**anisotropic refinement**

注：第一部分讲Step-30：各向异性加密；第二部分译者针对在各向异性加密中需要特别注意的网格等级的问题，结合早期Deal.II开发者在设计各向异性加密数据结构时的thesis，进行了有关说明。

这个例子是Step-12的延伸（DG方法求解线性输运方程），用于说明怎么进行各向异性的网格加密。包含如下内容：

1）各向异性加密：含义和动机

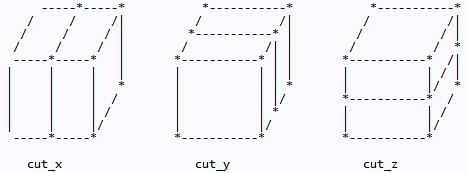
2）代码实现：怎样修改代码

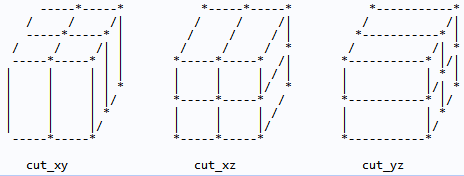
3）Jump指示器：用于DG中各向异性加密的指示器。

注：尽管这个程序是Step-12的修改版（对相同的方程使用相同的离散形式），但它是在MeshWorker模块出现之前就有了，所以它和现有的Step-12相似之处很少。

**各向异性加密**

所有可能的网格细分形式用枚举RefinementPossibilities::Possibilities来描述。





**代码实现**

有时候我们需要知道网格间的关系。这种关系来自两方面：邻居关系和母子关系。对于各项同性加密，deal.II使用了一些不变的规定。比如：一个细化的网格有2dim个子网格。且两个相邻网格最多只能相差一次细化等级：其中一个只能比另一个网格多加密一次，从而在公共边上最多有一个悬挂点。多数时候，对用户而言不需要深究这些底层细节。但为了某些应用，有时候就不得不深入了解下其实现机制。

在deal.II发展的早些时候，一个被细化的网格有多少子网格是个静态的信息2dim，但现在它依赖于网格的实际划分形式。可以用cell->n\_children()来获取实际的子网格数。之前使用的GeometryInfo<dim>::children\_per\_cell函数被替换为了GeometryInfo<dim>::max\_children\_per\_cell；GeometryInfo<dim>::subfaces\_per\_face被替换为了GeometryInfo<dim>::max\_children\_per\_face，或使用face->n\_children()。

另一个重要的方面是在组装网格间界面上的跳跃项时如何处理邻居关系。在Step-12的assemble\_system函数中我们知道，我们需要知道相邻网格相对当前网格而言，是稀疏的还是加密的或是同等level。但在各向异性加密情况下，之前采用的那种方法就不奏效了。因为在各向异性加密的情况下，仅仅靠网格单元携带的加密level信息是不够的，比如一个网格，先x向分割得到两个子cell，再对两个子cell进行y向分割，得到最终的子网格。如果认为最终的这些子cell是level2的，那用一次各向同性加密得到相同的子cell又是level1的。对于它们的邻居而言，如果认为这是按普通各向同性加密的话，自然判断这些子cell是level1。但如果采用各向异性这种level分类，那又会判断这些cell为level2。这就产生了矛盾：分明是同样的网格，按isotropic判断就是level1，按anisotropic判断就是level2。这肯定不对。笔者注：在deal.ii中应该仍然遵从后面那种判断，即认为是level2的，但又特别定义了一些函数来判断其真实的细化状态（而非仅仅依靠level来判断，尽管其level高，但可能是比当前网格稀疏的）。

**在各向异性加密后，一个稀疏的邻居并不一定是比我们所在的cell低一级（低级表示加密程度低），可能是任意级别的；实际上，甚至可能是比我们所在的cell密级更高，尽管它要稀一些。**因此怎样判断邻居的加密程度必须要采用某种不同的方式。

接下来，想要计算类似如下贡献项：C:\Users\zeng\AppData\Local\Temp\1510638968(1).png

别忘了我们会使用到FEFaceValues和FESubfaceValues来积分这样的项。我们会展示该怎么写代码，使得代码同时适用于各向同性和各项异性加密：

* 更密的邻居：

如果我们想要在一个active cell的face上计算这个积分，那么第一种可能是在当前face后的那个邻居更细化一些，即：它的孩子占据了当前公共边的一部分。这种情况下，当前face必须是细化了的，可通过询问:if(face->has\_children())。如果为真，我们需要在所有subfaces上循环，获取subface后邻居的孩子，进而reinit邻居的FEFaceValues对象和当前cell的不同subface的FESubfaceValues对象。

对于各向同性加密，这种情况比较简单，因为我们知道在各向同性加密时约定邻居只能与当前网格相差一个加密级别。然而，对于各向异性加密的网格这种约定就不成立了。尤其是对于3d情况，在我们感兴趣的face的后面的active cell可能不会恰好是当前网格的邻居的孩子，也可能是孙子甚至是更远的后代。所幸的是这种复杂性隐藏在库内部，库提供的函数同时适用于各向异性和各向同性的情况：**用户只需要调用cell->neighbor\_child\_on\_subface(face\_no，subface\_no)即可，它返回的是一个迭代器，指向当前网格在某个subface后相邻的那个网格（用face\_no和subface\_no指定是当前cell的哪个面及哪个子面）**。同样，在3D情况下有两种情况需要特殊考虑：

\* 邻居各项异性加密超过一次，这里要考虑的情况是有三个subfaces，而非两个或四个。比如下图表示正对我们的一个三维网格（邻居网格）的二维面（与我们所在的网格共享的面），可能是经过两次各向异性加密，得到了三个子面。



注意函数face->n\_children()与函数face->number\_of\_children()的细微区别。face->n\_children()返回的是直接孩子数量，对上面的例子就是2；face->number\_of\_children()返回的是active的所有后代数量，即3。使用face->number\_of\_children()对于2d，3d中的各向同性或各向异性情况都奏效，所以始终应该使用这个函数。

注意，在上面的最右的情况下，如果要继续细化靠左的两个细网格，则我们所处的网格就必须跟着细化了。因为必须要满足原则：每条边只能最多有一个悬挂点。

\* 存在一种情况：尽管邻居更稀疏，但它的孩子仍比我们当前的网格细密（理解这一说法的重点在于正确理解“邻居”的定义）。这种情况可能出现在当两个同样稀疏的网格被加密时，其中一个网格有两个孩子，而另一个有四个孩子。



如上图，左侧的两个cells源于母网格的一次在y方向的各向异性划分，而右侧的四个cells源于一次同时在y和z向的各向异性划分。左侧标记了#的网格有两个更密的邻居，标记为+，但是左侧网格的实际邻居是整个右侧母网格（邻居的加密级别不能高于当前网格，故而当你询问#网格的邻居时，你或许会认为其邻居是由那两个+网格构成的整体但实际上不是，因为你以为的这个网格实际上并不存在。你得到的邻居要么是同等level的要么是更稀疏的。）

所幸的是，函数cell->neighbor\_child\_on\_subface(face\_no,subface\_no)自己已经考虑到了这些情况。FESubfaceValues<dim>::reinit函数也考虑到了这些，故得到的结果总是正确的。

* 邻居我和一样密：

在排除了邻居有更细化的孩子的这种情况后，就只剩下两种可能，要么邻居跟当前网格一样粗细，要么比当前网格更稀疏。可使用函数cell->neighbor\_is\_ coarser(face\_no)返回的布尔值来得知。用法为：else if (!cell->neighbor\_is\_coarser(face\_no)). 只要cell->neighbor\_is\_coarser(face\_no)返回的为假，说明邻居不比我更稀疏，又由于我们定义了邻居的加密级别不能比我们更高（更密），故邻居只可能和当前网格一样密。

* 更稀疏的邻居：

最后剩下的情况就只有邻居更加稀疏了。函数cell->neighbor\_of\_coarser\_neighbor(face\_no)是函数cell->neighbor\_of\_ neighbor(face\_no)的推广。它返回一个数对，用以表明当前网格相对于邻居是其第几个面及子面上的（是邻居网格的第几个邻居）。

**网格光顺**有时候没被标记的网格也会被细化，这是为了满足网格光顺要求。也就是说相邻网格间的加密程度相差不能太大。我们约定，每条边上的悬挂格点最多只能有一个。

又如deal.II要求不能有单独被细化的网格（即其周围都是非细化的），因为这种情况下其上的自由度基本上都会被限制住（悬挂点收到约束）（参见Triangulation::MeshSmoothing）。

各向异性加密的光顺算法基本上由各项同性的光顺算法得来。有两个算法值得注意：

1）MeshSmoothing::limit\_level\_difference\_at\_vertices：在各向同性算法中，这个函数可以减小同一顶点周围的几个cells的level差。但在各向异性情况中没有对应的思想。所以在各向异性加密中不会用到这个函数。所以任何试图在已经各向异性加密的网格上调用这个函数都会抛出错误。

2）MeshSmoothing::allow\_anisotropic\_smoothing：网格光顺是为了限制悬挂点的数量，但没有必要为了减少一个悬挂点而把相邻的网格进行各向同性细化。可以使用各向异性的细化来实现光顺效果。所以对于各向异性加密，如果采用各向异性光顺，可减少多余网格。

**Jump indicator**

一般各向异性加密过程包括如下步骤：

1.计算误差指示器

2.利用误差指示器立flag用于加密。这些被标记的网格会自动被各向同性加密

3.在这些被标记的网格上计算另一个用于各向异性加密的指示器

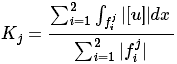
4.利用各向异性指示器立新的flag用于各向异性加密，其他flag不变

5.调用Triangulation<dim>::execute\_coarsening\_and\_refinement执行，同时会用到各向同性、异性的flag

上面的这种过程其实与hp自适应加密的流程基本相同。各向异性加密的网格只在已经标记了需要加密flag的网格上进行。

在这里我们使用的方法仅仅对DG方法适用。思想很简单：DG方法允许解在网格界面上有跳跃，尽管在网格内部解是连续的。当我们不断加密网格，我们预期的效果应该是跳跃趋于零，对解的近似效果越来越好。因此，**在跨越网格时越大的jump表示这个网格越该在正交于这个面的方向上进行加密。**（与应物所听高振勋报告时那个讨论一样）。当然，准确解也可能本身是不光滑的，有jump的。在那种情况下，跨过界面的大跳跃表示这个界面多少是平行于jump面的，所以也应该在正交于界面的方向上加密（从而加密线平行于界面）。

我们在这里提出的indicator计算平均跳跃，即：在两个面上u的跳跃值的绝对平均值，j表示某个坐标方向：



如果一个方向上的平均跳跃值比另一个方向上的大，且大于k倍，即C:\Users\zeng\AppData\Local\Temp\1510837080(1).png，那么就只在i方向上进行加密，否则双向加密。

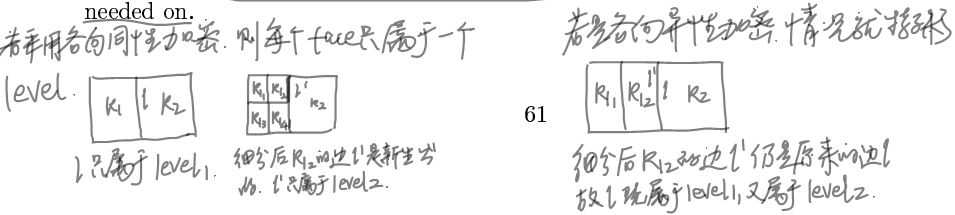
这种加密原则可以推广到system的情况：跳跃的绝对值应该替换成向量跳跃值的某种合适的范数。

**hierarchy of mesh**

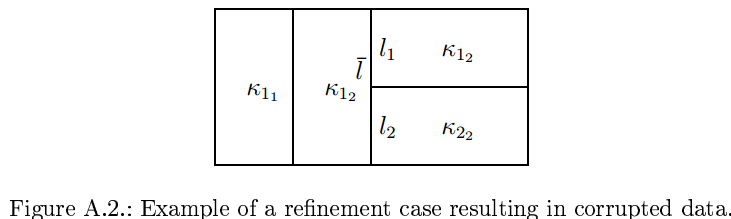
注：这部分内容取自Tobias Leicht的硕士论文Anisotropic Mesh Refinement for Discontinuous Galerkin Methods in Aerodynamic Flow Simulations, 2006.

在早期不涉及各向异性加密时的deal.II中，对3D网格而言，cell，face，edge在各向同性加密过程中，每加密一次，生成子cell同时也对应生成子face和子edge。所以这三者都是有level结构的。也就是说每个face或edge都只属于某一个level。只有顶点，既属于母cell，又属于子cell。所以顶点是以全局方式存储的，没有level结构。

但是如果引入各向异性加密，则可能出现某个face既属于子网格，又属于母网格。以下图为例：



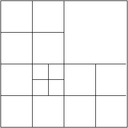
一种简单粗暴的处理方式是在每个level都造一个line的副本，从而使得line也具有level结构。但这又会有新的问题，比如我们想要再对k2进行各向异性加密：



如果k2细分为两个子cell，则对应有新的边l1和l2。但注意，这只是基于原先处于level n上的线；而处于level n+1上的副本是没有改变的。这就出现一个问题： 在加密后，线有孩子，但没有；但实际上它们代表了实际空间中的同一条线！为了不破坏数据完整性，这种加密方式是不被允许的。也就是说，如果采用这种做法，就不能允许在k2上这样的加密方式。为了解决这个问题，deal.II最终放弃了face，edge的level结构。所以在deal.II中face，edge，vertex都是全局保存的，不存在层次结构。

注：下面关于某个网格“neighbor”的描述取自手册Class中关于CellAccessor::neighbor(i)的描述

这个函数返回指向第i个邻居的迭代器。一个cell的neighbor最多具有与当前cell相同的level（不能比当前cell的更高）。比如：



如果我们处在右上那个网格上，问它它的左邻居是谁（按GeometryInfo中的惯例，这是第0号邻居），得到的是左上四个子网格的母网格。也就是说，得到的邻居的refinement level是和我们所在的cell的level相同，而且这个邻居有孩子。

另一方面，如果站在左上四子网格中的右上那个子网格上，问它它的右邻居（对应编号1）是谁，返回的是右上角那个大网格。右上大网格的level比当前我们所在网格的level低，且没有孩子。