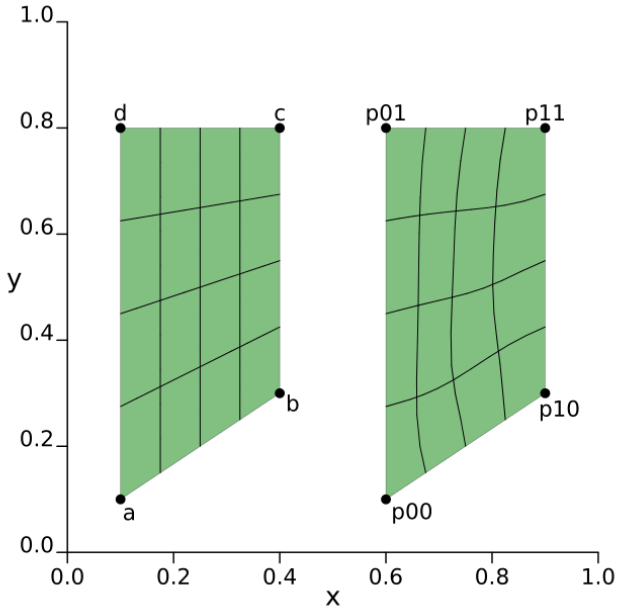



图 2.7:一个 CoonsPatch 和 AOPatch 函数的例子。在这些块上画的南到北向线段代表常数或 r，而西到东线代表常数 s。

• MeshPatch:new{sgrid=*myStructuredGrid*}:一个面，通过之前定义在四边形面上的结构化网格来定义。结构化网格可能已经从外部网格生成器导入(VTK 格式)。

• SubRangedSurface:new{underlying_surface=*mySurf*, r0=0.0, r1=1.0,s0=0.0, s1=1.0}:可以通过设置合适的 r0, r1, s0 和 s1 的值来选择一个曲面的截面。显示了整个参数范围的默认值。

• BezierPatch:new{points=Q}：由一张贝塞尔控制点组成的网定义的面。控制点由点 Q[n][m]组成，其中 n 为 r 方向上的控制点个数，m 为 s 方向上的控制点个数。Bezier 面由 r 方向上的 n-1 阶 Bezier 曲线族和 s 方向上的 m-1 阶

Bezier 曲线族构成。本节的末尾有一个构造示例。

以下是设置 ChannelPatches 的两种不同风格例子:

```
-- surface-example-2.lua

path1 = Arc3:new{p0=Vector3:new{x=0.2, y=0.3},
                 pmid=Vector3:new{x=0.5, y=0.25},
                 p1=Vector3:new{x=0.8, y=0.35}}
p = Vector3:new{y=0.2}; n = Vector3:new{y=1.0}
path2 = MirrorImagePath:new{original_path=path1,
                            point=p, normal=n}
channel = ChannelPatch:new{north=path1, south=path2,
                           ruled=false, pure2D=true}
bpath0 = channel:make_bridging_path(0.0)
bpath1 = channel:make_bridging_path(1.0)

yshift = Vector3:new{y=0.5}
path3 = TranslatedPath:new{original_path=path1, shift=yshift}
path4 = TranslatedPath:new{original_path=path2, shift=yshift}
channel_ruled = ChannelPatch:new{north=path3, south=path4,
                                 ruled=true, pure2D=true}

dofile("sketch-surface-example-2.lua")
```

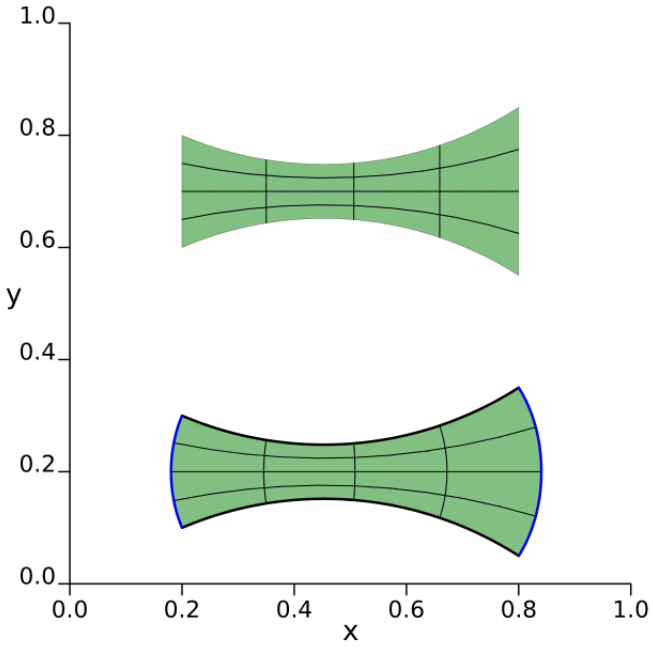



图 2.8:ChannelPatch 对象的示例。在块上绘制的南北向线代表常数 r，而东西方向线代表常数 s。下面的块使用(默认)贝塞尔曲线连接南和北向的定义路径。南北向路径以粗黑线呈现，r = 0 和 r = 1 的连接路径以蓝色显示。上面的示例使用了直线连接路径。

下面的脚本显示了 SweptPathPatch 的构造，其中南边的路径固定表面，西边的路径从它的路径上扫过。注意，在图 2.9 中，为西面路径建造的路径实际上不是最终表面的一部分。LuaFnSurface 示例将(r,s)空间中的单位正方形映射到(x,y)平面中两个圆之间的区域。

```
1 -- surface-example-3.lua
2
3 path1 = Arc3:new{p0=Vector3:new{x=0.2, y=0.7},
4                  pmid=Vector3:new{x=0.5, y=0.65},
5                  p1=Vector3:new{x=0.8, y=0.75}}
6 path2 = Line:new{p0=Vector3:new{x=0.1, y=0.7},
7                  p1=Vector3:new{x=0.1, y=0.9}}
8 swept = SweptPathPatch:new{south=path1, west=path2}
9
10 function myFun(r, s)
11 -- Map unit square to the region between two circles
12 local theta = math.pi*2*r
13 local bigR = 0.05 * (1-s) + 0.2 * s
14 local x0 = 0.5; local y0 = 0.3
15 return {x=x0+bigR * math.cos(theta), y=y0+bigR * math.sin(theta), z=0.0}
16 end
17 funSurf = LuaFnSurface:new{luaFnName="myFun"}
18
19 dofile("sketch-surface-example-3.lua")
```

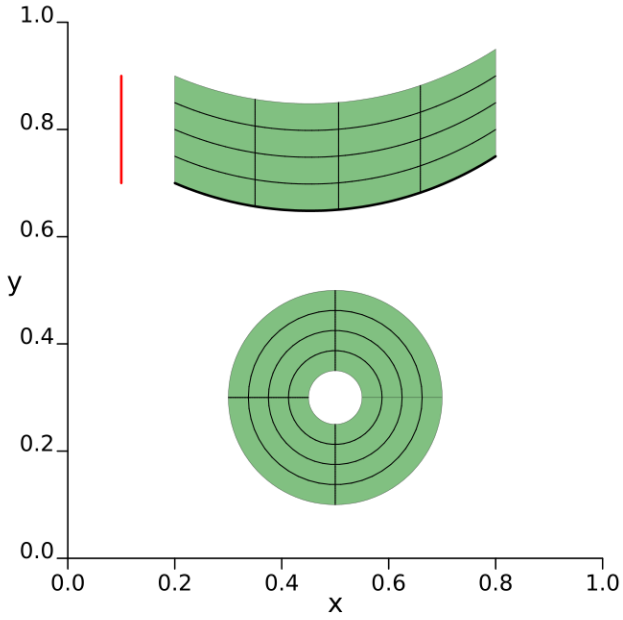



图 2.9:SweptPathPatch(上)和 LuaFnSurface(下)的示例。在块上画的南北向的线代表常数或 r，而东西方向的线代表常数 s。

下面的脚本显示了构建 BezierPatch 的示例。在图 2.10 中，贝塞尔面以绿色呈现。控制点以 Q[i][j]的形式标记。连接控制点的红线形成了贝塞尔面的控制网。

```
-- surface-example-4.lua
-- BezierPatch in 3D
n = 4
m = 3
Q = {}
for i=1,n do Q[i] = {} end
Q[1][1] = Vector3:new{x=0.0, y=3.0, z=0.0}
Q[1][2] = Vector3:new{x=0.0, y=3.0, z=2.0}
Q[1][3] = Vector3:new{x=0.0, y=3.0, z=4.0}
Q[2][1] = Vector3:new{x=3.5, y=1.0, z=-1.0}
Q[2][2] = Vector3:new{x=5.0, y=2.5, z=2.5}
Q[2][3] = Vector3:new{x=5.5, y=4.0, z=6.0}
Q[3][1] = Vector3:new{x=8.0, y=4.0, z=1.0}
Q[3][2] = Vector3:new{x=8.0, y=1.5, z=4.0}
Q[3][3] = Vector3:new{x=8.5, y=2.0, z=7.0}
Q[4][1] = Vector3:new{x=10.0, y=1.0, z=-2.0}
Q[4][2] = Vector3:new{x=11.0, y=2.0, z=2.0}
Q[4][3] = Vector3:new{x=11.0, y=3.0, z=6.0}
bezPatch = BezierPatch:new{points=Q}

dofile("sketch-surface-example-4.lua")
```

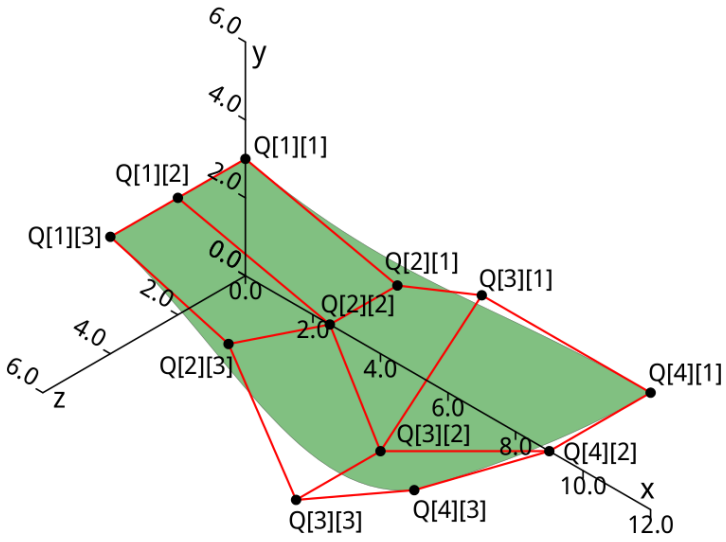



图 2.10:带有控制点(红色)和控制网的 BezierPatch(绿色)示例。

### 2.3.1 面上的路径

在 Eilmer3 中，Path 的子类可用于在参数曲面上构造路径。和底层表面一样，它由一对函数对象组成，用于求 r(t)和 s(t)的值。这些都局限于 t 的线性函数。在 Eilmer4 中，在参数曲面上创建路径的方法是构造 LuaFnPath。这更加灵活，因为现在任何函数都可以用于 r(t)和 s(t)。

## 2.4 体

在三维空间中的工作量显著增加了，主要是在指定流域所需的细节量上。在二维中，一个简单区域可能被表示为一个带有 4 个边界面的单一面，但在三维中，简单的三维体被 6 个面所包围，每个面都有 4 条边。如果两个面共用一条边，那么这个立体上就会有 12 条不同的边。ParametricVolume 对象，如图 2.11 所示，将 r,s,t 参数坐标映射到立体图形内的 x,y,z 物理空间坐标。为了帮助我们理解角、表面和指数的方向，您可以根据附录 A 的开发计划建造一个模型块。这应该会让我们回忆起幼儿园和小学的美好时光，至少对我来说是这样的。

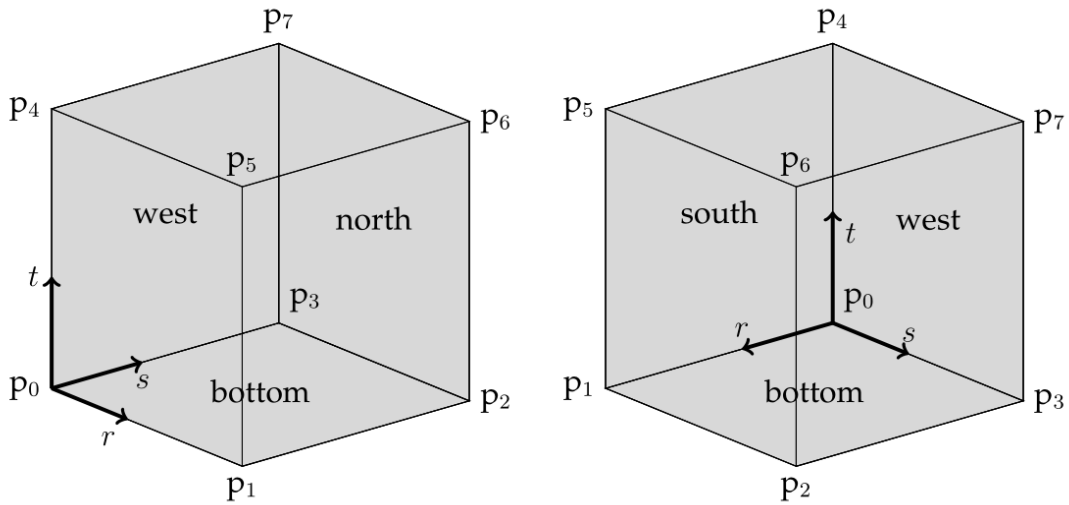



图 2.11:由六个边界面定义参数体的两个视图。这些图形是模糊的，但每个都应该显示一个空心框，每个视图中的远表面都做了标记。近表面是透明的，没有标记。在左侧视图中，$p_1$-$p_5$ 的边最接近观察者。在右侧视图中，$p_2$-$p_6$ 的边缘最接近观察者。

还有许多构造器，如下：

- TFIVolume:new{north=$surf_N$, east=$surf_E$, south=$surf_S$, west=$surf_W$, top=$surf_T$,

bottom=$surf_B$}：从一组六个参数的表面构造参数体，形成一个贴体的六面体。它假定表面是一致的，因为它们在其边缘相应的对齐，在边界表面上不留下任何缝隙。还需要必要的检查，以确保角点位置(至少)是一致的。

- TFIVolume:new{vertices={$\vec{p}_0$，$\vec{p}_1$，$\vec{p}_2$，$\vec{p}_3$，$\vec{p}_4$，$\vec{p}_5$，$\vec{p}_6$，$\vec{p}_7$}:通过顶点定义三维体，并假设这些点之间有直线边。子脚本对应于图 2.11 中的标记点。
- SweptSurfaceVolume:new{face0123=*surf*, edge04=*path*}:通过沿着路径挤压表面来构建三维体。当评估点时，实际上计算是通过 path(t)+surf(r,s) - surf(0,0)来完成的，这样路径有效地固定了 x,y,z 空间中的三维体。
- TwoSurfaceVolume:new{face0123=$surf_{bottom}$, face4567=$surf_{top}$}:通过在底部和顶部表面之间的线性插值来构造三维体。当计算点时，实际计算表达式为：(1.0−t)×$surf_{bottom}$(r,s)+t×$surf_{top}$(r,s)，从而在两个曲面上的相应点之间形成一条直线。
- LuaFnVolume:new{luaFnName="*myLuaFunction*"}:根据用户提供的 Lua 函数构造三维体。例如，简单的单元立方体可以定义为:

```
function myLuaFunction(r, s, t)
   return {x=r, y=s, z=t}
end
```

- SubRangedVolume:new{underlying_volume=*pvolume*, r0=0, r1=1, s0=0,s1=1, t0=0, t1=1}:将三维体构造为先前定义的体的子区域。没有指定任何 r, s, t 范围限制，它们都被假设为所示的默认值。

可以定义的最简单的参数体是由顶点定义的正六面体块。虽然这个例子很简单，但也有同样微妙的问题需要考虑，就像我们在构造 Bezier 路径时看到的那样。在脚本的第 21 行，顶点表是通过隐式的点编号进行组装的。请注意，默认情况下，Lua 将从 1 开始对表项进行编号。这是与在 D 编程中编写的核心程序的持续磨合点，其中数组下标从零开始，所以如果您显式地给顶点编号，一定要注意这一点。构造函数的 Lua 接口代码想要接收带有默认 Lua 编号的点数，尽管我们在文档中一直尝试使用 D 语言索引。在本例中，我们使用表的文字表达式来隐藏索引问题。

```
1 -- volume-example-1.lua
2 -- Transfinite interpolated volume
```

```
3
4 Lx = 0.2; Ly = 0.1; Lz = 0.5 -- size of box
5 x0 = 0.3; y0 = 0.1; z0 = 0.0 -- location of vertex p000/p0
6
7 -- There is a standard order for the 8 vertices that define
8 -- a volume with straight edges.
9 p000 = Vector3:new{x=x0, y=y0, z=z0} -- p0
10 p100 = Vector3:new{x=x0+Lx, y=y0, z=z0} -- p1
11 p110 = Vector3:new{x=x0+Lx, y=y0+Ly, z=z0} -- p2
12 p010 = Vector3:new{x=x0, y=y0+Ly, z=z0} -- p3
13 p001 = Vector3:new{x=x0, y=y0, z=z0+Lz} -- p4
14 p101 = Vector3:new{x=x0+Lx, y=y0, z=z0+Lz} -- p5
15 p111 = Vector3:new{x=x0+Lx, y=y0+Ly, z=z0+Lz} -- p6
16 p011 = Vector3:new{x=x0, y=y0+Ly, z=z0+Lz} -- p7
17
18 -- Assemble a table literal to avoid the issue
19 -- of numeric indices starting at 1 in Lua code
20 -- whilst starting at 0 in D code.
21 vList = {p000, p100, p110, p010, p001, p101, p111, p011}
22
23 my_vol = TFIVolume:new{vertices=vList}
24
25 dofile("sketch-volume-example-1.lua")
```

让我们使用在输入脚本中定义的 Lua 函数来构建相同的盒形三维体。该函数必须返回一个带有 x、y 和 z 坐标标记的表。该脚本实际上比前面的示例短，但回调到 Lua 解释器以获取坐标值的计算机制是有代价的。对于带有大网格的复杂几何图形，这可能会使准备阶段运行缓慢。对于简单的几何图形，这个成本很小。但是您准备脚本的时间很宝贵，所以应该选择您觉得方便的方法进行操作。

```
1 -- volume-example-2.lua
2 -- Volume defined on a Lua function.
3
4 Lx = 0.2; Ly = 0.1; Lz = 0.5 -- size of box
5 x0 = 0.3; y0 = 0.1; z0 = 0.0 -- location of vertex p000/p0
6
7 function myLuaFn(r, s, t)
8 return {x=x0+Lx * r, y=y0+Ly * s, z=z0+Lz * t}
9 end
10
11 my_vol=LuaFnVolume:new{luaFnName="myLuaFn"}
12
13 dofile("sketch-volume-example-2.lua")
```

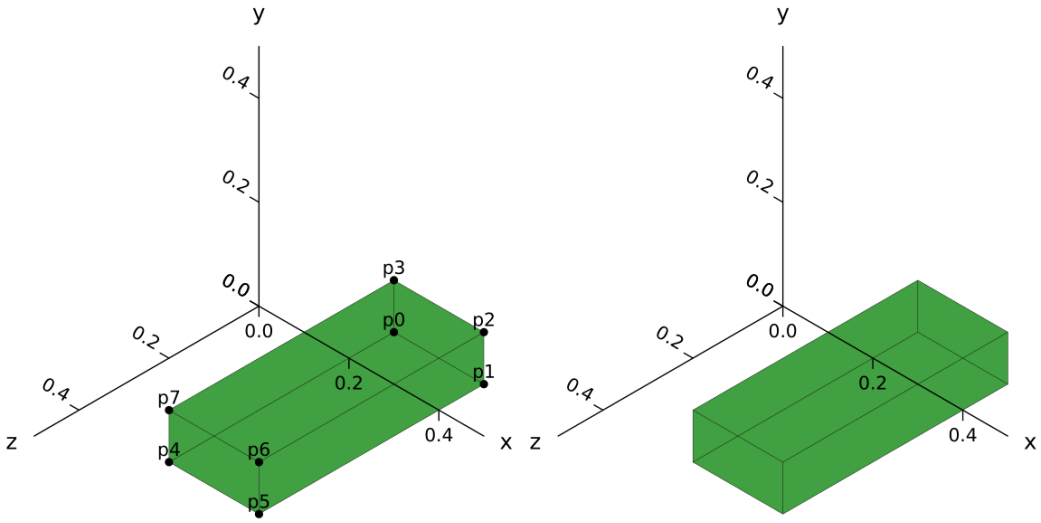



图 2.12:正六面体的等距视图。左视图显示了由顶点定义的三维体。右边的视图显示了由 Lua 函数定义的相同三维体。

一个经常被研究的三维结构是圆柱绕流。下面的脚本围绕柱面的一部分构造了一个流动区域。我们首先在 z=0 平面上构建(后续的)底面，然后沿着平行于 z 轴的 b0-d 线扫出三维体。

```
1 -- volume-example-3.lua
2 -- Parametric volume construced by sweeping a surface..
3
4 L = 0.4 -- cylinder length
5 R = 0.2 -- cylinder radius
6
7 -- Construct the arc along the edge of the cylinder
8 a0 = Vector3:new{x=R, y=0}; a1 = Vector3:new{x=0, y=R}
9 c = Vector3:new{x=0, y=0}
10 arc0 = Arc:new{p0=a0, p1=a1, centre=c}
11
12 -- Use a Bezier curve for the edge of the outer surface.
13 b0 = Vector3:new{x=1.5 * R, y=0}; b1 = Vector3:new{x=b0.x, y=R}
14 b3 = Vector3:new{x=0, y=2.5 * R}; b2 = Vector3:new{x=R, y=b3.y-R/2}
15 bez0 = Bezier:new{points={b0, b1, b2, b3}}
16
17 -- Construct the bottom surface
18 surf0 = CoonsPatch:new{west=bez0, east= arc0,
19                        south=Line:new{p0=b0, p1=a0},
20                        north=Line:new{p0=b3, p1=a1}}
21 -- Construct the edge along which we will sweep the surface.
22 d = Vector3:new{x=b0.x, y=b0.y, z=b0.z+L}
23 line0 = Line:new{p0=b0, p1=d}
```

```
24
25 my_vol = SweptSurfaceVolume:new{face0123=surf0, edge04=line0}
26
27 dofile("sketch-volume-example-3.lua")
```

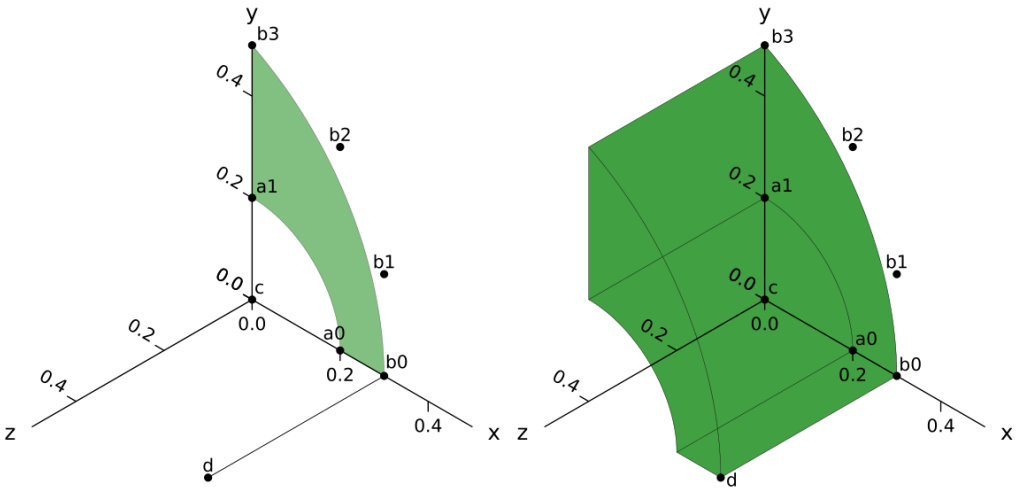



图 2.13:取圆柱体的一部分构造的三维体。左视图显示沿 b0-d 线扫过的表面，右视图显示构造的三维体。

## 2.5 操作元素

使用编程语言定义几何图形的主要优势是：同样的编程语言可以用来参数化和自动操作您的几何图形和网格。本节将举例说明如何使用此功能。

### 2.5.1 参数化几何

在最简单的形式中，可以提前定义一些变量，如比例因子或壁面的特定角度。下图是 flow-solver 用户指南中尖锥模拟的参数化示例。在前几行中，要么用文字，要么用代数表达式，将几个 Lua 变量初始化为特定的值。这些变量被用来定义元素，这些元素以全参数的方式用来定义流域。

```
1 -- sharp-cone-parameterised-example.lua
2
3 -- Parameters defining cone and flow domain.
4 theta = 32 -- cone half-angle, degrees
5 L = 0.8 -- axial length of cone, metres
```

```
6 rbase = L*math.tan(math.pi * theta/180.0)
7 x0 = 0.2 -- upstream distance to cone tip
8 H = 2.0 -- height of flow domain, metres
9
10 -- Set up two quadrilaterals in the (x,y)-plane by first defining
11 -- the corner nodes, then the lines between those corners.
12 a = Vector3:new{x=0.0, y=0.0} -- f----e---------d
13 b = Vector3:new{x=x0, y=0.0} -- | | |
14 c = Vector3:new{x=x0+L, y=rbase} -- |quad| quad |
15 d = Vector3:new{x=x0+L, y=H} -- | 0 | 1 |
16 e = Vector3:new{x=x0, y=H} -- | | ___---c
17 f = Vector3:new{x=0.0, y=H} -- a----b---
18
19 -- Define the two surfaces
20 quad0 = makePatch{p00=a, p10=b, p11=e, p01=f}
21 quad1 = makePatch{p00=b, p10=c, p11=d, p01=e, gridType="ao"}
```

### 2.5.2 eval 函数

eval 函数是最有用的函数之一。在由(t)、(s,t)或(r,s,t)定义的参数位置中，这个函数评估当前路径、表面或三维体元素，并返回一个新的点。在 2.1 节（my_path:eval(t), my_surf:eval(s,t) 或 my_vol:eval(r,s,t)）描述的作为每个面向对象的 Lua 协定的特殊几何对象中，通过调用 eval 方法来执行函数。或者可以直接调用几何对象，如 my_path(t)。有关如何使用 eval 函数来操作几何学，请参阅以下部分。

### 2.5.3 细分路径

在几何生成中，一个常见的任务是细分现有的直线，并在断点处添加一个新点。特别是当使用参数曲线，如 Bezier 贝塞尔曲线或 Splines 样条曲线时，简单地定义在一点上相交的两条线并不是一个理想的选择。如下面的例子和图 2.14 所示，渐变(和高阶导数)的线在界面处不匹配，所以线的整体形状可以改变。更好的方法是使用一条线，然后使用 SubRangedPath 将其分割为两条线，然后在接口上创建一个新点。

下面是分割 Bezier 曲线的一个例子，其结果如图 2.14 所示。

```
1 -- split-path-example.lua
2
```

```
-- Single Bezier curve of order 4.
b={}
b[1] = Vector3:new{x=0.1, y=0.7}
b[2] = Vector3:new{x=0.15, y=0.8}
b[3] = Vector3:new{x=0.5, y=0.9}
b[4] = Vector3:new{x=0.6, y=0.7}
b[5] = Vector3:new{x=0.9, y=0.7}
single_bez = Bezier:new{points={b[1], b[2], b[3], b[4], b[5]}}

-- offset points in y-direction for 2nd and 3rd line
c = {}
d = {}
for i,v in ipairs(b) do
c[i] = Vector3:new{x=v.x, y=v.y-0.35} -- offset points by -0.35
d[i] = Vector3:new{x=v.x, y=v.y-0.7} -- offset points by -0.7
end

-- create two Bezier curves
two_bez_0 = Bezier:new{points={c[1], c[2], c[3]}}
two_bez_1 = Bezier:new{points={c[3], c[4], c[5]}}

-- create split line
t = 0.5 -- define parameterised location of split
bez = Bezier:new{points={d[1], d[2], d[3], d[4], d[5]}}
C = bez:eval(t)
line_d1_C = SubRangedPath:new{underlying_path=bez, t0 = 0.0, t1 = t}
line_C_d5 = SubRangedPath:new{underlying_path=bez, t0 = t, t1 = 1.0}

dofile("sketch-split-path-example.lua")
```

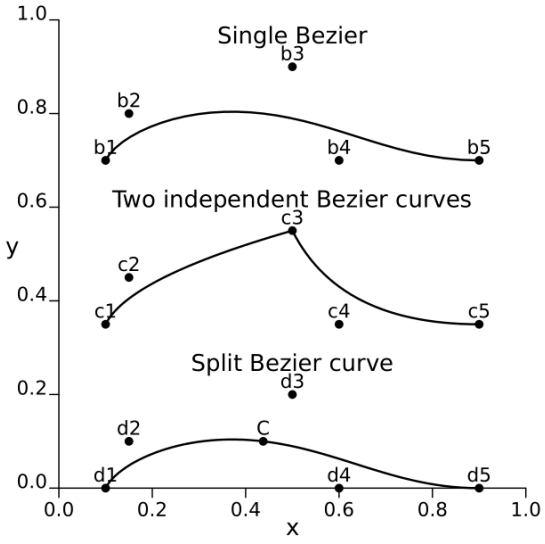



图 2.14:使用不同方法将 Bezier 曲线分割成两个路径元素的效果。

### 2.5.4 创建一条与路径垂直的线

当创建一个符合壁面的精细网格时，一个常见的要求是让线和网格构建点定位于代表壁面的路径。使用 eval 函数可以很容易地自动化这个过程。

在下面的代码中，点 A 创建于标准化位置 t=0.3 处。然后用有限差分法求出 A 点的切向量(等于路径的梯度)。这是一个壁面法向量，法向量是通过交换 x 和 y 分量并减去 x 分量的符号而产生的。最后，通过在 A 点上添加一个缩放后的法向量来确定 B 点的位置，结果如图 2.15 所示。

```
1 -- point-normal-to-wall-example.lua
2
3 -- Single Bezier curve of order 4.
4 b={}
5 b[1] = Vector3:new{x=0.1, y=0.1}
6 b[2] = Vector3:new{x=0.15, y=0.2}
7 b[3] = Vector3:new{x=0.5, y=0.3}
8 b[4] = Vector3:new{x=0.6, y=0.1}
9 b[5] = Vector3:new{x=0.9, y=0.1}
10 single_bez = Bezier:new{points={b[1], b[2], b[3], b[4], b[5]}}
11
12 -- Create point along wall
13 t = 0.3
14 A = single_bez:eval(t)
15
16 -- Calculate gradient vector by evaluating path at t +/- dt
17 dt = 0.01
18 gradient = single_bez:eval(t+dt) - single_bez:eval(t-dt)
19
20 -- create wall normal vector and normalise
21 normal = Vector3:new{x=-gradient.y, y=gradient.x}
22 normal:normalize()
23
24 -- Create point B at distance L
25 L = 0.2
26 B = A + L * normal -- add vectors
27
28 -- Create line connecting A and B
29 mypath = Line:new{p0=A, p1=B}
30
31 dofile("sketch-point-normal-to-wall-example.lua")
```

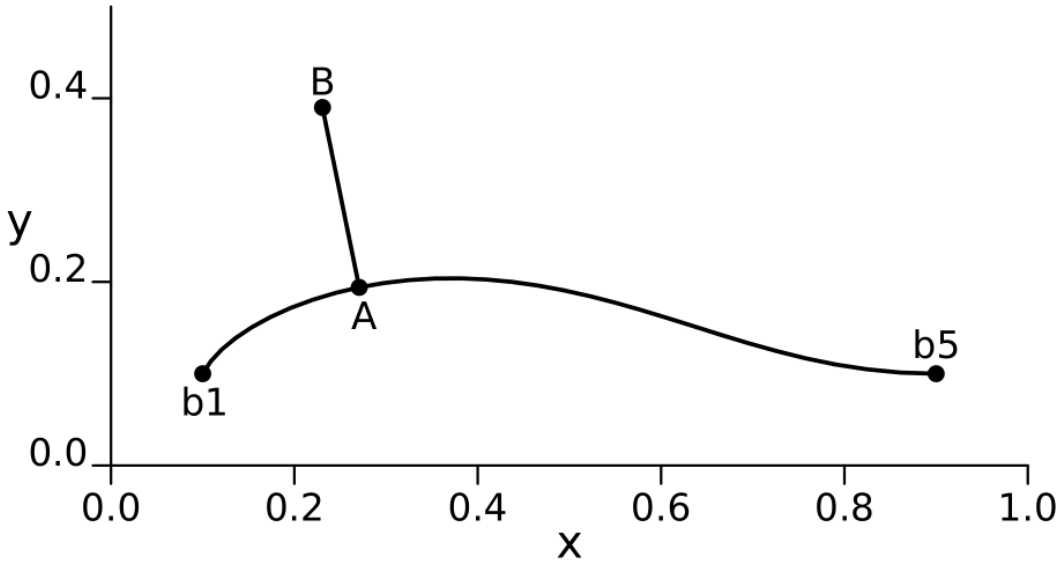



图 2.15:构造一个垂直于壁面的点。

# 3.网格

由于流动求解器期望将气体流域和固体域指定为有限体积单元网格，因此我们现在需要考虑二维块和三维体的离散。通常，我们将面和体对象以及聚集函数对象传递给 StructuredGrid 类的构造函数，并返回定义有限体积单元格顶点的网格点。该网格结合边界条件和初始气体状态(如主求解器用户指南[2]所述)，然后定义一个流体块。

作为一个鼓励性的例子，特别是对于 CFD 的 MECH4480 学生来说，可以考虑在一瓶 *James Boag's Premium*(一款啤酒)周围的区域构建一个二维网格。图 3.1 显示了瓶体侧卧时的最终块布置。您可以在 x=0 到 x=0.2 米的曲线上看到瓶子的轮廓。我们只模拟了上半平面，气体域是围绕瓶的周边区域。此外，我们将分阶段进行建模，从单个块定义有限的子区域开始。

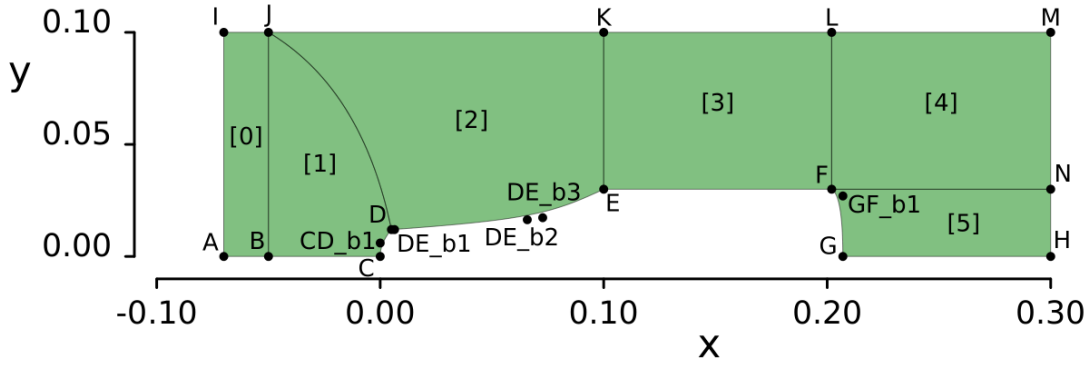



图 3.1:Ingo 啤酒瓶与 x 轴对齐的示意图。这个 PDF 图是从 SVG 文件生成的，经过一些编辑将点和块标签移动到最佳的位置。

## 3.1 制作简单的二维网格

我们从瓶子主体上方的块[3]开始，并定义 4 个节点，E,F,K 和 L 来标记感兴趣的点(如图 3.2)。定义区域的简单方法是创建一个 CoonsPatch 对象，它的四条边被指定为直线。StructuredGrid 构造器给出了这个平面对象、每个参数方向上的顶点数、可能还包括沿每条边的聚集函数列表。平面的参数方向 r 和 s 分别与网格索引方向 i 和 j 对齐。后面我们将重新回到聚集讨论。

```
1 -- the-minimal-grid.lua
2
3 -- Create the nodes that define key points for our geometry.
4 E = Vector3:new{x=0.1, y=0.03}; F = Vector3:new{x=0.202, y=0.03}
5 K = Vector3:new{x=0.1, y=0.1}; L = Vector3:new{x=0.202, y=0.1}
6
7 patch3 = CoonsPatch:new{north=Line:new{p0=K, p1=L},
8                         east=Line:new{p0=F, p1=L},
9                         south=Line:new{p0=E, p1=F},
10                        west=Line:new{p0=E, p1=K}}
11
12 grid3 = StructuredGrid:new{psurface=patch3, niv=21, njv=21}
13 grid3:write_to_vtk_file("the-minimal-grid.vtk")
14
15 dofile("sketch-minimal-grid.lua")
```

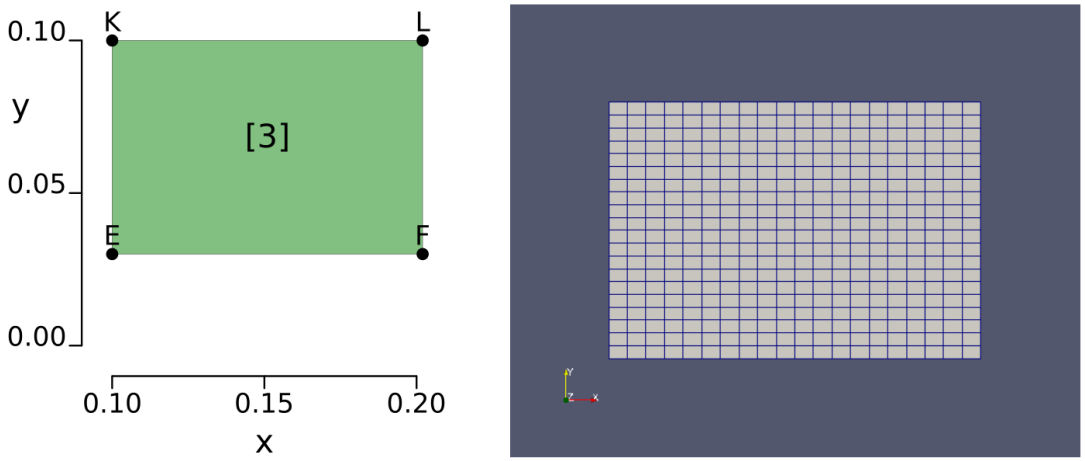



图 3.2: 啤酒瓶最终模型中一个简单分区域的单个块。

在前一个脚本的最后一行调用了用于制作 SVG 草图的脚本。它的内容有点晦涩，特别是在绘制坐标轴时，但它允许我们根据情况定制绘图。

```
1 -- sketch-minimal-grid.lua
2 -- Called by the user input script to make a sketch of the flow domain.
3 -- PJ, 2016-09-28
4 sk = Sketch:new{renderer="svg", projection="xyortho", canvas_mm={0.0,0.0,120.0,120.0}}
5 sk:set{viewport={0.0,-0.1,0.3,0.2}}
6 sk:start{file_name="the-minimal-grid.svg"}
7
8 -- for flow domain
9 sk:set{line_width=0.1, fill_colour="green"}
10 sk:render{surf=patch3}
11 sk:text{point=0.25 * (E+F+K+L), text="[3]", font_size=12}
12
```

```
13 -- labelled points of interest
14 sk:dotlabel{point=E, label="E"}; sk:dotlabel{point=F, label="F"}
15 sk:dotlabel{point=K, label="K"}; sk:dotlabel{point=L, label="L"}
16
17 -- axes
18 sk:set{line_width=0.3} -- for drawing rules
19 sk:rule{direction="x", vmin=0.1, vmax=0.2, vtic=0.05,
20          anchor_point=Vector3:new{x=0.1,y=-0.01},
21          tic_mark_size=0.004, number_format="%.2f",
22          text_offset=0.012, font_size=10}
23 sk:text{point=Vector3:new{x=0.15,y=-0.035}, text="x", font_size=12}
24 sk:rule{direction="y", vmin=0.0, vmax=0.1, vtic=0.05,
25          anchor_point=Vector3:new{x=0.09,y=0},
26          tic_mark_size=0.004, number_format="%.2f",
27          text_offset=0.005, font_size=10}
28 sk:text{point=Vector3:new{x=0.07,y=0.075}, text="y", font_size=12}
29
30 sk:finish{}
```

## 3.2 结构化网格类

StructuredGrid 类的构造函数接受下列任意一个项:

- path: 一个 Path 对象，沿着该对象定义若干离散点来表示一维网格。
- psurface: 一个 ParametricSurface 对象，在其上定义一个规则的点网格来表示二维网格的顶点。
- pvolume：一个 ParameterVolume 对象，通过它定义一个三维网格的点。

构造函数首先查找 path 项，如果没有找到，则尝试 psurface 项。如果都没有找到，它最后尝试查找 pvolume 项。一旦它找到一个有效的几何项，它就执行命令，忽略其它选项。定义网格的其它基本项是每个索引方向的顶点数，这些顶点被命名为 niv, njv 和 nkv。网格的索引方向 i, j, k 分别对应参数坐标方向 r, s, t。记住，每个索引方向上的有限体积单元格数比该方向上的顶点数少 1。我们通常会用单元格的数量来考虑网格的大小，因此会在构造函数调用中指定顶点的数量为 n+1，其中 n 是我们想要的单元格的数量。

在没有进一步信息的情况下，网格生成器将均匀地在每个参数方向上分布顶点。这可能对应也可能不对应于 x,y,z 空间中的均匀分布，这取决于所提供的几

何对象的定义。为了获得网格顶点分布的均匀性，例如在定义表面的边缘时，你可能需要使用 ArcLengthParameterizedPath(参见第 15 页)。

有时您确实想要沿着一个或多个指标方向的非均匀分布点。顶点沿几何对象的每条边的分布可以通过聚集函数 cluster-functionobject 指定。它们是 UnivariateFunction 对象，如下:

- LinearFunction:new{t0=$t_0$ , t1=$t_1$ }：其中 $t_{new} = t_0 \times (1 - t_{old}) + t_1 \times t_{old}$。而 $t_0$ 和 $t_1$ 的默认值分别是 0.0 和 1.0.
- RobertsFunction:new{end0=*false*, end1=*false*, beta=$\beta$}：其中的 end0, end1 整数标志表明我们希望往哪个方向聚集(可能是两个)。当 beta > 1.0 时，指定了聚集的强度，beta 值越小，聚集越强。例如，1.3 的值是相对较弱的聚集，而 1.01 的值则是相当强的。将参数空间两端的 end0 项和 end1 项分别设为 true(即 t = 0 和 t = 1)。
- LuaFnClustering:new{luaFnName='myClusterFn'}：其中 luaFnName 是指定聚集的用户定义函数名称。在调用这个聚集函数对象之前，用户定义的函数应该出现。用户的函数应该接受一个 float 参数并返回一个 float 值。用户定义函数的作用是提供一个从参数空间 t = 0→1 的均匀分布到某个非均匀分布 s=0→1 的映射。下面给出了一个用户定义函数的抛物线分布例子：

```
function myParabola(t)
    s = t * t
    return s
end
```

这可以通过创建 LuaFnClustering 对象来作为一个集群函数:

```
myClustering = LuaFnClustering:new{luaFnName='myParabola'}
```

要为一维网格指定单个聚集函数，请在表中提供一个 cf 项给 StructuredGrid 构造函数。要指定二维和三维网格所需的聚集函数对象，需要提供一个命名对象表作为 cfList 项。在二维中，边缘的名称是北、东、南和西。在三维中，12 条边的名称为 edge01, edge12, edge32, edge03, edge45, edge56, edge76, edge47, edge04, edge15, edge26 以及 edge37。请注意，三维名称使用顶点标签(参见图 2.11)来指示 r, s 或 t 参数的变化方式。请查看附录 A 中的调试多维数据集，以便更好地了

解这种安排。(您把立方体的边缘剪下来粘在一起了，对吧?)您不需要为每条边指定一个聚集函数。如果您不为特定的边指定一个函数，它将得到一个默认的线性函数，结果是一个均匀分布的点集。

因此，将所有这些放在一起，我们就得到了一个二维网格的构造函数模板:

StructuredGrid:new{psurface=*myPatch*, niv=$n_i$ , njv=$n_j$ ,

cfList={north=$cf_N$ , south=$cf_S$ , west=$cf_W$ , east=$cf_E$ } }

省略较长的聚集函数 cluster-function 列表，参数体的构造函数模板看起来更简单了，如下:

StructuredGrid:new{pvolume=*myVolume*, niv=$n_i$ , njv=$n_j$ , nkv=$n_k$ }

### 3.2.1 结构化网格方法

一旦有了 StructuredGrid 对象，即 sgrid，就可以调用许多方法，如下：

- sgrid:get_niv()：返回 i 方向上的顶点数。空括号表示不需要提供参数。
- sgrid:get_njv()：返回 j 方向上的顶点数。
- sgrid:get_nkv()：返回 k 方向上的顶点数。对于二维网格，它的值为 1
- sgrid:get_vtx(*i, j, k*)：返回 Vector3 对象，给出 x,y,z 空间中顶点的位置。如果没有提供 j 或 k，则假定值为零。每个方向的索引值从 0 开始，并以(但小于)该方向的顶点数为范围。这与通常的 Lua 约定(从 1 开始)不一致，但它与 D 代码实现一致的 StructuredGrid 类。
- sgrid:subgrid($i_0$ , $n_i$ , $j_0$ , $n_j$ , $k_0$ , $n_k$ )：返回一个 StructuredGrid 对象，该对象已被构造为原始网格的一个分段。在每个方向上，$i_0$ 指定子网格的第一个顶点，$n_i$ 计算出进一步的顶点数，直到子网格的结束。$n_i$ 实际上是子网格中 i 方向上的单元数。如果没有提供 $k_0$ 和 $n_k$ 的值，则默认为 0 和 1。$j_0$ 和 $n_j$ 也是如此。
- sgrid:write_to_vtk_file(*fileName*)：将网格以经典 VTK 格式写入指定的文件。这种格式是一种简单的 ASCII 文本格式，可以用任何文本编辑器查看，也可以用像 Paraview 这样的可视化程序显示。
- sgrid:write_to_gzip_file(*fileName*)：将网格写入 Eilmer4 使用的本机格式。它本质上是一个 gzip 压缩的文本文件，其格式在 D 代码文件 src/geom/sgrid.d

中的执行代码定义。它与经典的 VTK 格式非常相似。

- sgrid:joinGrid(*otherGrid, joinLocation*)：在 sgrid 上将 *otherGrid* 追加到 *joinLocation* 上。连接位置可以指定为 east、imax、north 或 jmax。
- sgrid:find nearest_cell_centre(*x, y, z*)：返回在中心最接近 *x*、*y*、*z* 点单元格的单个索引，以及从该点到单元格中心位置的距离。未提供的参数默认为零。

### 3.2.2 导入 Gridpro 网格

未绑定函数 importGridproGrid(*fileName*, *scale*)将读取一个完整的 Gridpro 网格文件，并返回一个 StructuredGrid 对象列表(即 Lua 表)。一个完整的 Gridpro 网格文件可能包含几个块，因此返回一个 Lua 表。在使用由 CAD 几何图形构建的 Gridpro 网格时要注意，这些图形的尺寸通常是毫米。对于这样一个文件，所需的 *scale* 值为 0.001，以便将坐标转换为米。*scale* 的默认值是 1。

## 3.3 非结构化网格类

这是对 Eilmer4 的一个新课程，我们没有很多使用它的经验。目前，构造 UnstructuredGrid 对象的选项有:

- 构造一个 StructuredGrid 对象；
- 导入 SU2 格式的网格，它可能来自 Pointwise 编写的文件；
- 使用 Heather 的论文，但目前这项工作还没完成；

如果您仔细查看 Eilmer 源代码附带的示例集，您就可以发现构建和导入非结构化网格的示例。

## 3.4 创建多块网格

当创建一个相当复杂的流域时，最好是构建一组的块，其中每个块大致是一个四边形，但边界路径与要建模的对象的曲线需要相互匹配。将整个气流区域划分为 6 个区块后得到的网格如图 3.3 所示。

```
-- the-plain-bottle.lua

-- Create the nodes that define key points for our geometry.
A = Vector3:new{x=-0.07, y=0.0}; B = Vector3:new{x=-0.05, y=0.0}
C = Vector3:new{x=0.0, y=0.0}; D = Vector3:new{x=0.005, y=0.012}
E = Vector3:new{x=0.1, y=0.03}; F = Vector3:new{x=0.202, y=0.03}
G = Vector3:new{x=0.207, y=0.0}; H = Vector3:new{x=0.3, y=0.0}
I = Vector3:new{x=-0.07, y=0.1}; J = Vector3:new{x=-0.05, y=0.1}
K = Vector3:new{x=0.1, y=0.1}; L = Vector3:new{x=0.202, y=0.1}
M = Vector3:new{x=0.3, y=0.1}; N = Vector3:new{x=0.3, y=0.03}

-- Some interior Bezier control points
CD_b1 = Vector3:new{x=0.0, y=0.006}
DJ_b1 = Vector3:new{x=-0.008, y=0.075}
GF_b1 = Vector3:new{x=0.207, y=0.027}
DE_b1 = Vector3:new{x=0.0064, y=0.012}
DE_b2 = Vector3:new{x=0.0658, y=0.0164}
DE_b3 = Vector3:new{x=0.0727, y=0.0173}

-- Now, we join our nodes to create lines that will be used to form our blocks.
-- lower boundary along x-axis
AB = Line:new{p0=A, p1=B}; BC = Line:new{p0=B, p1=C}
GH = Line:new{p0=G, p1=H}
CD = Bezier:new{points={C, CD_b1, D}} -- top of bottle
DE = Bezier:new{points={D, DE_b1, DE_b2, DE_b3, E}} -- neck of bottle
EF = Line:new{p0=E, p1=F} -- side of bottle
-- bottom of bottle
GF = ArcLengthParameterizedPath:new{
underlying_path=Bezier:new{points={G, GF_b1, F}}}
-- Upper boundary of domain
IJ = Line:new{p0=I, p1=J}; JK = Line:new{p0=J, p1=K}
KL = Line:new{p0=K, p1=L}; LM = Line:new{p0=L, p1=M}
-- Lines to divide the gas flow domain into blocks.
AI = Line:new{p0=A, p1=I}; BJ = Line:new{p0=B, p1=J}
DJ = Bezier:new{points={D, DJ_b1, J}}
JD = ReversedPath:new{underlying_path=DJ}; EK = Line:new{p0=E, p1=K}
FL = Line:new{p0=F, p1=L};
NM = Line:new{p0=N, p1=M}; HN = Line:new{p0=H, p1=N}
FN = Line:new{p0=F, p1=N}

-- Define the blocks, boundary conditions and set the discretisation.
n0 = 10; n1 = 4; n2 = 20; n3 = 20; n4 = 20; n5 = 12; n6 = 8

patch = {}
```

```
45 patch[0] = CoonsPatch:new{north=IJ, east=BJ, south=AB, west=AI}
46 patch[1] = CoonsPatch:new{north=JD, east=CD, south=BC, west=BJ}
47 patch[2] = CoonsPatch:new{north=JK, east=EK, south=DE, west=DJ}
48 patch[3] = CoonsPatch:new{north=KL, east=FL, south=EF, west=EK}
49 patch[4] = CoonsPatch:new{north=LM, east=NM, south=FN, west=FL}
50 patch[5] = CoonsPatch:new{north=FN, east=HN, south=GH, west=GF}
51
52 grid = {}
53 grid[0] = StructuredGrid:new{psurface=patch[0], niv=n1+1, njv=n0+1}
54 grid[1] = StructuredGrid:new{psurface=patch[1], niv=n2+1, njv=n0+1}
55 grid[2] = StructuredGrid:new{psurface=patch[2], niv=n3+1, njv=n2+1}
56 grid[3] = StructuredGrid:new{psurface=patch[3], niv=n4+1, njv=n2+1}
57 grid[4] = StructuredGrid:new{psurface=patch[4], niv=n5+1, njv=n2+1}
58 grid[5] = StructuredGrid:new{psurface=patch[5], niv=n5+1, njv=n6+1}
59
60 for ib = 0, 5 do
61 fileName = string.format("the-plain-bottle-blk-%d.vtk", ib)
62 grid[ib]:write_to_vtk_file(fileName)
63 end
64 dofile("sketch-plain-bottle.lua")
```

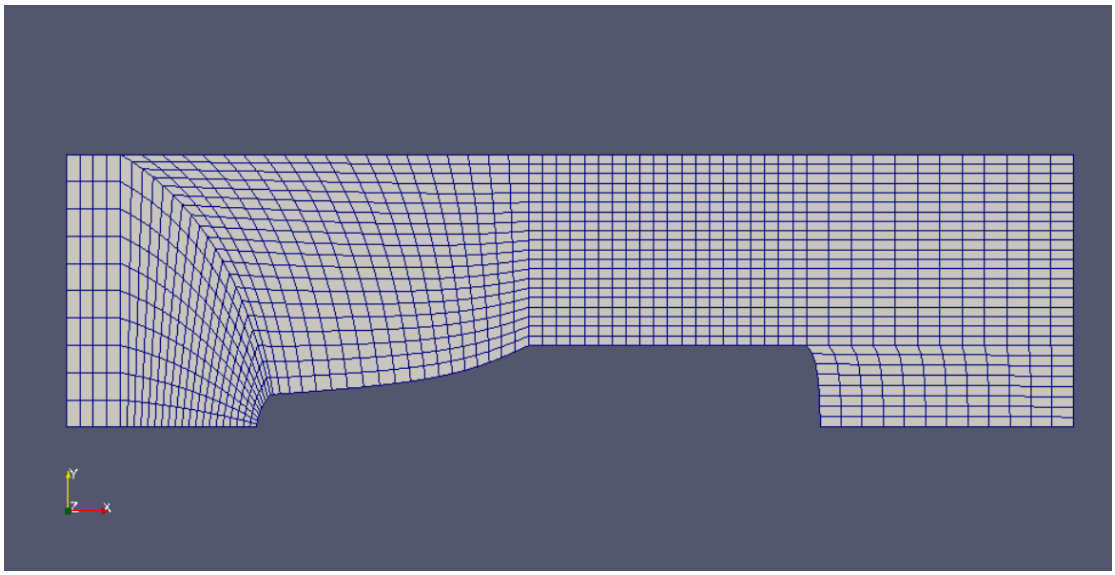


图 3.3:Ingo 啤酒瓶周围区域的多块网格模型。

每个块都是独立于其它块生成的。您需要做的是确保块共有的定义边是一致的，并且沿着这些边的单元离散化与另一个块的任何相邻边的相应离散化是一致的。通过只定义每条边一次，并在不同块的定义中重复使用那条路径，那么第一个约束就很容易满足。有时，一对块的方向和每个块内路径的特定方向意味着：一条定义边需要与原始边的意义相反。在这种情况下，ReversedPath 类可能很有用。对于这个示例，请注意实际上脚本使用第 36 行中的路径 DJ 为

JD 构造了这样一个路径。路径 JD 是定向的，可以作为 patch 1 的北边界;路径 DJ 是定向的，可以作为 patch 2 的西边界。

### 3.4.1 使用 clustering 提升网格

现在我们可以调整网格，并通过调整沿着每个块边缘的点的聚集程度来改善单元格的分布和形状。下面脚本的结果请参见图 3.4。用于度量聚集强度的特定值是特别的，需要通过一些尝试和错误来获得这些特定值。

同样，每个块的每条边上的点分布是独立计算的，因此用户有责任确保相邻块的相应边上的单元格对齐。这需要在这些边缘上使用匹配的聚集函数。

```
1 -- the-clustered-bottle.lua
2
3 -- Create the nodes that define key points for our geometry.
4 A = Vector3:new{x=-0.07, y=0.0}; B = Vector3:new{x=-0.05, y=0.0}
5 C = Vector3:new{x=0.0, y=0.0}; D = Vector3:new{x=0.005, y=0.012}
6 E = Vector3:new{x=0.1, y=0.03}; F = Vector3:new{x=0.202, y=0.03}
7 G = Vector3:new{x=0.207, y=0.0}; H = Vector3:new{x=0.3, y=0.0}
8 I = Vector3:new{x=-0.07, y=0.1}; J = Vector3:new{x=-0.05, y=0.1}
9 K = Vector3:new{x=0.1, y=0.1}; L = Vector3:new{x=0.202, y=0.1}
10 M = Vector3:new{x=0.3, y=0.1}; N = Vector3:new{x=0.3, y=0.03}
11
12 -- Some interior Bezier control points
13 CD_b1 = Vector3:new{x=0.0, y=0.006}
14 DJ_b1 = Vector3:new{x=-0.008, y=0.075}
15 GF_b1 = Vector3:new{x=0.207, y=0.027}
16 DE_b1 = Vector3:new{x=0.0064, y=0.012}
17 DE_b2 = Vector3:new{x=0.0658, y=0.0164}
18 DE_b3 = Vector3:new{x=0.0727, y=0.0173}
19
20 -- Now, we join our nodes to create lines that will be used to form our blocks.
21 -- lower boundary along x-axis
22 AB = Line:new{p0=A, p1=B}; BC = Line:new{p0=B, p1=C}
23 GH = Line:new{p0=G, p1=H}
24 CD = Bezier:new{points={C, CD_b1, D}} -- top of bottle
25 DE = Bezier:new{points={D, DE_b1, DE_b2, DE_b3, E}} -- neck of bottle
26 EF = Line:new{p0=E, p1=F} -- side of bottle
27 -- bottom of bottle
28 GF = ArcLengthParameterizedPath:new{
29 underlying_path=Bezier:new{points={G, GF_b1, F}}}
30 -- Upper boundary of domain
```

```
31 IJ = Line:new{p0=I, p1=J}; JK = Line:new{p0=J, p1=K}
32 KL = Line:new{p0=K, p1=L}; LM = Line:new{p0=L, p1=M}
33 -- Lines to divide the gas flow domain into blocks.
34 AI = Line:new{p0=A, p1=I}; BJ = Line:new{p0=B, p1=J}
35 DJ = Bezier:new{points={D, DJ_b1, J}}
36 JD = ReversedPath:new{underlying_path=DJ}; EK = Line:new{p0=E, p1=K}
37 FL = Line:new{p0=F, p1=L};
38 NM = Line:new{p0=N, p1=M}; HN = Line:new{p0=H, p1=N}
39 FN = Line:new{p0=F, p1=N}
40
41 -- Define the blocks, boundary conditions and set the discretisation.
42 n0 = 10; n1 = 4; n2 = 20; n3 = 20; n4 = 20; n5 = 12; n6 = 8
43
44 patch = {}
45 patch[0] = CoonsPatch:new{north=IJ, east=BJ, south=AB, west=AI}
46 patch[1] = CoonsPatch:new{north=JD, east=CD, south=BC, west=BJ}
47 patch[2] = CoonsPatch:new{north=JK, east=EK, south=DE, west=DJ}
48 patch[3] = CoonsPatch:new{north=KL, east=FL, south=EF, west=EK}
49 patch[4] = CoonsPatch:new{north=LM, east=NM, south=FN, west=FL}
50 patch[5] = CoonsPatch:new{north=FN, east=HN, south=GH, west=GF}
51
52 rcfL = RobertsFunction:new{end0=true, end1=false, beta=1.2}
53 rcfR = RobertsFunction:new{end0=false, end1=true, beta=1.2}
54
55 grid = {}
56 grid[0] = StructuredGrid:new{psurface=patch[0], niv=n1+1, njv=n0+1}
57 cfList = {north=RobertsFunction:new{end0=false, end1=true, beta=1.1}, south=rcfR}
58 grid[1] = StructuredGrid:new{psurface=patch[1], niv=n2+1, njv=n0+1,
59                              cfList=cfList}
60 cfList = {north=rcfR, west=RobertsFunction:new{end0=true, end1=false, beta=1.1}}
61 grid[2] = StructuredGrid:new{psurface=patch[2], niv=n3+1, njv=n2+1,
62                             cfList=cfList}
63 grid[3] = StructuredGrid:new{psurface=patch[3], niv=n4+1, njv=n2+1}
64 grid[4] = StructuredGrid:new{psurface=patch[4], niv=n5+1, njv=n2+1,
65                              cfList={north=rcfL, south=rcfL}}
66 grid[5] = StructuredGrid:new{psurface=patch[5], niv=n5+1, njv=n6+1,
67                              cfList={north=rcfL, south=rcfL}}
68
69 for ib = 0, 5 do
70 fileName = string.format("the-clustered-bottle-blk-%d.vtk", ib)
71 grid[ib]:write_to_vtk_file(fileName)
72 end
```

进一步改进网格可以通过在瓶子表面周围引入一层块，使靠近表面的单元格

始终接近正交，并更精细地向表面聚集。额外的块会增加输入脚本的复杂性，但沿块边缘提供了一些与单元格数量有关的解耦，并允许单元格朝着瓶表面精细化聚集，而不会大大增加气体流动区域其他部分的单元格细化。这种网格非常适合于粘性流动的模拟。

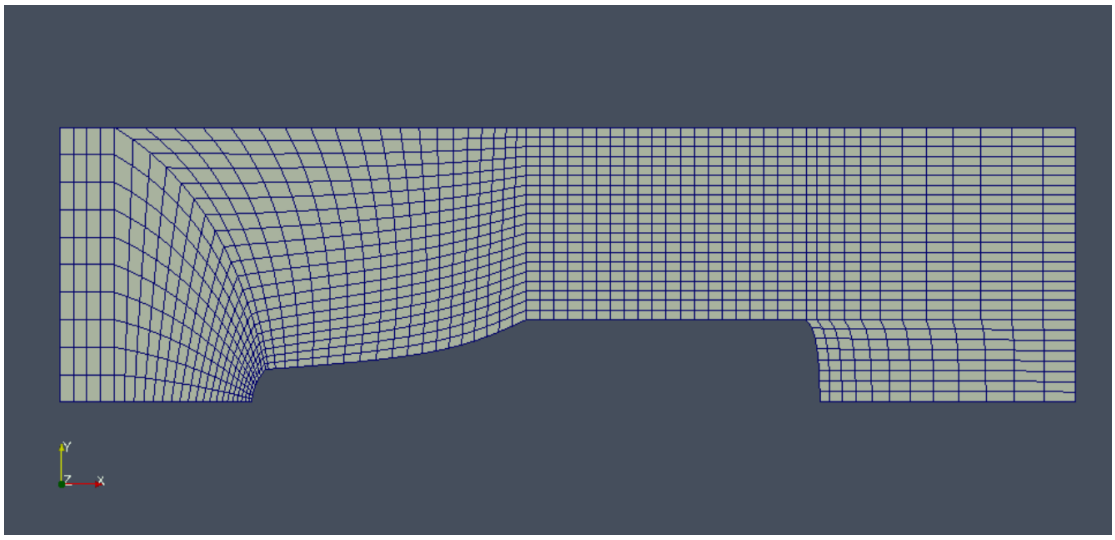


图 3.4:围绕着 Ingo 啤酒瓶的多个方块网格。

# 参考文献

[1]. Peter Jacobs and Rowan Gollan. Implementation of a compressible-flow simulation code in the D programming language. In Advances of Computational Mechanics in Australia, volume 846 of Applied Mechanics and Materials, pages 54–60. TransTech Publications, 9 2016.

[2]. Peter A. Jacobs and Rowan J. Gollan. The Eilmer 4.0 flow simulation program: Guide to the transient flow solver, including some examples to get you started. School of Mechanical and Mining Engineering Technical Report 2017/26, The University of Queensland, Brisbane, Australia, February 2018.

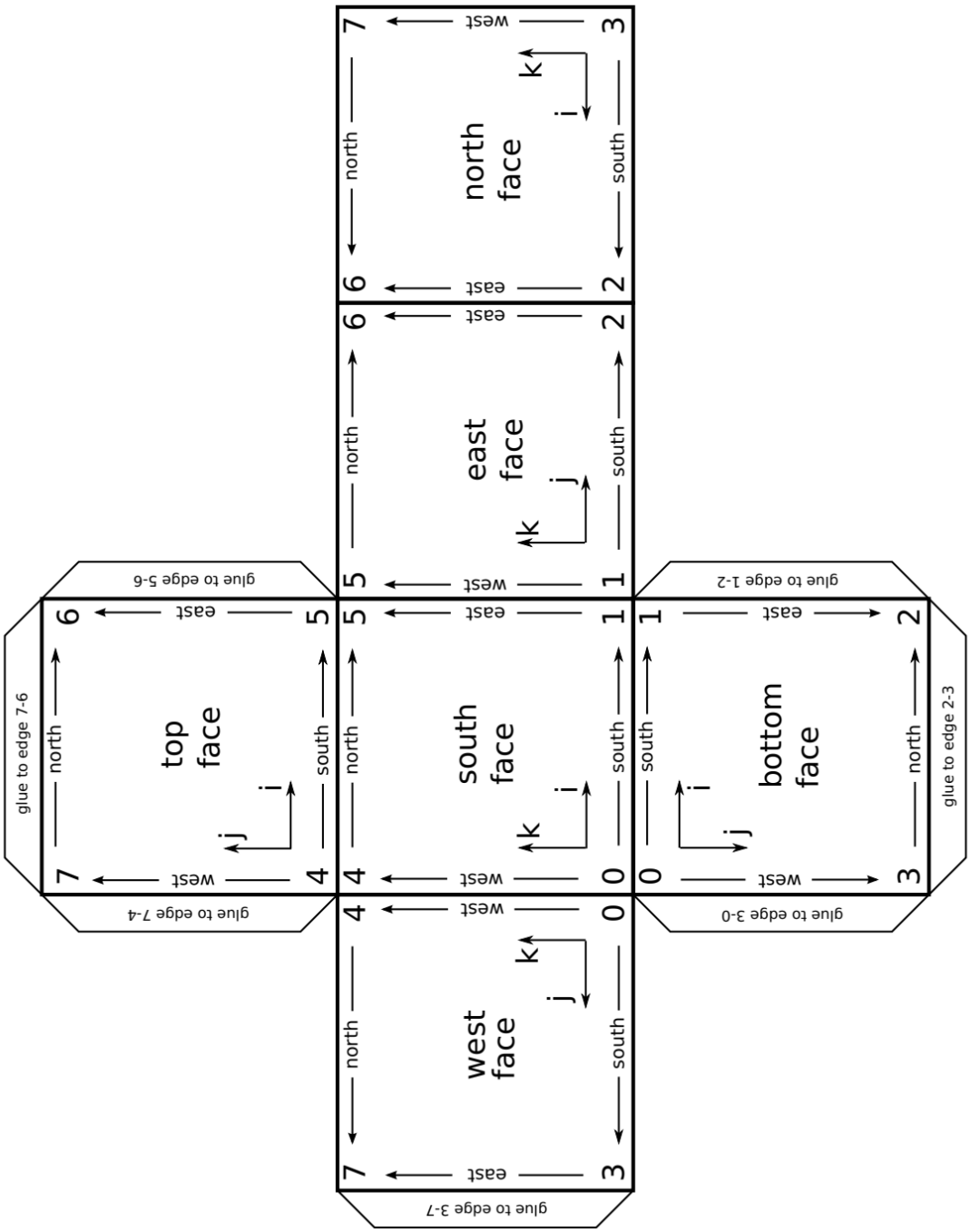
[3]. Roberto Ierusalimschy. Programming in Lua. Lua.org, 2006.

[4]. M. J. Zucrow and J. D. Hoffman. Gas Dynamics Volume 1. John Wiley & Sons, New York, 1976.

[5]. P. M. Knupp. A robust elliptic grid generator. Journal of Computational Physics, 100(2):409–418, 1992.

# A. 制作自己调试的多维数据集

从这一页的背面剪下图像，沿着所有的边缘折叠，然后把您自己的立方体粘在一起。一对立方体对于整理结构化网格块之间的连接规范非常方便。


north face
east face
south face
west face
top face
bottom face
glue to edge 7-6
glue to edge 5-6
glue to edge 1-2
glue to edge 3-0
glue to edge 7-4
glue to edge 3-7
glue to edge 2-3
north
south
east
west
i
j
k