现在我们知道并且懂得了基本方程,如何离散化它们并使用计算机对流体进行数值模拟呢?要做到这一点有很多选择,许多新方法正在研究.我们将无法覆盖该领域的十分之一,而是将重点放在一些完善的图形方法上.

**2.1 分割**

我们采取的第一个选择是分割：将复杂的方程式分解成各个组成部分，依次求解.如果我们说一个量的变化率是多个项的总和,我们可以数值求解每一项并最终将所有结果累加起来.

让我们通过一个非常简单的“玩具”示例(一个常微分方程)使这一点更加清楚：

您当然已经知道答案是,但是我们要使用基于分割的数值方法求解.我们将其分为两个步骤，每个步骤看起来像一个简单的前向欧拉更新(请参阅附录A):

用在这里的上标符号表示在时间步处所计算的值,是连续时间步长之间的时间量(这里的上标符号不要和指数弄混了,因为下标我们要用作网格的索引.在本书中,如果不作特别说明,上标普遍用作时间步索引). 我们要做的是将方程分为两个步骤：在第一步（2.1）之后，我们得到一个中间量，其中包括第一项（= 1）而不是第二项（= 2）的贡献;然后第二步（2.2）将中间值合并到最终结果.显然,在此示例中，我们得到了正确的答案,而拆分并没有给我们带来任何好处.

让我们将示例升级到更有趣的内容:

这里的和是一些黑盒函数，它们代表单独的软件模块.我们可以将前向欧拉再次分割：

简单的泰勒级数分析表明，如果您担心的话，这仍然是一阶准确的算法.如果不是，请忽略此：

咋一看,这种分割并没有比原始前向欧拉法提供更多的好多.这里也是我们比较复杂的地方.假设我们将和分割成不同的软件模块,并且每一个都有非常好的特殊数值方法求解简单方程

这正是分割的动机:我们可能无法轻松处理整个方程的复杂性,但它建立在我们能够简单处理的单独项之上的.假设特殊的积分算法为和.然后我们的分割方法是

如果和只是前向欧拉，则这与方程(2.4)和(2.5)完全相同。如果您进行泰勒级数分析，则可以证明这仍然有一阶准确的方法.

拆分实际上只是应用于微分方程的分而治之的原则:解决整个问题可能非常难,但是可以将其拆分为更容易解决的部分,然后合并结果.

**2.2 分割流体方程**

我们将对不可压缩流体方程使用分割.特别是,我们将对流部分,物体力(重力)部分和压强/不可压缩性部分分开.如果粘度很重要,我们也可以选择将其分离出来:请参阅第10章.

也就是说，我们将设计出解决这些简单方程的方法：

我们在对流方程中使用了一般量,是因为我们也关注其它变量,不仅仅是速度.

解决公式(2.8)中对流方程的算法我们命名为:在时间间隔内通过速度场对量进行对流运算.第三章将对此提供一些较好的方法.

对于公式(2.9)中的物体力,前向欧拉足够了.

对于公式(2.10)中的压强/不可压缩性部分,我们将要开发一个算法称为用于计算和应用正确的压强使得保持零散度并且同样满足固体边界条件.第五章将处理此内容.

我们在上一节中提到的重要前提条件/保证问题是，对流仅应在无散度的速度场中进行.当我们四处移动流体并希望体积守恒时，我们正在移动的速度场必须没有散度:我们已经在第1章中介绍了这一点.因此,我们要确保仅使用的输出运行 **()**:分割的顺序非常重要！

将所有东西组织在一起,得到我们基础的流体算法:

1. 以一个初始无散度的速度场开始.
2. 对于时间步
   1. 从时间到确定一个好的时间步.
   2. 设.
   3. 相加.
   4. 设.

2.3 时间步

确定合适的时间步长是算法的第一步.一个明显的问题是我们不想超出当前动画帧的持续时间：如果我们选择一个候选但发现，则应该限制并设置一个标志,以提醒我们已到达帧末.

受到这种限制后,我们应该选择一个满足模拟所有单独步骤(平流,体力等)的.我们将在随后的章节中进行讨论.通常,选择时间步长中的最小值是安全的(但并非在所有情况下都可以保证).

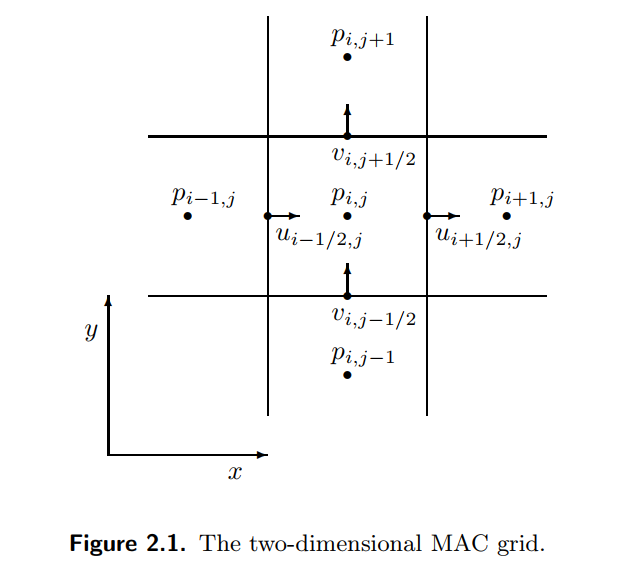
最后,对于仿真所需的质量,我们可能需要采取更小的时间步长来充分解决流体现象. 通常,这是基于计算事物在模拟中移动的速度,并通过相应地调整时间戳来限制它们可以一步走多远.例如,过长的时间步长可能意味着快速移动的固体的位置仅在其通过烟熏区域之前和之后进行采样,而这种采样不足会导致烟雾无法得到适当的干扰.限制时间步长,以便可以肯定地在烟熏区域的中间时间对固体进行采样,可以得到更好的质量结果.

但是,在某些情况下,我们可能会有性能要求,不允许我们在每一帧中采取很多小的时间步长.如果我们每帧只有三个时间步长,那么最好确保至少是帧时间的三分之一.这可能大于每个步骤建议的时间步长,因此我确保本书中所使用的方法都可以容忍使用超出期望的时间步长,在这种情况下,它们能够产生合理的结果,即使它们在数量上不准确.

**2.4 网格**

到目前为止,我们仅讨论了时间离散,而不是空间离散.虽然我们将在后续章节中对此进行更详细的介绍,但在这里我们将介绍基本的网格结构.

在计算流体动力学的早期,Harlow和Welch引入了**标记单元[marker-and-cell]**(MAC)方法[HW65]来解决不可压缩的流动问题.该论文的基本创新之一是一种新的网格结构(正如我们将在后面看到的那样),它为强制执行不可压缩性提供了非常有效的算法，尽管对于其它方面似乎都不方便.



所谓的MAC网格是交错的网格,即其中不同的变量存储在不同的位置.我们首先从二维角度来研究,如图2.1所示.压强在网格单元的中心采样,用表示.速度分割为两个笛卡尔分量.水平分量是在垂直单元格面的中心采样,例如用表示单元格和之间的水平速度.垂直分量在水平单元格面的中心采样，例如单元格和之间的垂直速度是.请注意,这意味着我们不在任何地方存储速度向量:速度的不同分量在不同的位置采样,不能简单地组合成一个向量.还要注意,对于网格单元,我们已经在其每个面的中心处采样了速度的法线分量：这很自然地使我们能够估计流入和流出单元的流体量.

在三个维度上,MAC网格的设置方式相同,压强位于网格单元中心,速度的三个不同分量被分割,因此我们在每个单元面的中心采样速度的法向分量(请参见图2.2).

我们将在第5章中更详细地说明为什么使用这种交错排列的方式,但简要地讲一下,以便我们可以将准确的中心差用于压力梯度和速度场的发散,而不会出现通常的中心差缺点. 仅考虑一维示例:在网格位置处采样并估算的导数.在网格点处无偏移计算,主要公式是一阶中心差分:

这是无偏且精确到,与前向或后向差分(例如)相反,例如

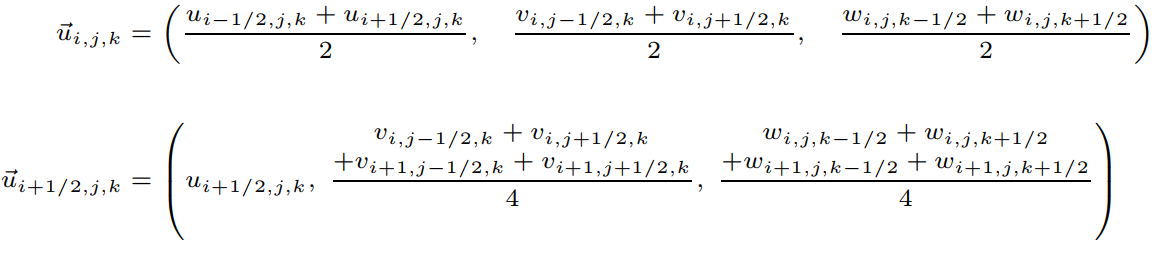
该函数向右偏移,仅精确到.但是,公式的主要问题在于,网格点的导数估算完全忽略了那里采样的值!要了解为什么如此糟糕,请回想一下,可以将常数函数定义为一阶导数为零的函数.如果我们要求有限差分为零，则允许不一定是恒定的——可能与和完全不同,但中心差分仍将报告导数为零,如只要满足.例如指数函数,它与常数相差很远，但符合公式的导数定义.另一方面,只有真正恒定的函数才能满足等于零的前向差分.公式的问题在技术上被称为具有非平凡的零空间:公式计算结果为零的函数集包含的内容不只是应限制于其的常数函数.

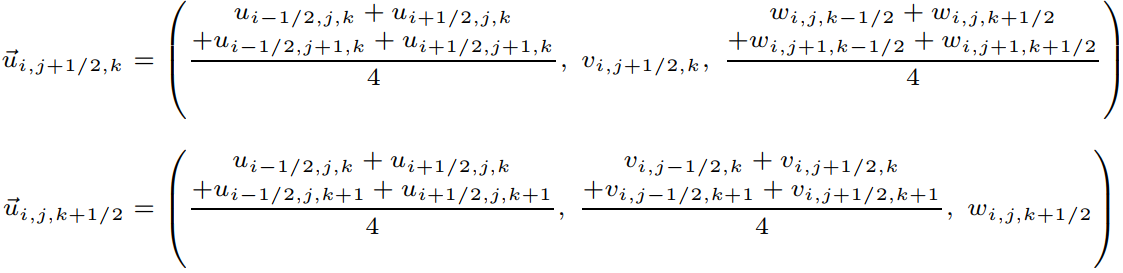
我们如何在避免零空间情况下获得无偏二阶精度中心差?答案是使用交错的网格:在中间点取值，.那么我们自然可以将网格点的导数估计为

这是无偏且精确到,但它不会跳过任何值.因此,如果将其设置为零,则只能为常数:零空间是正确的.设置了MAC网格,我们使用交错的中心差分形式能够在需要时估算压强解算导数(不可压缩条件).

交错的MAC网格非常适合处理压强和不可压缩性,但坦率地说,对于其他用途而言,这很痛苦.例如,我们想在某个地方估算整个速度向量,那么即使我们正在看一个网格点,也总是需要使用某种插值!在空间的任意位置,我们将对速度的每个分量分别进行双线性或三线性插值,但是由于这些分量彼此偏移,因此我们需要为每个分量计算一组不同的插值权重.在网格位置本身,可以归结为一些简单的平均.在二维中,这些平均值为

在三维空间上,公式相似:





最后,一个关于好奇的半指数的单词是有序的,例如.这些在理论上和概念上都很方便,可以固定采样点相对于网格的位置.但是，实现显然应该使用整数索引.需要一个标准约定,例如:

然后对于网格,压强存储在数组中,分量存储在数组中,分量存储在数组中,分量存储在数组中.

**2.5 动态稀疏网格**

对于基本的流体求解器,尤其是如果流体填充了其大部分边界框,则上述基本3D网格可以正常工作,并且是实现和使用的最简单的结构.但是,总体上存在三个问题.

第一个是最容易处理的：如果在整个模拟过程中流体占用的区域发生了显着变化（例如，从最初干燥的场景中注水），则采用覆盖整个区域的单个静态网格可能会看到流体,但是非常浪费。显而易见的解决方案是在每个时间步长动态调整网格尺寸及其在空间中的位置，以覆盖感兴趣的区域。对于液体，这可能只是液体先前状态的边界框，由流速场乘以时间步长进行调整，以估计其可移动的位置，再加上至少几个网格单元的安全性（并避免尴尬的网格边界条件影响水）。对于烟雾来说，由于模拟的流体（空气）整体上扩散到大气中，可能会更加棘手：在这种情况下，诸如烟雾浓度明显为零的区域周围的最小填充距离之类的标准可以很好地发挥作用。

第二个问题就是现代硬件的效率。 点和在内存中将是nx×ny个彼此远离的元素，因此，我们在网格上进行的大多数常见操作都不会像以下那样具有良好的数据局部性： 我们可以预期到的页面错误和高速缓存未命中的数量将超过严格必要的数量。

第三个也是最严重的问题是，当流体仅占据其边界盒体积的一小部分时，会浪费多少内存和处理空间。 这样的例子比比皆是：弯曲的河流，从倾斜的玻璃杯上溅出的液体，水池上方的瀑布，飞机弯曲轨迹上稀薄的烟迹，不平坦表面上的小火焰。

最后两个（包括第一个）的解决方案是使用稀疏的受阻网格[Bri03，LKHW03，MCZ07，Mus11，Mus13]。从概念上讲，我们从一个非常大的虚拟网格开始，例如沿每个维度为232，并在每个坐标中由带符号的32位整数进行索引（以便于使用负索引和正索引）。该虚拟网格足够大，可以覆盖我们进行的任何流体模拟，而无需调整其原点或尺寸。然后，我们将网格划分为固定大小的块或图块，每个块或图块的尺寸为6×6×6，这些块或图块存储为微小的连续3D阵列。最后，我们仅在给定时间步长中分配和存储所需的块，并存储在从索引映射到存储块的关联数据结构中。由于我们仅存储我们关注的网格块，因此这是一个稀疏结构，很好地解决了第一个和第三个问题：我们不会在动作之外浪费体素存储或处理。因为我们映射体素块而不是单个体素，所以可以最小化关联数据结构的开销。同时，块内的操作具有非常好的数据局部性，从而解决了第二个问题。

我最初的稀疏块状网格[Bri02]实现使用简单的两级层次结构，即指向精细块的3D“粗”指针数组，但后来的工作表明，将其扩展到三级或四级树更为有效[ Bri03，Mus13]。 另外，为避免某些树结构开销并可能允许更大的索引空间，您可以使用哈希表存储对精细块的引用[Bri03]，但缺点是并行更新哈希表要麻烦得多。

围绕稀疏的块状网格编写代码比使用密集的网格要难得多：用简单的索引，而不是整个域只有三个嵌套循环和简单的索引编制，还需要在块上有一个外部循环，并需要更多的逻辑来访问相邻的体素（这可能是或完全不存在）。另一个微妙的困难是MAC网格的错位：例如，用于u组件的块必须具有与任何其他组件相同的尺寸，因此，通过在烟熏模拟中增加额外程度的对称性，可能会稍微破坏对称性u在块的负面方面的自由度要比在积极方面大。补救措施是忽略（尽管存储）在没有负x相邻块的任何块的负侧的u样本的额外层，依此类推。由于此类复杂问题，我强烈建议您在第一个流体求解器中使用密集的3D阵列，并（与本书中的所有内容一样）在2D中进行稀疏网格模拟的原型，然后再进行3D建模。

**2.6 二维模拟**

最后一点有必要重复和强调,并在必要时多次重复.始终以2D形式制作3D解算原型. 始终以2D形式制作3D解算原型.即使您使用了有效的3D解算器,也要保持2D原型的最新状态,以便在开发新功能或旧错误时可以轻松地跳回它.

即使最终我们只关心3D仿真结果，所以认为在2D中实现算法似乎有点浪费时间，有时会有些尴尬，因为3D图形的内容创建工具通常不会有2D模式-我十年来在许多代码库，项目，机构和公司中的经验是，这可以极大地提高生产率。 流体求解器很少在您第一次编写时就开始工作，即使在达到可接受的结果后，偶尔会破坏您的仿真的罕见错误也可能会持续很长时间。 毕竟，我们要面对一些非常棘手的尚未解决的问题。 期望花费大量时间来追踪奇怪的工件，NaN，无法收敛的迭代等，并考虑围绕该现实优化您的开发时间.

2D解算器需要的代码比3D少一些，运行速度至少快一个数量级，更优雅地扩展至高分辨率，偶尔避免碰到某些几何算法的“维数诅咒”（例如，稳健地发现某个点是否在内部 2D多边形比查找点是否在3D网格内要简单得多，并且通常包含更少的索引和更简单的数学公式-所有这些使编程和调试更加高效。 最重要的是，在2D中可视化和分析结果要比在3D中容易得多：一张图像甚至数字电子表格都可以立即准确地告诉您2D速度场中正在发生的事情，但是尝试在3D中查看同一件事是一个主要问题 科学可视化本身存在的问题。 同时，从2D中学到的大部分经验都可以延续到3D，因此在跳到3D时，在2D求解器中进行工作所花费的时间和精力总是可以无数次地得到恢复。