三维精确 power 图的 GPU 并行计算

肖艳阳[1)], 李渭[2)], 徐少平[1)]

[1)] (南昌大学数学与计算机学院　南昌　330031)
[2)] (南昌大学软件学院　南昌　330031)
(xiaoyanyang@ncu.edu.cn)

摘　要: power 图(加权 Voronoi 图)的计算是计算机图形学和计算几何等领域的一项基础任务. 针对求解三维 power 图的传统串行方法所需的时间成本较高, 且现有并行算法所得结果为近似解, 提出一种新颖的 GPU 并行计算方法. 首先给出 power 图与高一维受限 Voronoi 图的等价构造方法, 将 Voronoi 图的无网格方法直接推广到 power 图的计算. 因此, 给定的加权种子点被置于更高一维空间中的一组方格内, 据此快速搜索每个种子点的邻居关系, 进而使用各个种子点与其若干个邻居的中垂面对各自的power胞元进行并行裁剪, 以快速地获取三维空间中的精确power图. 对比不同求解域下和 5 万个种子点的计算耗时, 比现有方法具有超过 3 倍的加速比.

关键词: power 图; 加权 Voronoi 图; 精确计算; 并行计算; GPU
中图法分类号: TP391.41　　**DOI:** 10.3724/SP.J.1089.2023.2023-00018

Parallel Computation of 3D Accurate Power Diagrams on GPU

Xiao Yanyang[1)], Li Wei[2)], and Xu Shaoping[1)]

[1)] (*School of Mathematics and Computer Sciences, Nanchang University, Nanchang*　330031)
[2)] (*School of Software, Nanchang University, Nanchang*　330031)

Abstract: The computation of power diagram (or weighted Voronoi diagram) is a fundamental task in computer graphics and computational geometry etc. Given the high time cost of serial methods and non-accurate results generated by parallel methods, a novel parallel computing method on GPU is proposed in this paper. We first give the equivalent construction from power diagrams to restricted Voronoi diagrams in one dimension higher, allowing us to extend the meshless method for generating Voronoi diagrams to the computation of power diagrams. Based on which, the given weighted sites are placed into a set of grids in the space with one dimension higher, from where the neighbors of each site can be identified quickly, then the power cells are clipped in parallel by using the bisectors between the corresponding site and the neighbors, which can efficiently obtain accurate power diagrams in the 3D space. By comparing the time cost under different domains and 50 thousand sites, the proposed method possesses more than three times the acceleration ratio with existing methods.

Key words: power diagram; weighted Voronoi diagram; exact computation; parallel computation; GPU

Voronoi 图[1]是一种常见的区域划分结构, 通常由一组种子点和欧几里得距离度量定义. 每个种子点对应空间中的一个子区域, 即 Voronoi 胞元, 其内的任意一点与该种子点的距离都小于它到其

收稿日期: 2023-04-15; 修回日期: 2023-10-21. **基金项目**: 国家自然科学基金(62102174, 62162043); 江西省自然科学基金(20212BAB212012). **肖艳阳**(1991—), 男, 博士, 讲师, 主要研究方向为计算机图形学、计算几何; **李渭**(1986—), 男, 博士, 讲师, 主要研究方向为计算机图形学、点云处理; **徐少平**(1976—), 男, 博士, 教授, 博士生导师, 主要研究方向为计算机图形学、图像处理、人工智能等.

他种子点的距离. 使用不同的距离度量, 或将种子点替换为其他的几何对象, 即可定义广义的 Voronoi 图. power 图(也被称为加权 Voronoi 图)[2]即是其中一种, 目前已被应用于如超像素分割[3-4]、采样[5-6]、网格生成[7-8]、自适应重网格化[9-10]和流体仿真[11-12]等多个方向. 因此, power 图的计算是一项基础任务, 提升其效率对于许多下游应用具有重要意义.

由于每个种子点被附加了一个被称作权重的实数, 因此 power 图的计算比 Voronoi 图的计算更加复杂. 虽然已有很多 Voronoi 图的计算方法[13-20], 但是它们一般无法被直接推广到 power 图的计算问题中. 目前, 适用于任意 power 图的精确计算方法仍然较少. Aurenhammer[2]提出将加权种子点映射到更高一维空间的超平面, 将高维空间裁剪成一个多面体, 其下表面被映射回原始空间中得到 power 图结果. 另一种更加常用的方法是基于 power 图与正则三角网格(regular triangulation, RT)[21]的对偶性, 首先为加权种子点构造一个正则三角网格, 每个种子点即可显式地获取各自的邻接关系, 则其 power 胞元可依次用对应种子点与其各个邻居的加权中垂面对空间进行裁剪来获取. 然而, 该方法在计算高维 power 图时所需的时间成本仍然较高. 这是由于构建高维的正则三角网格较为复杂, 特别是当种子点数量较多时, 这一步骤会显著地增加耗时.

为解决三维 power 图计算效率较低的问题, 一些并行计算方法被陆续提出. 桂志强等[22]提出了一种基于跳转泛洪算法(jump flooding algorithm, JFA)的 GPU 并行方法, 其设定固定分辨率的三维纹理供 JFA 访问, 并迭代更新每个像素的最近种子点, 因此该方法产生的每个胞元实际上是一组像素的聚类, 所得 power 图是一个近似的结果. 虽然提高纹理的分辨率可以适当降低近似误差, 但是这也会显著地增加算法的时间开销. 近年来, 在 Voronoi 图计算的研究中, 一类被称为无网格的方法受到许多关注[18-20]. 因该方法具有高度的并行特性, 其 GPU 并行版本也已被相继提出[19-20]. 然而, 这些并行方法均无法保证产生的 Voronoi 图为精确的结果. Zhai 等[12]和 Basselin 等[23]采用类似的策略将无网格方法推广用于 power 图的 GPU 并行计算中, 但其推广对加权种子点具有很强的约束, 即相邻种子点的权重必须非常接近, 显然无法适用于一般 power 图的求解.

由于 power 图的距离度量在权重维度上并非是递增的, 因此若直接将上述无网格方法应用于一般 power 图的计算, 在对与其对应种子点有关的加权中垂面进行排序时, 需要遍历所有其他的种子点, 否则无法保证结果的正确性. 为此, 本文提出一种简单的映射方法, 用于构造与原 power 图等价的高一维受限 Voronoi 图, 使得 Voronoi 图的计算方法能够用于 power 图的求解. 在此基础上, 结合一种改进的无网格方法, 本文提出一种 GPU 并行计算方法, 能够快速地获取精确的三维 power 图结果, 且其适用于一般 power 图的计算问题. 在介绍本文方法之前, 先简单回顾 Voronoi 图的无网格计算方法.

1　Voronoi 图的无网格计算方法

对于一组给定的种子点, 每个种子点对应的 Voronoi 胞元是该种子点与其他所有种子点之间半空间的交集. 基于此, 无网格方法在计算某个种子点的胞元时, 需要先对其他种子点按照离该种子点的距离由近及远地排序, 然后按顺序用该种子点与其邻居的中垂面对空间进行裁剪. 实际上, 该裁剪过程在满足特定条件时可以提前结束, 即种子点到某个邻居的距离大于该种子点到其胞元最远距离的 2 倍. 因此, 在这类方法的具体实现中, 只需要找出每个种子点最近的若干个邻居即可. 图 1 给出了该方法生成胞元的过程示意图. 其中, 除目标种子点(红色)外, 其他种子点(绿色)按照到目标种子点的距离排序, 由近及远地用目标种子点与其各个邻居的中垂面(虚线) 对胞元进行裁剪, 直至达到结束条件, 即红色种子点到其与 $\boldsymbol{v}_6$ 的中垂面(棕色虚线)距离大于其到胞元的最远距离.

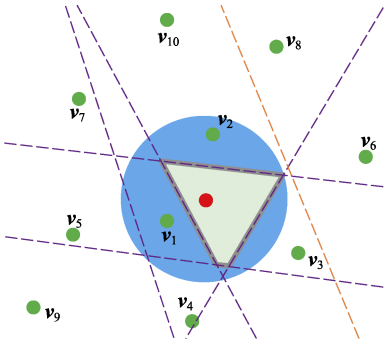


图 1　无网格方法生成 Voronoi 胞元的过程示意图

需要注意的是, 获取各个胞元的精确结果所需的邻居数量并不相等. 然而, 基于 GPU 的特性, 在现有的并行计算方法中, 每个种子点的邻居数

量都设定为相等的数值. 为保证获取 Voronoi 图的精确结果, 邻居数量往往需要设定为较大的数值, 这对于显卡内存容量的要求较高. 因此, 现有方法通常会设置一个折中的邻居数量, 以平衡精度和硬件要求. 因此现有 GPU 方法无法保证产生的 Voronoi 图是精确结果, 即部分胞元的体积可能大于其精确结果的体积, Voronoi 图结果的精确程度取决于种子点的分布情况和邻居数量.

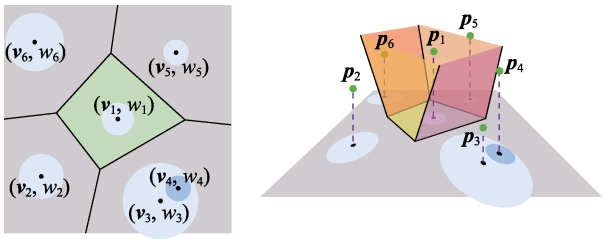
本文采用一种基于方格的邻居搜索和胞元裁剪策略, 提出的 GPU 并行计算方法可获得 Voronoi 图的精确结果. 为使该方法能够适用于 power 图的计算, 本文引入一种映射方法, 在高一维空间中构造与 power 图等价的受限 Voronoi 图.

2 power 图的等价受限 Voronoi 图

在 d 维空间 $\mathbb{R}^d$ 中, 给定 n 个种子点 $\{\boldsymbol{v}_i\}_{i=1}^n$, 每个种子点都有一个权重 $w_i, i=1,2,\cdots,n$, 如图 2a 所示 power 图将空间 $\mathbb{R}^d$ 划分成一组胞元 $\{\Omega_i\}_{i=1}^n$. 其中,

$$\Omega_i=\left\{\boldsymbol{x}\in\mathbb{R}^d \,\middle|\, D(\boldsymbol{x},\boldsymbol{v}_i,w_i)\leqslant D(\boldsymbol{x},\boldsymbol{v}_j,w_j),\forall j\neq i\right\} \quad (1)$$

且 $D(\boldsymbol{x},\boldsymbol{v},w)=\|\boldsymbol{x}-\boldsymbol{v}\|^2-w$ 为距离度量函数.



a. power图 b. 等价的高一维受限Voronoi图

图 2 power 图与高一维受限 Voronoi 图的等价性示意图

函数 D 在权重维度上非单调递增的问题可以通过引入最大权重的方式解决. 将最大权重标记为 $w_{\max}$, 即 $w_{\max}=\max\{w_1,w_2,\cdots,w_n\}$. 将其作为常数引入到距离函数中, 即

$$D(\boldsymbol{x},\boldsymbol{v},w)=\|\boldsymbol{x}-\boldsymbol{v}\|^2+w_{\max}-w \quad (2)$$

这并不会使式(1)不等式关系发生改变. 因此, 采用式(2)作为距离度量所定义的区域划分结果与原 power 图一致. 此外, 由于 $w_{\max}-w_i\geqslant 0,\forall i$ 成立, 式(2)可作变换

$$D(\boldsymbol{x},\boldsymbol{v},w)=\|\boldsymbol{x}-\boldsymbol{v}\|^2+\left(0-\sqrt{w_{\max}-w}\right)^2=\|\boldsymbol{y}-\boldsymbol{p}\|^2.$$

其中, $\boldsymbol{y}=(\boldsymbol{x},0)$; $\boldsymbol{p}=\left(\boldsymbol{v},\sqrt{w_{\max}-w}\right)$. $\boldsymbol{p}\in\mathbb{R}^{d+1}$ 是种子点 $\boldsymbol{v}$ 在高一维空间中的映射点. 因此, power 胞元可等价定义为

$$\Omega_i=\left\{\boldsymbol{x}\in\mathbb{R}^d \,\middle|\, \|\boldsymbol{y}-\boldsymbol{p}_i\|^2\leqslant\|\boldsymbol{y}-\boldsymbol{p}_j\|^2,\forall j\neq i\right\}.$$

即点 $\boldsymbol{p}_i$ 的 $(d+1)$ 维 Voronoi 胞元与空间 $\mathbb{R}^d$ 的交集(也被称为受限 Voronoi 胞元). 图 2 以二维情况为例, 给出了两者的等价性示意图.

至此, 本文给出了一种简单的将加权种子点映射到高一维空间中的方法, 使得每个 power 胞元都可以直接通过计算对应映射点的高一维受限 Voronoi 胞元来获得.

3 精确 power 图的并行计算

本文聚焦于三维精确 power 图的并行计算, 输入是被四面体化的求解域 $M=\{\tau_k\}_{k=1}^m$ 和一组加权种子点 $V=\{(\boldsymbol{v}_i,w_i)\}_{i=1}^n$, 输出是每个种子点的 power 胞元 $\{\Omega_i\}_{i=1}^n$ 与求解域的交集.

上述等价映射过程使得现有计算 Voronoi 图的 GPU 并行方法可以直接应用到 power 图的计算中. 然而, 正如前文所述, 这些方法无法保证产生的结果是精确的. 究其原因, 是这些方法只能提供固定数量的最近邻查询. 具体体现在 2 个方面: 一是每个种子点的邻居数量是固定的, 可能导致某些胞元比精确胞元体积更大; 二是求解域每个四面体的最近种子点数量是固定的, 导致无法找到更多与某些狭长四面体有交集的胞元.

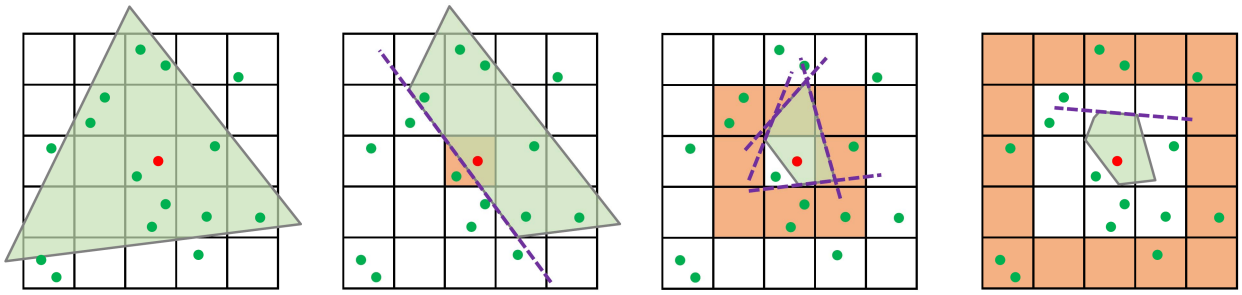
为快速地获取精确的结果, 本文方法采用一种基于方格的遍历策略, 能够妥善解决现有方法的上述 2 个问题.

3.1 power 胞元的构造

根据前文所述, 将给定的加权种子点提升到四维空间中得到点集 $\left\{\boldsymbol{p}_i=\left(\boldsymbol{v}_i,\sqrt{w_{\max}-w_i}\right)\right\}_{i=1}^n$, 下面只需以它们作为 Voronoi 种子点构造各自的胞元即可. 与无网格方法的思想类似, 本文也是通过搜索每个种子点的若干个最近邻居, 依次用它们与该种子点的中垂面对胞元进行裁剪. 区别在于, 无网格方法要求邻居的先后顺序必须严格按照各自离该种子点的距离排序, 而本文方法并不要求这种严格的排序, 因此两者的裁剪结束条件也不同.

首先, 在四维空间中开辟一组方格, 要求能够将每个映射点 $\boldsymbol{p}_i$ 都置于其中某个方格内. 根据每个方格内存储点的数量平均为 k_g 来设定方格的分辨率大小. 这样, 对于某个映射点而言, 可以通过其所在方格并由内而外地遍历周围方格, 以快速搜索出与其相邻的其他映射点.

接下来, 每个 power 胞元 Ω_i 可以通过下述过程独立地构造出来. 以求解域的包围盒作为 Ω_i 的初始值, 以其映射点 $\boldsymbol{p}_i$ 所在的方格 g_i 为中心, 由内向外一环一环地访问 g_i 及其邻近方格, 遍历其中包含的其他映射点, 利用它们与 $\boldsymbol{p}_i$ 的四维中垂面对 Ω_i 进行裁剪, 并在满足终止条件时提前结束. 该过程的示意图如图 3 所示(注意在实际算法中, 方格是四维的, 胞元是三维的).



a. 初始状态　b. 0 环裁剪　c. 1 环裁剪　d. 2 环裁剪

图 3　基于方格的胞元裁剪过程示意图

裁剪过程的终止条件同样与方格绑定. 这里维护 2 个距离, 一是 $\boldsymbol{p}_i$ 到 Ω_i 的最远距离 $d_{\max}$, 每次 Ω_i 被裁剪之后都需要立即更新 $d_{\max}$; 二是 $\boldsymbol{p}_i$ 到即将访问的某一环目标方格围成的内边界的最小距离 r_g, 当环数为 0 时, $r_g=0$. 因此, 本文将胞元裁剪的结束条件设置为 $r_g \geqslant 2d_{\max}$, 这表明即将访问的某一环目标方格内的所有映射点到 $\boldsymbol{p}_i$ 的距离一定大于 $2d_{\max}$, 后续的裁剪必然不会对 Ω_i 产生影响, 可以提前结束裁剪过程. 实际上, 每次 Ω_i 有更新时都判断结束条件是否达到, 以防止产生过多的无效裁剪.

这种胞元的构造过程所得的结果是精确的, 具体的并行构造算法步骤如下.

算法 1. power 图的并行构造算法.

输入. 加权种子点集 $V=\{(\boldsymbol{v}_i,w_i)\}_{i=1}^n$.

输出. power 胞元 $\{\Omega_i\}_{i=1}^n$.

Step1. 并行地生成映射点 $\{\boldsymbol{p}_i\}_{i=1}^n$.

Step2. 开辟一组四维方格, 并行地按照坐标将各个映射点放入对应方格内.

Step3. 并行计算各个 power 胞元 $\{\Omega_i\}_{i=1}^n$:

Step3.1. $\Omega_i \leftarrow$ 求解域的包围盒, 更新 $d_{\max}$;

Step3.2. $g_i \leftarrow \boldsymbol{p}_i$ 所在的方格;

Step3.3. 环数 $r \leftarrow 0$, $r_g \leftarrow 0$;

Step3.4. 若 $r_g \geqslant 2d_{\max}$, 转 Step3.7, 否则执行下一步;

Step3.5. 访问所有以 g_i 为中心在环数 r 上的方格, 遍历其内除 $\boldsymbol{p}_i$ 之外的其他映射点, 用它们与 $\boldsymbol{p}_i$ 的中垂面依次对 Ω_i 进行裁剪, 并实时更新 $d_{\max}$;

Step3.6. $r \leftarrow r+1$, 更新 r_g, 转 Step3.4;

Step3.7. 输出 Ω_i, 结束.

算法 1 的计算效率受方格内平均映射点数量 k_g 影响. 当其数值较大时, 每个方格内的映射点增多, 导致算法 1 在某一环的裁剪过程需要遍历更多的相邻映射点, 可能会产生过多的无效裁剪. 图 4 展示了不同的 k_g 取值对算法 1 的效率对比, 测试对象以立方体为求解域及在其内随机产生的 4 组加权种子点集, 统计各组种子点集在取不同的 k_g 值时算法 1 的耗时. 从图 4 可以看出, 较低的 k_g 取值有利于算法 1 尽快结束, 即可以尽快达到胞元裁剪的结束条件. 但是 k_g 越小, 方格的数量越多, 对显卡存储容量的要求越高. 为平衡效率和硬件要求, 本文默认取 $k_g=2$.

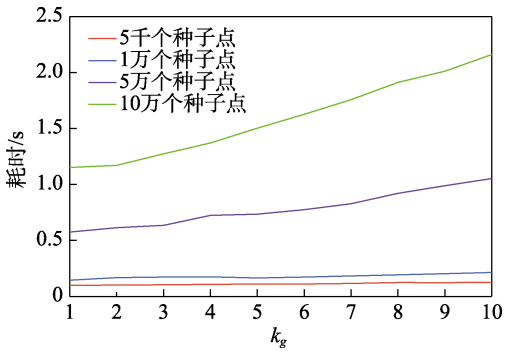


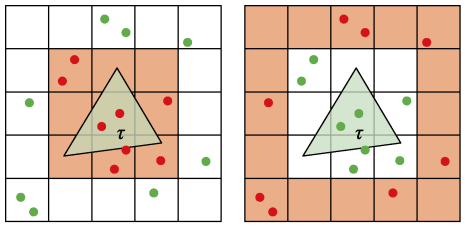
图 4　不同 k_g 取值对算法 1 的效率对比

3.2 power 图与求解域的交集

当求解域为规则的长方体时，算法 1 输出的胞元即为最终结果. 但是，在许多实际应用中，求解域往往具有复杂的边界. 因此，需要进一步求解 power 图与求解域的交集.

一种简单的做法是，并行地让每个胞元与求解域的每个四面体进行求交，但这种方式只适用于求解域的四面体数量很少或每个 power 胞元都有极大概率与所有四面体有交集的情况，否则当四面体数量较多时将非常耗时. Liu 等[20]采取的做法是，对每个四面体找出离其重心最近的若干个种子点，再求它与这些种子点对应胞元的交集. 这种做法存在一个不可忽视的问题，即当四面体较为狭长时，可能无法找出更远的与之有交集的胞元，导致结果不精确.

为了获得 power 胞元与求解域的精确交集，本文继续采用基于上述方格的求交策略. 如图 5 所示，对求解域某个四面体 τ，首先根据其包围盒确定包含图 5a 所示四面体的初始方格区域，并行地计算该区域内所有映射点的 power 胞元与该四面体的交集. 当存在以下情况之一时，需要如图 5b 所示向外扩大方格范围继续搜索，即：(1) 该四面体还未被裁剪过；(2) 某个映射点 $\boldsymbol{p}_i$ 的 power 胞元的边界面不全在该四面体外部，且该边界面是由 $\boldsymbol{p}_i$ 与其他映射点 $\boldsymbol{p}_j$ 的中垂面裁剪所得，$\boldsymbol{p}_j$ 所在的方格在当前区域的外层. 用符号 $r_{\max}$ 标记所有 $\boldsymbol{p}_j$ 所在的方格到初始方格区域的最大环数距离，其初值为 0，当出现情况(2)时更新 $r_{\max}$. 当以上 2 种情况均不满足时，称达到求交计算的结束条件，此时针对四面体 τ 的计算结束. 该并行求交过程产生的结果同样是精确的.



a. 初始方格区域　　b. 继续向外搜索

图 5　基于方格的求解域与 power 图求交策略示意图

算法 2. power 图与求解域的并行求交算法.

输入. 四面体集合 $\{\tau_k\}_{k=1}^{m}$ 和胞元 $\{\Omega_i\}_{i=1}^{n}$.

输出. 各个胞元与求解域的交集 $\left\{\Omega_{i|M}\right\}_{i=1}^{n}$.

Step1. 并行处理 $\{\tau_k\}_{k=1}^{m}$：

Step1.1. 根据 τ_k 的包围盒确定方格区域 G_0；

Step1.2. 环数 $r \leftarrow 0$，$r_{\max} \leftarrow 0$；

Step1.3. 获取距 G_0 环数为 r 的方格区域 G_r；

Step1.4. 并行访问 G_r 中的每个映射点，计算各个映射点对应 power 胞元 Ω_i 与 τ_k 的交集 t_{ik}，$\Omega_{i|M} \leftarrow \Omega_{i|M} \bigcup t_{ik}$，$r_{\max} \leftarrow t_{ik}$ 边界面对应的邻居映射点所在方格距 G_0 的最大环数；

Step1.5. 环数 $r \leftarrow r+1$；

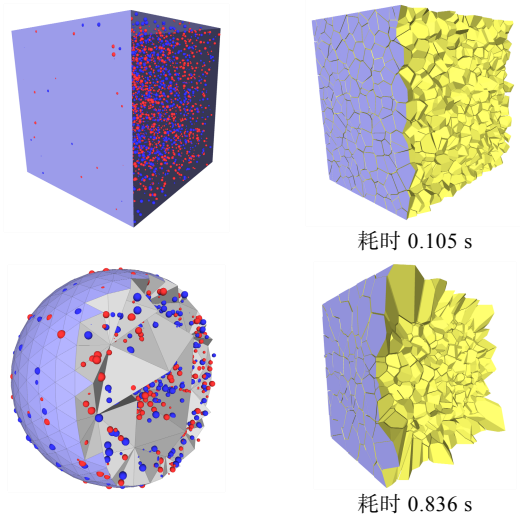
Step1.6. 若 $r < r_{\max}$，转 Step1.3；否则转 Step2.

Step2. 输出 $\Omega_{i|M}$，结束.

4 算法改进

实际上，很大一部分种子点的 power 胞元与求解域的交集实际上即为 power 胞元本身，称这部分种子点为内部种子点，剩余的部分称为边界种子点. 将算法 2 作用于内部种子点是没有必要的，只需计算边界种子点的 power 胞元与求解域的交集即可.

此外，对于边界复杂的求解域且种子点只在求解域内部的情况，在算法 1 中，边界种子点的胞元在经过最初的若干次裁剪后，真正起作用的裁剪可能已经结束. 但是由于这些种子点到各自胞元的最远距离较大且一直无法降低，它们在算法 1 中达到结束条件需要访问更多更远的方格，这使得算法 1 的效率被极大地拉低. 图 6 展示了一个对比示例. 在种子点数量均为 5 千的情况下，以立方体作为输入时算法 1 所需的耗时仅为 0.105 s，而以球体为求解域时算法 1 的耗时是立方体情况的 8 倍. 因此，应避免直接计算复杂求解域情况下边界种



a. 输入求解域与 5 千种子点　　b. 算法 1 结果

图 6　算法 1 对不同输入的效率对比

子点的 power 胞元.

为了提高算法 1 和算法 2 的整体效率, 本文提出一种改进的算法过程, 即算法 1 只作用于内部种子点, 而算法 2 只作用于边界种子点.

一个关键的问题是如何区分内部或边界种子点. 解决方法是计算 power 图约束在求解域边界上的受限 power 图. 将算法 2 输入中的四面体集合替换为求解域的边界三角形集合, 并将其中的求交步骤替换为图 3 的裁剪步骤, 可以得出部分种子点的受限 power 胞元非空, 即为边界种子点, 其余为内部种子点. 接下来只需对内部种子点执行算法 1 即可.对于边界种子点, 由于未获得显式的 power 胞元用于算法 2 进行求交计算, 本文采用上述计算受限 power 图的过程对边界四面体进行裁剪, 并只保留边界种子点的裁剪结果. 当边界四面体的相邻四面体也与边界种子点的 power 胞元有交集时, 需将该四面体放入算法 2 的输入集合. 因此边界种子点的胞元结果包含了若干个被裁剪的四面体, 与内部种子点的 power 胞元共同组成最终的结果.

改进的算法过程比原始版本具有更高的计算效率. 图 7 展示了以球体为求解域、不同数量种子点时两种算法过程的效率对比. 可以看出, 改进的算法过程所需的耗时远远少于原始版本的耗时, 并且随着种子点数量的增多, 加速比越高.

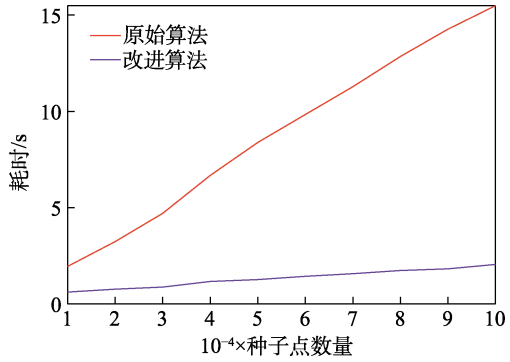


图 7　改进的算法过程与原始版本的效率对比

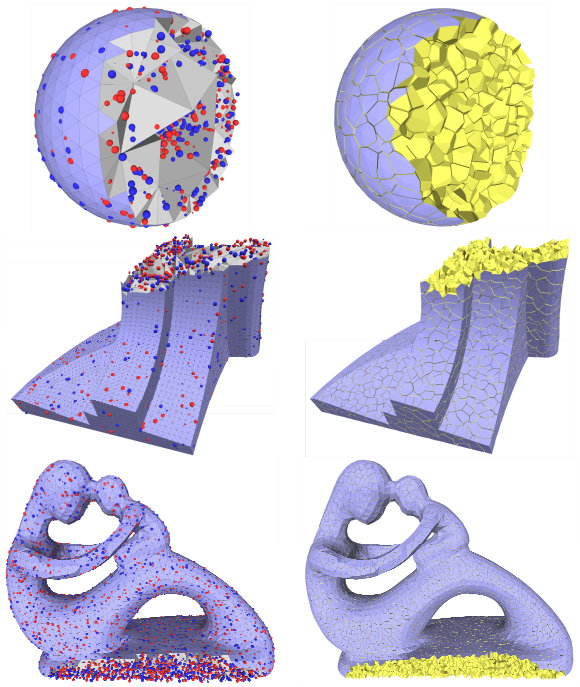
5　实验结果和比较

基于 CUDA 11.0 计算平台实现了本文方法, 实验结果在配备有 Intel Core i7-12700 2.1 GHz CPU, 16 GB RAM 和 NVIDIA RTX 3080 GPU 的 PC 上获得, 浮点数采用双精度. 采用 Sutherland Hodgeman 算法[24]裁剪多边形和多面体.

下面给出本文方法的结果, 并与主流方法进行对比, 包括基于正则三角网格对偶性的 CPU 串行方法(RT 方法[21], 采用计算几何算法库[25]实现), Liu 等[20]基于无网格方法的 GPU 实现(需将本文的四维映射点作为该方法的输入).

5.1　实验结果

图 8 给出了本文方法获得 power 图结果的 3 个示例, 图 8a 所示为输入的求解域(分别是四面体化后的球、fandisk 和 fertility 模型)和加权种子点(以颜色球表示, 红色代表正权值, 蓝色代表负权值, 球越大代表权值绝对值越大), 数量分别为 5 000, 10 000 和 50 000, 权重在求解域包围盒的 [−1%, 1%]范围内随机生成; 图 8b 所示为本文方法获得的最终 power 图结果. 本文方法具有理论上的保证, 不考虑计算机的数值精度误差, 所得的 power 图结果及其与求解域的交集均是精确解.



a. 输入求解域及加权种子点　　b. 最终 power 图结果

图 8　本文方法获得的 power 图结果

以球体作为求解域, 种子点及其权重在球范围内随机生成, 本文测试了不同种子点数量下与 RT 方法[21]和 Liu 等[20]方法的耗时对比, 不同方法对每组种子点的耗时统计为运行 10 次后的平均值, 曲线在图 9 中给出. 可以看到, 随着种子点数量的增长, 传统的 RT 方法所需的时间成本增长较快, Liu 等[20]方法所需的时间成本较低且变化较缓慢, 本文方法的耗时则略高于 Liu 等方法. 这是由于本文方法的胞元裁剪次数通常会比 Liu 等方法多. 然而, 在种子点分布呈现白噪声性质时, Liu 等方法产生的部分胞元比精确解具有更大的体积, 而本文在

耗时没有大幅增加的前提下保证了结果的精确性. 图 10 展示了一个对比示例. 每个胞元的体积与精确解体积的差值相对于精确解体积的比值被映射为一个颜色, 并将该胞元绘制为对应的颜色. 图 10a 所示为 Liu 等方法结果中部分胞元的体积比精确解更大, 图 10b 所示为本文方法获得的精确结果.

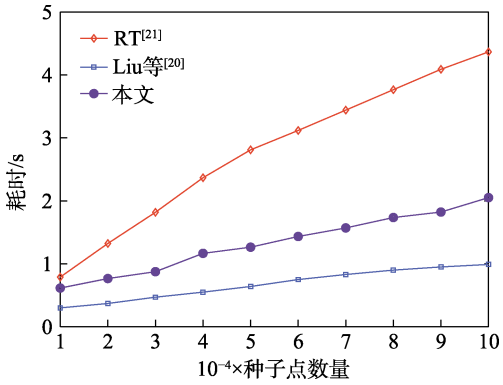
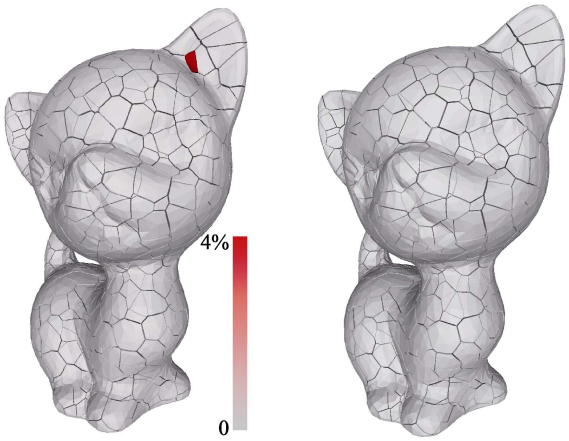


图 9 3 种方法在不同种子点数量时的耗时对比



a. Liu等[20]的近似结果 b. 本文的精确结果

图 10 Liu 等[20]方法与本文方法的结果对比

此外, 表 1 分步骤地给出了本文方法与 RT 方法的耗时对比. 可以看出, 在输入数据量更大的情况下, 本文方法比 RT 方法具有更高的加速比.

表 1 2 种方法构造三维精确 power 图的步骤耗时对比

模型	种子点数量	四面体数量	RT 方法[21]			本文			
			构造 RT/s	裁剪/s	总计/s	受限 power 图/s	内部胞元/s	边界胞元/s	总计/s
立方体	5000	6	0.054	0.132	0.186	不适用	0.105	不适用	**0.105**
球	5000	1983	0.055	0.416	0.471	0.027	0.082	0.233	**0.342**
fandisk	10000	25822	0.103	1.475	1.578	0.102	0.157	0.965	**1.224**
fertility	50000	15663	0.763	3.893	4.656	0.073	0.423	0.981	**1.477**

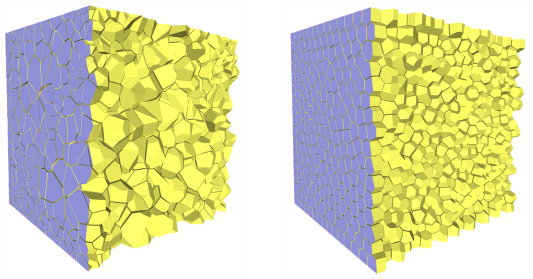
注. 粗体表示耗时较少.

5.2 应 用

蓝噪声采样是计算机图形学领域的一个重要问题. de Goes 等[5]和 Xin 等[6]分别提出了带容积约束的重心 power 图方法, 能够获取与给定密度自适应的均匀点集, 即通过求解

$$E\left(\{\boldsymbol{v}_i, w_i\}_{i=1}^n\right)=\sum_{i=1}^n\int_{\Omega_i}\rho(\boldsymbol{x})\|\boldsymbol{x}-\boldsymbol{v}_i\|^2\,\mathrm{d}\boldsymbol{x}-\sum_{i=1}^n w_i\left(\int_{\Omega_i}\rho(\boldsymbol{x})\mathrm{d}\boldsymbol{x}-c\right) \tag{3}$$

的最小值来获得. 其中, $\rho(\boldsymbol{x})$ 是给定的密度函数; Ω_i 是 power 胞元与求解域的交集; c 是给定的容积约束. 本文设定 $\rho(\boldsymbol{x})\equiv 1$ 来生成全局均匀的采样结果, 并采用 Xin 等算法来搜索式(3)的最优解. 在该算法的迭代过程中, power 图的计算被高频率地使用, 这里将其替换为本文方法. 图 11 展示了立方体内 3000 个采样点的初始化和迭代 100 次后的 power 图结果. 在该例子中, 与采用 RT 方法相比, 在 Xin 等算法中采用本文方法计算 power 图以生成蓝噪声采样结果时可节省 53 s.



a. 初始 power 图 b. 100 次迭代后的 power 图

图 11 本文方法应用于重心 power 图的生成

6 结 语

本文提出了一种加权种子点的映射方法, 可以构造与原 power 图等价的高一维受限 Voronoi 图, 使得任意的 Voronoi 图计算方法可以直接应用于 power 图的计算问题中. 基于此, 本文将无网格方法推广用于 power 图的计算. 与现有基于该方法的 GPU 算法要求严格的种子点邻居顺序不同, 本文采用一种基于方格的邻居搜索和胞元裁剪策略,

可以保证生成的 power 图结果是精确的.

然而, 本文方法也存在局限性. 如图 12 所示, 当加权种子点的分布较为极端时, 在裁剪过程中, 某些种子点到其胞元的最远距离 $d_{\max}$ 可能一直无法降低, 导致裁剪结束条件很难达到, 这些种子点将遍历过多的邻居, 导致算法的效率很低. 因此, 本文方法还值得进一步改进, 扩大其适用范围.

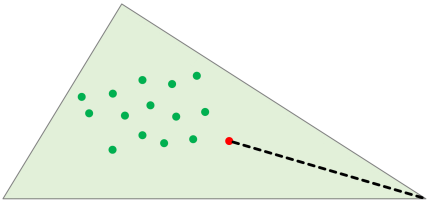


图 12　本文方法局限性示意图

参考文献(References):

[1] Aurenhammer F. Voronoi diagrams - a survey of a fundamental geometric data structure[J]. ACM Computing Surveys, 1991, 23(3): 345-405

[2] Aurenhammer F. Power diagrams: properties, algorithms and applications[J]. SIAM Journal on Computing, 1987, 16(1): 78-96

[3] Fiedler M, Alpers A. Power-SLIC: fast superpixel segmentations by diagrams[OL]. [2023-04-15]. https://arxiv.org/abs/2012.11772

[4] Ma D Y, Zhou Y F, Xin S Q, *et al*. Convex and compact superpixels by edge-constrained centroidal power diagram[J]. IEEE Transactions on Image Processing, 2021, 30: 1825-1839

[5] de Goes F, Breeden K, Ostromoukhov V, *et al*. Blue noise through optimal transport[J]. ACM Transactions on Graphics, 2012, 31(6): Article No.171

[6] Xin S Q, Lévy B, Chen Z G, *et al*. Centroidal power diagrams with capacity constraints: computation, applications, and extension[J]. ACM Transactions on Graphics, 2016, 35(6): Article No.244

[7] Budninskiy M, Liu B B, de Goes F, *et al*. Optimal voronoi tessellations with hessian-based anisotropy[J]. ACM Transactions on Graphics, 2016, 35(6): Article No.242

[8] Xiao Y Y, Chen Z G, Cao J, *et al*. Optimal power diagrams via function approximation[J]. Computer-Aided Design, 2018, 102: 52-60

[9] Yan D M, Wonka P. Gap processing for adaptive maximal Poisson-disk sampling[J]. ACM Transactions on Graphics, 2013, 32(5): Article No.148

[10] Yan D M, Guo J W, Jia X H, *et al*. Blue-noise remeshing with farthest point optimization[J]. Computer Graphics Forum, 2014, 33(5): 167-176

[11] de Goes F, Wallez C, Huang J, *et al*. Power particles: an incompressible fluid solver based on power diagrams[J]. ACM Transactions on Graphics, 2015, 34(4): Article No.50

[12] Zhai X, Hou F, Qin H, *et al*. Fluid simulation with adaptive staggered power particles on GPUs[J]. IEEE Transactions on Visualization and Computer Graphics, 2020, 26(6): 2234-2246

[13] Brown K Q. Voronoi diagrams from convex hulls[J]. Information Processing Letters, 1979, 9(5): 223-228

[14] Fortune S. A sweepline algorithm for Voronoi diagrams[J]. Algorithmica, 1987, 2(1-4): 153-174

[15] Green P J, Sibson R. Computing Dirichlet tessellations in the plane[J]. The Computer Journal, 1978, 21(2): 168-173

[16] Yan D M, Lévy B, Liu Y, *et al*. Isotropic remeshing with fast and exact computation of restricted Voronoi diagram[J]. Computer Graphics Forum, 2009, 28(5): 1445-1454

[17] Yan D M, Wang W P, Lévy B, *et al*. Efficient computation of clipped Voronoi diagram for mesh generation[J]. Computer-Aided Design, 2013, 45(4): 843-852

[18] Lévy B, Bonneel N. Variational anisotropic surface meshing with Voronoi parallel linear enumeration[C] //Proceedings of the 21st International Meshing Roundtable. Heidelberg: Springer, 2013: 349-366

[19] Ray N, Sokolov D, Lefebvre S, *et al*. Meshless voronoi on the GPU[J]. ACM Transactions on Graphics, 2018, 37(6): Article No.265

[20] Liu X H, Ma L, Guo J W, *et al*. Parallel computation of 3D clipped Voronoi diagrams[J]. IEEE Transactions on Visualization and Computer Graphics, 2022, 28(2): 1363-1372

[21] Edelsbrunner H, Shah N R. Incremental topological flipping works for regular triangulations[J]. Algorithmica, 1996, 15(3): 223-241

[22] Gui Zhiqiang, Yao Yuyou, Zhang Gaofeng, *et al*. An efficient computation method of 3D-power diagram[J]. Journal of Zhejiang University: Science Edition, 2021, 48(4): 410-417(in Chinese)
(桂志强, 姚裕友, 张高峰, 等. 3D-power 图的快速生成方法[J]. 浙江大学学报: 理学版, 2021, 48(4): 410-417)

[23] Basselin J, Alonso L, Ray N, *et al*. Restricted power diagrams on the GPU[J]. Computer Graphics Forum, 2021, 40(2): 1-12

[24] Sutherland I E, Hodgeman G W. Reentrant polygon clipping[J]. Communications of the ACM, 1974, 17(1): 32-42

[25] CGAL Editorial Board. CGAL User and reference manual[OL]. [2023-04-15]. https://www.cgal.org