前沿计算研究实践（II）	**2019 年春季**

专题 10：离散数据插值

教授：陈宝权　　作者：胡新宇、叶振涛、史百丰、李世成、张文涵、唐正纲

免责声明：该笔记尚未进行正式出版物的通常审查，未经教师允许不能向课程外分发.

10.1　问题引入

10.1.1　问题介绍

2 维举例：给定所示不同灰度的点（如左图），如何计算出一个完整的图像（如右图），在每一个点插入对应灰度值?

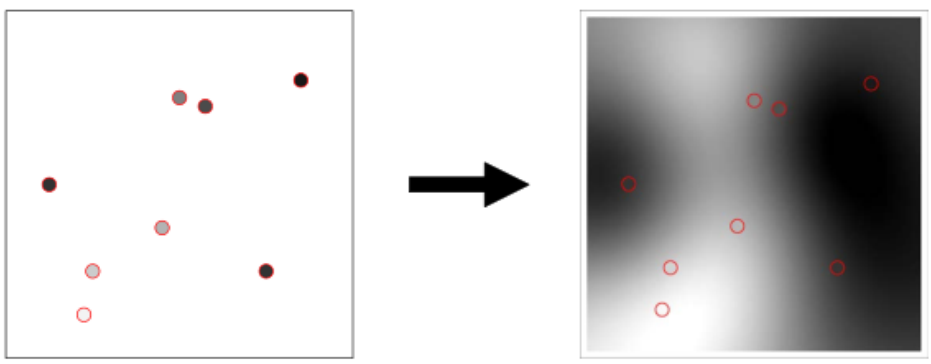


图 10.1: 对灰度点插值得到完整图像

10.1.2　形式化描述

给定一组对应于标量数据值 $\{v_1 \dots v_n\}$ 的 N 维数据点 $\{\boldsymbol{p}_1 \dots \boldsymbol{p}_n\}$，计算定义在 N 维空间上所有点的函数 $f(\boldsymbol{p}) : \mathbb{R}^N \to \mathbb{R}$ 满足 $f(\boldsymbol{p}_i) = v_i$, 对于 $i = 1, 2, \dots, n$.

定义 10.1 插值 在一组已知数据点的范围内添加新数据点的方法。

可以使用插值来填充缺失的数据、对现有数据进行平滑处理以及进行预测等。

10.1.3 应用举例：表面重构

10.1.3.1 描述

大多数情况下，我们将真实世界的 3D 对象捕获为对象表面上的数据点云，然后使用算法重建它。为了便于讨论，我们考虑 2D 情况，其中点云由蓝色点突出显示，我们需要连接其形成闭合曲线。

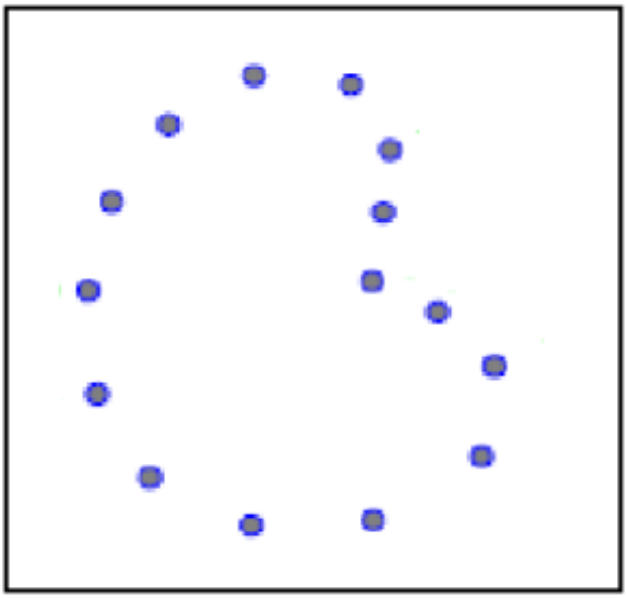


图 10.2: 2 维点云

一个直接的想法：将点云中的每个点视为强度为 0 的数据点。我们只需要找到插值函数 $f(\boldsymbol{p}):\mathbb{R}^2\to\mathbb{R}$ 满足对于所有 $\boldsymbol{p}_i$ 点，有 $f(\boldsymbol{p}_i)=0$. 但存在一个平凡解为常值函数 $f(\boldsymbol{p})\equiv 0$，任何算法都可能得出这个我们并不关心的解。为避免这种情况，我们在要重建的曲线外部或内部的位置添加一些具有正或负强度的人工数据点。

10.1.3.2 插值

对于每一个给定的 $\boldsymbol{p}_i$, 根据初始点云能够计算得到对应的向外单位法向量 $\hat{\boldsymbol{n}}_i$，我们在 $\boldsymbol{p}_i+d\hat{\boldsymbol{n}}_i$ 位置构造绿色数据点带有强度 d，在 $\boldsymbol{p}_i-d\hat{\boldsymbol{n}}_i$ 位置构造红色数据点带有强度 $-d$。附加约束强制插值函数 $f(\boldsymbol{p})$ 在曲线内变为负，在外部变为正，此时 $f(\boldsymbol{p})=0$ 即为连接原始点云的闭合曲线。

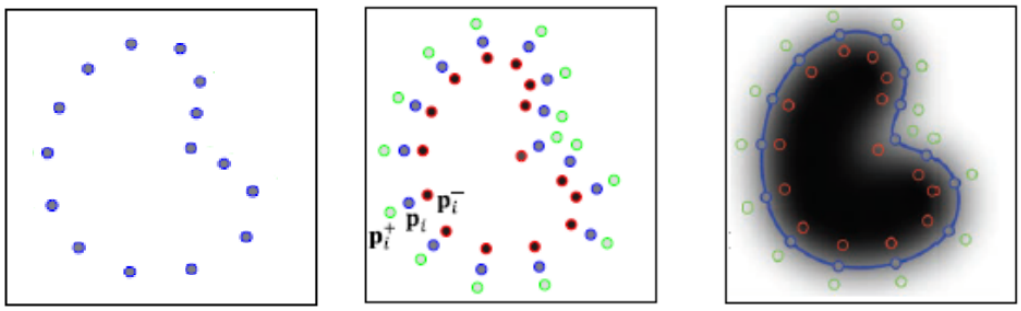


图 10.3: 插值重构

上述想法可以自然推广到三维的情况，得到插值函数之后，利用一些软件即可重构出目标表面。

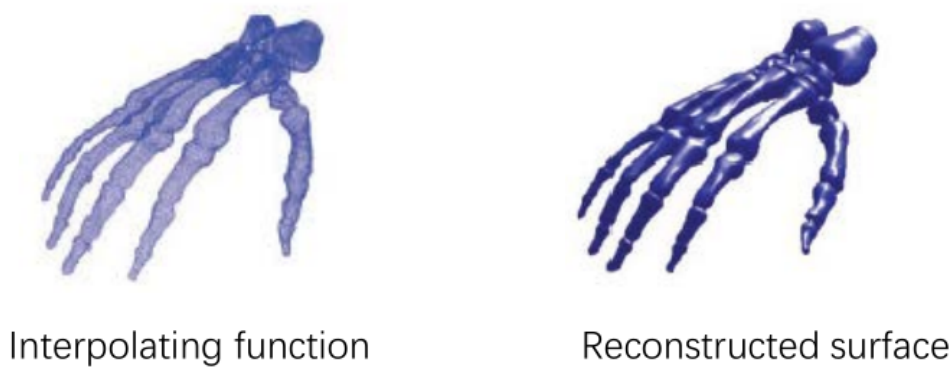


图 10.4: 3 维插值重构

从而问题的核心在于得到合理的插值函数。对于插值函数，我们主要关注它的光滑性和低振荡性，这两点对于直觉上的美观和实际应用上来说都是至关重要的。接下来我们将介绍三类不同的插值函数。

10.2 分段线性插值 (Piecewise linear interpolation)

10.2.1 一维情况

10.2.1.1 两个数据点

考虑简单的情况，若仅有两个分布在一维的数据点和对应强度值：$(P_1, v_1), (P_2, v_2)$。自然的插值函数是将两点连线如图所示：

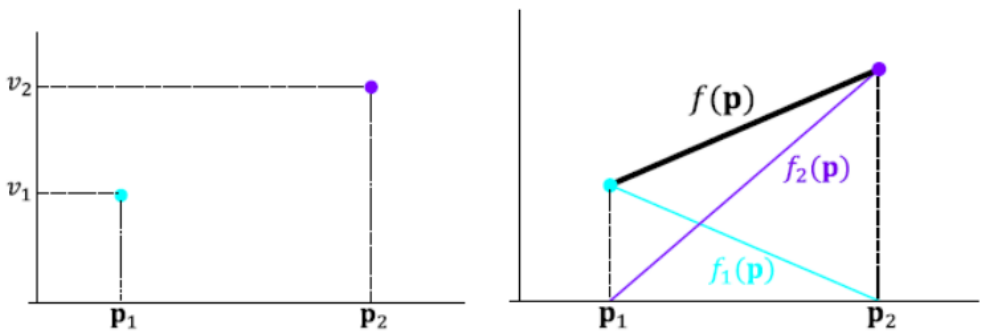


图 10.5: 1 维简单情况

可以将插值函数表示为:

$$f(\boldsymbol{p}) = \frac{\boldsymbol{p}_2 - \boldsymbol{p}}{\boldsymbol{p}_2 - \boldsymbol{p}_1} v_1 + \frac{\boldsymbol{p} - \boldsymbol{p}_1}{\boldsymbol{p}_2 - \boldsymbol{p}_1} v_2 = f_1(\boldsymbol{p}) + f_2(\boldsymbol{p})$$

10.2.1.2 多个数据点

对于多个数据点的情况，我们可以将每两个相邻的点做线性插值，最后连接得到一条折线，类似上述表示形式，我们定义钟形函数（bump function）如下：

定义 10.2 ***bump function***

$$f_i(\boldsymbol{p}) = \begin{cases} \frac{\boldsymbol{p}-\boldsymbol{p}_{i-1}}{\boldsymbol{p}_i-\boldsymbol{p}_{i-1}} v_i & \textit{if} \quad i > 1 \quad \textit{and} \quad \boldsymbol{p} \in [\boldsymbol{p}_{i-1}, \boldsymbol{p}_i), \\ \frac{\boldsymbol{p}_{i+1}-\boldsymbol{p}}{\boldsymbol{p}_i-\boldsymbol{p}_{i-1}} v_i & \textit{if} \quad i < n \quad \textit{and} \quad \boldsymbol{p} \in [\boldsymbol{p}_i, \boldsymbol{p}_{i+1}], \\ 0 & \textit{otherwise} \end{cases}$$

对于所有 $\boldsymbol{p} \in [\boldsymbol{p}_1, \boldsymbol{p}_n]$.

注意对于边界点 ($\boldsymbol{p}_1$ 和 $\boldsymbol{p}_n$) 而言，对应钟形函数只在一段区间内非零，而对于任意内部点，对应钟形函数均有两段取值非零区间。利用钟形函数，对于多个点我们得到的插值函数可以表示为:

$$f(\boldsymbol{p}) = \sum_{i=1}^{n} f_i(\boldsymbol{p})$$

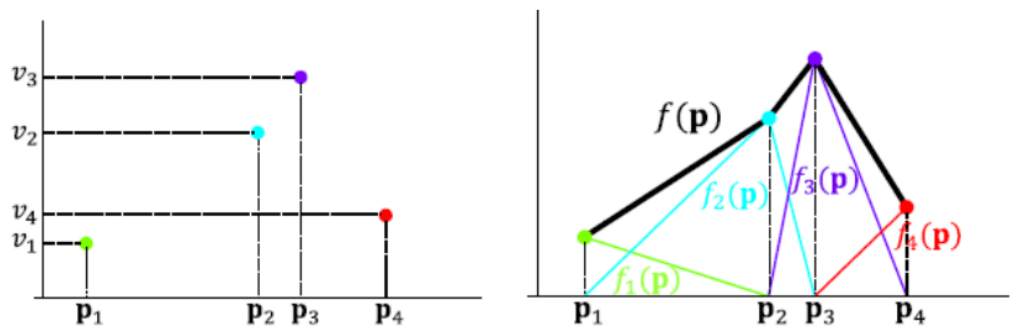


图 10.6: 1 维多数据点情况

10.2.1.3 改进策略

上述得到的插值函数即为将所有相邻点直接连接得到的折线，（除非恰好斜率相同）在每一个分界点（即原数据点）处均不光滑，其中一个原因是每一段函数均只利用了相邻两个点的信息。一个自然的推广是，每次利用相邻三个点的信息构造插值函数，即利用

$$f(\boldsymbol{p}) = \frac{(\boldsymbol{p}-\boldsymbol{p}_2)(\boldsymbol{p}-\boldsymbol{p}_3)}{(\boldsymbol{p}_1-\boldsymbol{p}_2)(\boldsymbol{p}_1-\boldsymbol{p}_3)}v_1 + \frac{(\boldsymbol{p}-\boldsymbol{p}_1)(\boldsymbol{p}-\boldsymbol{p}_3)}{(\boldsymbol{p}_2-\boldsymbol{p}_1)(\boldsymbol{p}_2-\boldsymbol{p}_3)}v_2 + \frac{(\boldsymbol{p}-\boldsymbol{p}_1)(\boldsymbol{p}-\boldsymbol{p}_2)}{(\boldsymbol{p}_3-\boldsymbol{p}_1)(\boldsymbol{p}_3-\boldsymbol{p}_2)}v_3$$

构造插值函数。相比仅利用两个点的线性插值，得到的插值函数光滑性得到了一定程度的改善 (每三个点的中间点保证光滑)。

但是在分界点仍然不光滑，可以预料，按照上述方法即使扩展到考虑每 k 个相邻点，在分界处仍然会不光滑，并且计算代价也大为提升。

10.2.1.4 三次样条插值 (Cubic Interpolation)

为了得到一个光滑的插值函数，在此我们介绍名为三次样条插值的方法。该方法的核心思想是，每次利用相邻四个点的信息，但仅用来构造中间两个点之间的插值函数，从而使得每段插值函数之间仍能保

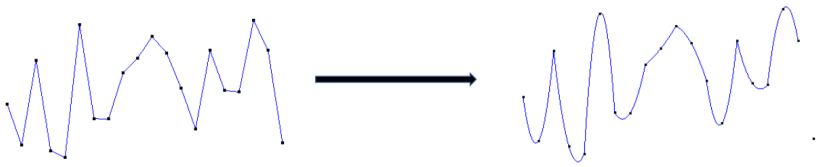


图 10.7: 3 近邻插值

持光滑性。

考虑依次序排列的四个点 (p_0, p_1, p_2, p_3), 我们将利用这四个点的信息构造连接 p_1 和 p_2 的插值函数，以参数方程记为 $f(x) = ax^3 + bx^2 + cx + d$，$x \in [0, 1]$, 即有 $f(0) = p_1$,$f(1) = p_2$(为简单起见，此处用 p_i 表示对应点的强度值，请注意区分)。显然该方程的导数 $f'(x) = 3ax^2 + 2bx + c$ 在 $[0, 1]$ 间连续。为确定四个参数我们需要四个方程，其中两个是 $f(0) = p_1$,$f(1) = p_2$ 已知，我们应当利用剩下两个方程使得构造得到的插值函数满足在分界点光滑，一个简单的想法是考虑下列四个方程：

$$
\begin{aligned}
f(0) &= p_1 \\
f(1) &= p_2 \\
f'(0) &= \frac{p_2 - p_0}{2} \\
f'(1) &= \frac{p_3 - p_1}{2}
\end{aligned}
$$

简单验证即可得到满足上述方程的插值函数能够满足在相邻点保持光滑性的要求。解之，得到

$$
\begin{aligned}
a &= -\frac{1}{2}p_0 + \frac{3}{2}p_1 - \frac{3}{2}p_2 + \frac{1}{2}p_3 \\
b &= p_0 - \frac{5}{2}p_1 + 2p_2 - \frac{1}{2}p_3 \\
c &= -\frac{1}{2}p_0 + \frac{1}{2}p_2 \\
d &= p_1
\end{aligned}
$$

最后得到的插值函数每段形如：

$$
f(p_0, p_1, p_2, p_3, x) = (-\frac{1}{2}p_0 + \frac{3}{2}p_1 - \frac{3}{2}p_2 + \frac{1}{2}p_3)x^3 + (p_0 - \frac{5}{2}p_1 + 2p_2 - \frac{1}{2}p_3)x^2 + (-\frac{1}{2}p_0 + \frac{1}{2}p_2)x + p_1
$$

相比于每次考虑 3 个相邻点的插值函数，光滑性得到了进一步改善。

注意到三次样条插值实际上无法构造出第一段和最后一段插值函数的合理取值，这可以由别的方法补充完整。事实上，三次样条插值虽然保证了插值函数的一阶光滑性，但是并不能保证二阶以及更高阶的光滑性（可以利用类似的想法直接构造出二阶光滑的插值函数，但是计算复杂度会有质的变化）。

图 10.8: 三次样条插值

10.2.2 二维及高维情况

为了将线性插值应用于高维的情况，我们需要将数据点细分成多个小的区域，并在每一个小区域中应用线性插值的方法。对于二维的情况，我们首先将数据点分割为一个一个三角形。对于每一个三角形区域，我们先利用一维的线性插值方法计算出两点之间连边的强度值，并利用边上一点和顶点进一步计算每一个三角形内部点的强度值从而得到整个二维区域的强度分布。

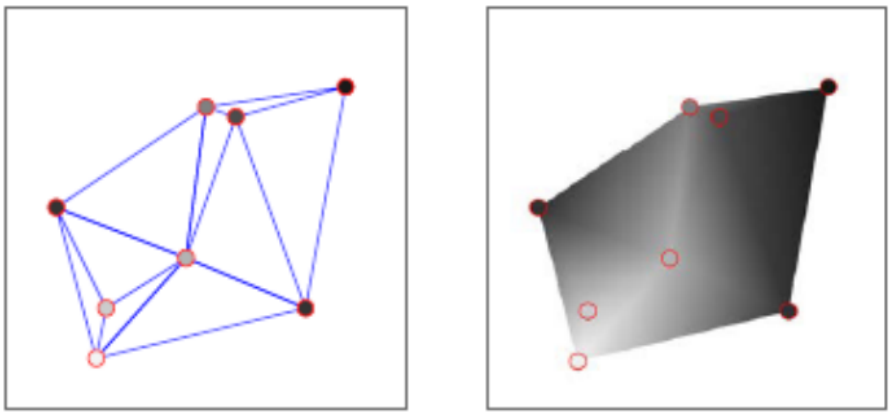


图 10.9: 二维插值

上述方法可以自然推广到三维及以上，但是一个问题是最后得到的强度分布会依赖于我们分割三角形的方式，从而导致解法的不确定性。下图展示了两种不同三角形分割方式下得到的强度分布。

多解性（或者说歧义性）是线性插值方法的一个固有弊端。事实上我们只需要知道，线性插值具有简单直接的特点，但是往往不够光滑。我们需要得到性质更加优越的插值函数。

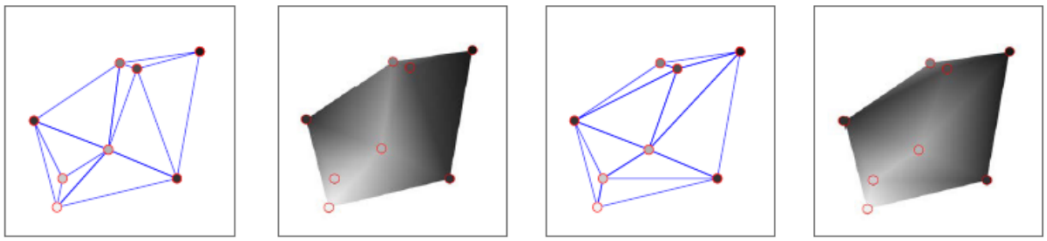


图 10.10: 多解性

10.3 Shepard 插值（Shepard's Interpolation）

定义 10.3 ***Shepard* 插值**
给定数据点（*data points*）$\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ 及数据值（*data values*）$\{v_1, \ldots, v_n\}$，选择钟形函数为

$$f_i(\mathbf{p}) = \frac{(\|\mathbf{p} - \mathbf{p}_i\|^{-\alpha})}{\sum_{j=1}^{n}(\|\mathbf{p} - \mathbf{p}_j\|^{-\alpha})} v_i$$

则关于这组数据点和数据值的 *Shepard* 插值函数为

$$f(\mathbf{p}) = \sum_{i=1}^{n} f_i(\mathbf{p})$$

其中 $\|\cdot\|$ 是欧氏距离且 $\alpha > 0$.

定理 10.4 *Shepard* 插值函数是对数据点 $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ 及数据值 $\{v_1, \ldots, v_n\}$ 的插值函数.

证明：
当 $\mathbf{p} \to \mathbf{p}_i$ 时，

$$f_i(\mathbf{p}) = \frac{(\|\mathbf{p} - \mathbf{p}_i\|^{-\alpha})}{\sum_{j=1}^{n}(\|\mathbf{p} - \mathbf{p}_j\|^{-\alpha})} v_i = \left(1 - \frac{\sum_{k \neq i}(\|\mathbf{p} - \mathbf{p}_k\|^{-\alpha})}{\sum_{j=1}^{n}(\|\mathbf{p} - \mathbf{p}_j\|^{-\alpha})}\right) v_i \to v_i.$$

当 $\mathbf{p} \to \mathbf{p}_j$ 时，

$$f_i(\mathbf{p}) = \frac{(\|\mathbf{p} - \mathbf{p}_i\|^{-\alpha})}{\sum_{j=1}^{n}(\|\mathbf{p} - \mathbf{p}_j\|^{-\alpha})} v_i \to 0.$$

故 $f(\mathbf{p})$ 满足对于任意 $i = 1, 2, \ldots, n$ 都有 $f(\mathbf{p}_i) = v_i$. ■

相比于分段线性插值（piecewise linear interpolation），Shepard 插值具有如下优点：

- 更加光滑，在 $\alpha > 1$ 时插值函数一阶可导；

- 插值函数的定义域不受给定数据点的限制，可以外推到全空间。

但 Shepard 插值的缺点在于，插值函数在数据点处的一阶导数为 0，这使得计算出的插值函数不够平滑，因此我们引入了径向基函数插值。

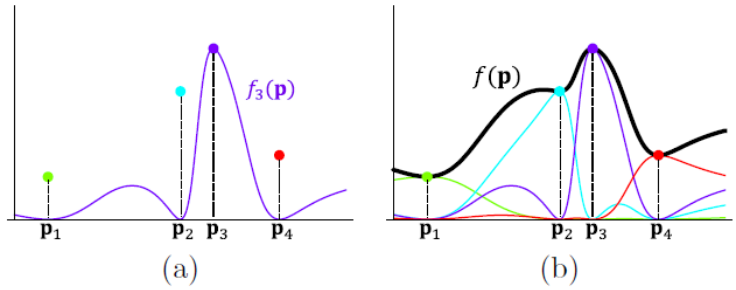


图 10.11 一维情形下的 Shepard 插值结果。左图为在 $\mathbf{p}_3$ 处的钟形函数，右图中的黑色曲线为 Shepard 插值函数。

10.4 径向基函数（Radial Basis Functions）

定义 10.5 径向核函数（*radial kernel*）
给定径向函数 $\phi : [0, \infty) \to \mathbb{R}$，赋范向量空间 $(V, \|\cdot\|)$ 及点 $c \in V$，称 $\phi_c(x) := \phi(\|x - c\|)$ 为一个以 c 为中心的径向核函数.

定义 10.6 径向基函数（*radial basis function*，*RBF*）
给定径向函数 $\phi : [0, \infty) \to \mathbb{R}$ 和赋范向量空间 $(V, \|\cdot\|)$，集合 $\{\phi_c | c \in V\}$ 被称作一组径向基函数，如果对于任意点集 $(c_i)_{i=1}^n$，满足 $\phi_{c_1}, \cdots, \phi_{c_n}$ 线性无关，且矩阵

$$A = \begin{bmatrix} \phi(\|c_1 - c_1\|) & \cdots & \phi(\|c_1 - c_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|c_n - c_1\|) & \cdots & \phi(\|c_n - c_n\|) \end{bmatrix}$$

是可逆矩阵.

一些常用的径向函数包括:

- 高斯函数（Gaussian）：$\phi(r) = e^{-\epsilon r^2}$
- 多二次函数（multiquadratic）：$\phi(r) = \sqrt{1 + \epsilon r^2}$
- 逆二次函数（inverse quadratic）：$\phi(r) = \frac{1}{1+\epsilon r^2}$
- 薄板样条（thin plate spline）：$\phi(r) = r^2 \ln r$

- 多重调和样条（polyharmonic spline）：$\phi(r) = \begin{cases} r^k & k = 1, 3, 5, \cdots \\ r^k \ln r & k = 2, 4, 6, \cdots \end{cases}$

定义 10.7 径向基函数插值

给定数据点 $\{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ 及数据值 $\{v_1, \ldots, v_n\}$，取钟形函数为 $f_i(\mathbf{p}) = \phi(\|\mathbf{p} - \mathbf{p}_i\|)c_i$，并选取 c_i 使得 $f(\mathbf{p}_i) = \sum_{j=1}^{n} f_j(\mathbf{p}_i) = v_i$ 对于 $i = 1, 2, \ldots, n$ 都成立，即 c_i 是如下线性方程组的解：

$$A\mathbf{c} = \mathbf{v}$$

其中

$$A = \begin{bmatrix} \phi(\|p_1 - p_1\|) & \cdots & \phi(\|p_1 - p_n\|) \\ \vdots & \ddots & \vdots \\ \phi(\|p_n - p_1\|) & \cdots & \phi(\|p_n - p_n\|) \end{bmatrix}$$

$$\mathbf{c} = [c_1, \cdots, c_n]^T$$

$$\mathbf{v} = [v_1, \cdots, v_n]^T$$

则关于这组数据点和数据值的 *RBF* 插值函数为

$$f(\mathbf{p}) = \sum_{j=1}^{n} c_j f_j(\mathbf{p})$$

由于选取的这些径向基函数在更少的地方一阶导数为 0，因此 RBF 插值的结果比之前所介绍的两种插值的结果要更加平滑。下图给出了使用三重调和样条和高斯函数在一维情形下的 RBF 插值结果。

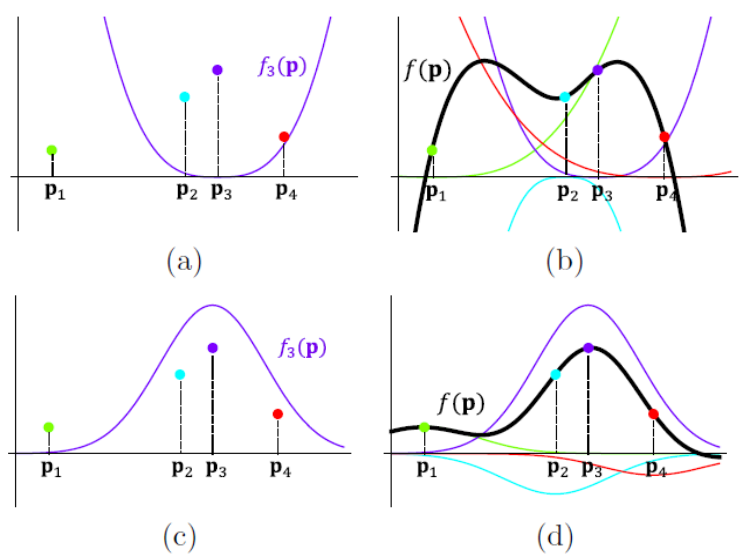


图 10.12 一维情形下的 RBF 插值结果。(a)(b) 为使用三重调和样条的插值结果，(c)(d) 为使用高斯函数的插值结果。

10.5 再现多项式

10.5.1 问题引入

径向基函数已经解决了光滑和震荡的问题，但是在有些方面还存在不足。比如考虑一维情况，我们的数据点是从一些简单的函数上采样得到的，但用径向基函数插值得到的函数比原函数复杂很多。

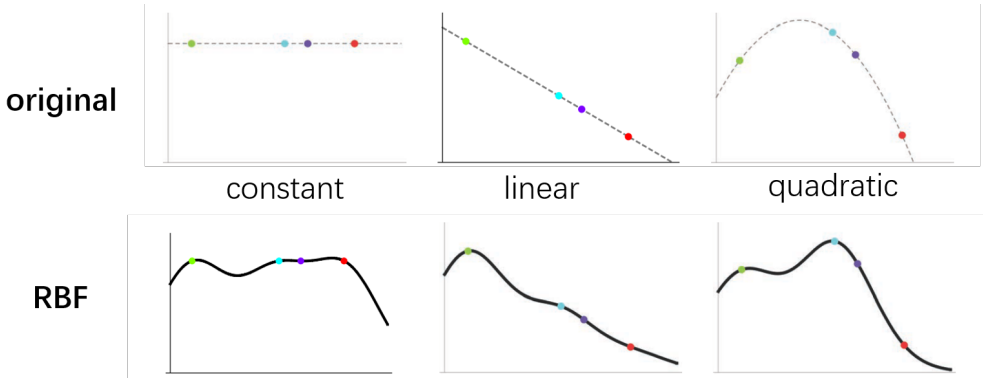


图 10.17 ，第一行是从简单函数采样，从左到右依次是常值函数，线性函数，二次函数，第二行是选取某个 RBF 插值的结果。

10.5.2 加入多项式项

一个自然的想法就是在之前的径向基函数前加上一个多项式项来增强表达能力。新的径向基函数如下:

$$f(p) = g(p) + \Sigma_{i=1}^{n} f_i(p)$$

其中，$g(p)$ 是最高次为 k 的多项式，k 的选取与希望重现的多项式有关。一般地，$g(p)$ 可表示成:

$$g(p) = \Sigma_{i=1}^{l} a_i m_i(p)$$

其中，$m_i(p)$ 是 p 的坐标分量的次数不超过 k 的单项式，a_i 是待定的参数
例如，二维情况，p_x, p_y 是点 p 的两个坐标分量，则
常值多项式:

$$g(p) = a_0$$

二次多项式:

$$g(p) = a_0 + a_1 p_x + a_2 p_y + a_3 p_x^2 + a_4 p_y^2 + a_5 p_x p_y$$

10.5.3 新的约束

引入多项式项后，增强了径向基函数的表达能力，但多项式系数带来了更多的未知数，需要找到新的约束条件。
这里，我们为每个单项式加上一个正交约束：

$$\Sigma_{i=0}^{n} c_i m_j(p_i) = 0, for j = 1, 2, ..., l$$

于是，需要求解的线性方程组如下：

$$\begin{bmatrix} A & M \\ M^T & 0 \end{bmatrix} \begin{bmatrix} c \\ a \end{bmatrix} = \begin{bmatrix} v \\ 0 \end{bmatrix}$$

其中，

$$M = \begin{bmatrix} m_1(p_1) & \cdots & m_l(p_1) \\ \vdots & \ddots & \vdots \\ m_1(p_n) & \cdots & m_l(p_n) \end{bmatrix}, a = \begin{bmatrix} a_1 & \cdots & a_l \end{bmatrix}^T$$

插值之后效果如下：

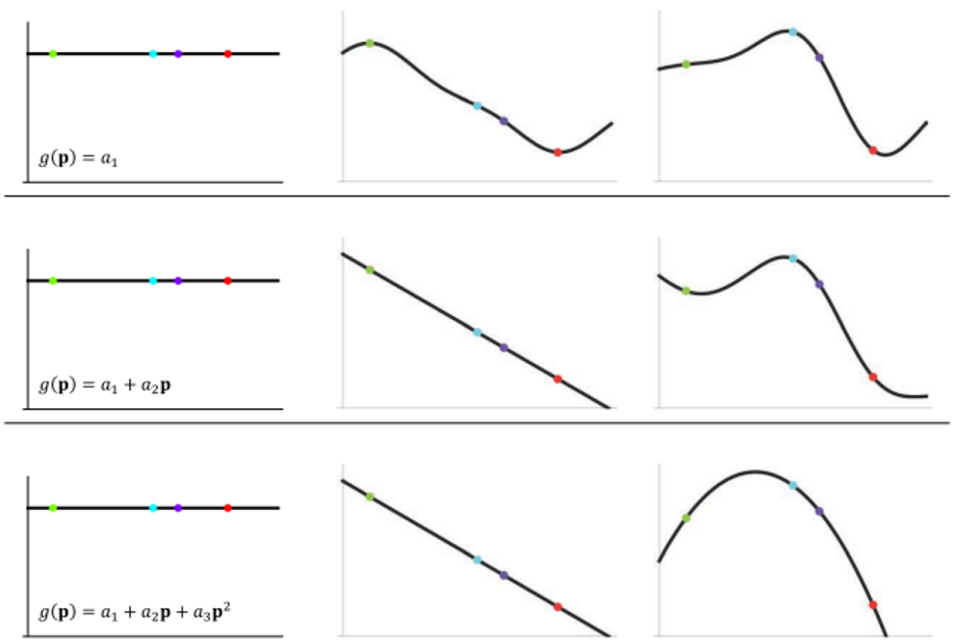


图 10.18

10.6 基于径向基函数的三维物体重建和再现

10.6.1 引言

径向基函数的插值可以应用于通过激光扫描得到的点云数据重建三维物体的表面，并生成网格和填补缺失数据区的空洞。
在之前的论文中，有人实现了这一点，但是由于计算复杂度和存储空间不够好，只适用于较小数据规模的重建。这篇论文先形式化定义了重建问题，再介绍了几种加速方法和处理噪声的方法，最后展示实际应用。

10.6.2 形式化问题

即寻找一个函数 f，其隐式定义了一个表面 (函数值为 0 的点组成的集合)，且满足采样点的函数值为 0。

10.6.3 加速方法和去噪

10.6.3.1 快速方法

实际应用中不需要无限精度, 于是在允许一定精度误差的条件下, 可以用另一篇论文中提到的快速方法进行加速计算。
首先, 我们寻找函数时允许采样点的函数值不需要精确地等于 0, 而是在 0 的某个邻域内, 对于不同的采样点可以选取不同的精度要求。
其次, 在得到函数之后, 我们进行重采样生成表面三角形网格时, 不需要选取函数值精确等于 0 的点, 而是允许重采样点的函数值在 0 的某个邻域内。
下图就是对这种允许误差存在的方法的形象描述:

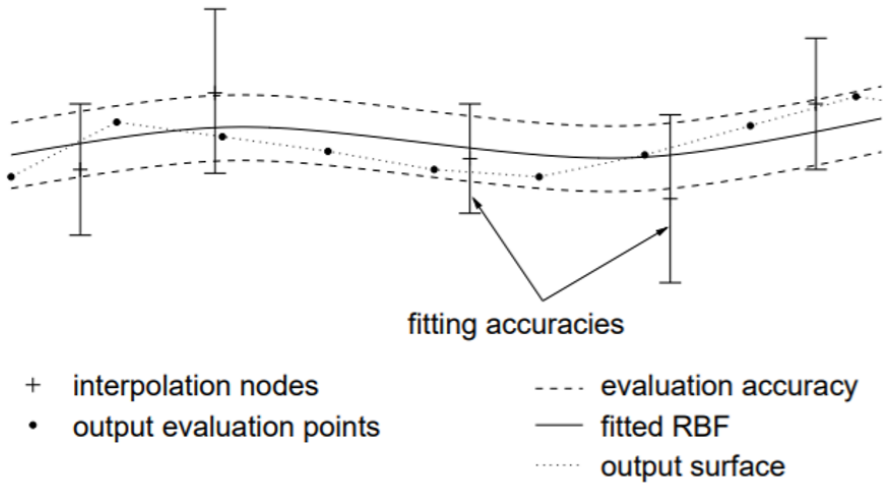


图 10.19

10.6.3.2 中心点的减少

算法的复杂度与中心的个数有关, 自然地想到减少中心点以加速计算。这里采用贪心算法, 先取所有采样点的某个子集, 进行插值, 如果剩下的点的函数值在在 0 的某个邻域内, 则认为找到了足够好的函数, 否则加入一部分点, 重复上述步骤。

10.6.3.3 处理噪声

激光扫描数据通常是多次扫描后拼合得到的, 容易产生冗余和缺失。噪声问题的处理方法是, 引入一个光滑参数 ρ, 需要求解的线性方程组变成:

$$\begin{bmatrix} A - 8N\phi\rho I & M \\ M^T & 0 \end{bmatrix} \begin{bmatrix} c \\ a \end{bmatrix} = \begin{bmatrix} v \\ 0 \end{bmatrix}$$

10.6.4 实际应用

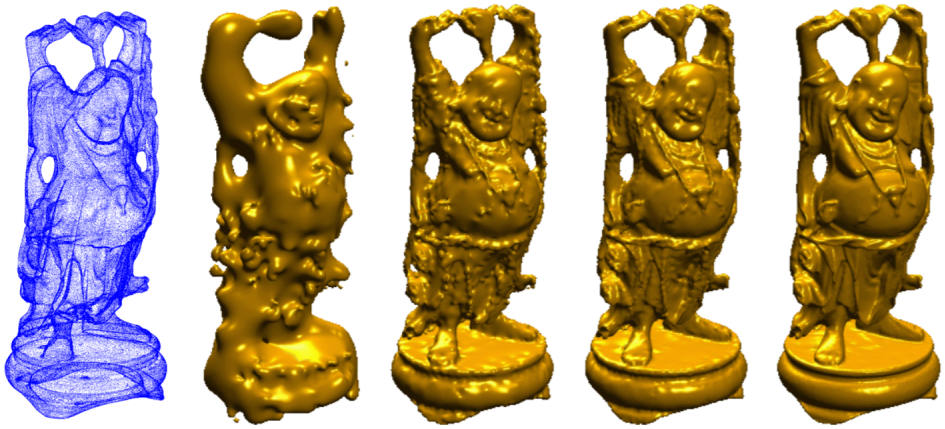
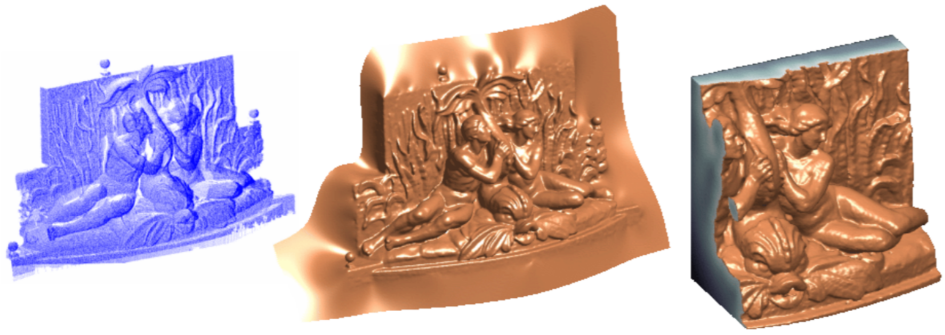


图 10.20 最左图是点云数据, 从左往右是贪心算法逐渐增加插值的采样点后的效果。



(a) (b) (c)

Figure 8: RBF approximation of noisy LIDAR data. (a) 350,000 point-cloud, (b) the smooth RBF surface approximates the original point-cloud data, (c) cut-away view illustrating the RBF distance field and the preservation of the gap between the arm and the torso.

图 10.21 左图是点云数据, 可以看到存在数据缺失区。中图是用引入光滑参数后的效果图。右图是展示插值计算得到的 RBF 函数, 颜色越深函数值越大。

(a) (b) (c)

Figure 9: (a) Exact fit, (b) medium amount of smoothing applied (the RBF approximates at data points), (c) increased smoothing.

图 10.22 左图是精确插值的结果, 中图和右图展示修改光滑参数 ρ 带来的效果。

10.7 基于不同规模局部径向基函数对 3D 散点的表面重建的办法 [OBS03]

10.7.1 引言

使用径向基函数（RBF）对散乱数据插值或拟合的方法被广泛使用。但使用局部还是全局径向基函数仍是一个艰难的选择。

- 局部 RBF：支配集半径为有限值 σ。计算更简单且快速，局部细节能更好保存，但对点的密度会更敏感。
- 全局 RBF：支配集可为整个欧式空间。计算更复杂且慢，但在修复不完整数据上更有用。

我们可以用一种层次结构在不同层用不同半径的 RBF 进行插值，以实现对表面从粗糙到精细的刻画，将局部与全局的优势集于一身。

10.7.2 单层插值

问题描述与之前相同：给定点集 $P = \{p_i | 1 \le i \le n\}$，每个点 p_i 上存储有法向量 n_i 指向平面外部，我们需要用一个函数 f 插值点集中的点以重建平面。

在层次结构中的某一层，我们使用半径固定为 σ 的局部 RBF 之和作为 f 进行插值。f 形式如下：

$f(x) = \sum_{p_i \in P} [g_i(x) + \lambda_i] \phi_\sigma(||x - p_i||)$

其中：$\phi_\sigma(x) = \phi(x/\sigma)$, $\phi(x) = (1-x)_+^4(4x-1)$

g_i, λ_i 是未知的函数和系数，需要我们依以下的方式确定；ϕ 的函数图像见 10.11：

g_i 设定为一个二次函数，其系数需被设为使加权均方误差最小的值，详细公式见下：

$$g_i = \arg\min_h \sum_{(u_j, v_j, w_j) = p_j \in P} \phi_\sigma(\|p_j - p_i\|)(w_j - h(u_j, v_j))^2 \tag{10.1}$$

$$h(u, v) = Au^2 + Bv^2 + Cuv \tag{10.2}$$

其中 A,B,C 是二次函数的系数。如此设计的目的是使 g_i 尽可能贴合点 p_i 邻域的情况，且距离 p_i 越远的点影响对 g_i 影响越小。(u, v, w) 是一个三维直角坐标系，其中 w 与 p_i 法向量方向一致，(u, v) 则为与 w 正交的平面中的任意两个彼此正交的向量。

为了使 f 能正确地插值 P 内的所有点（即对任意 $p_i \in P$，有 $f(p_j) = 0$），需要正确设定 $\lambda_1, ..., \lambda_n$ 的值，而解 λ 即为解线性方程组。当线性方程组系数矩阵正定且稀疏时使用预处理共轭梯度法可高效解决，详细细节请见 [2].

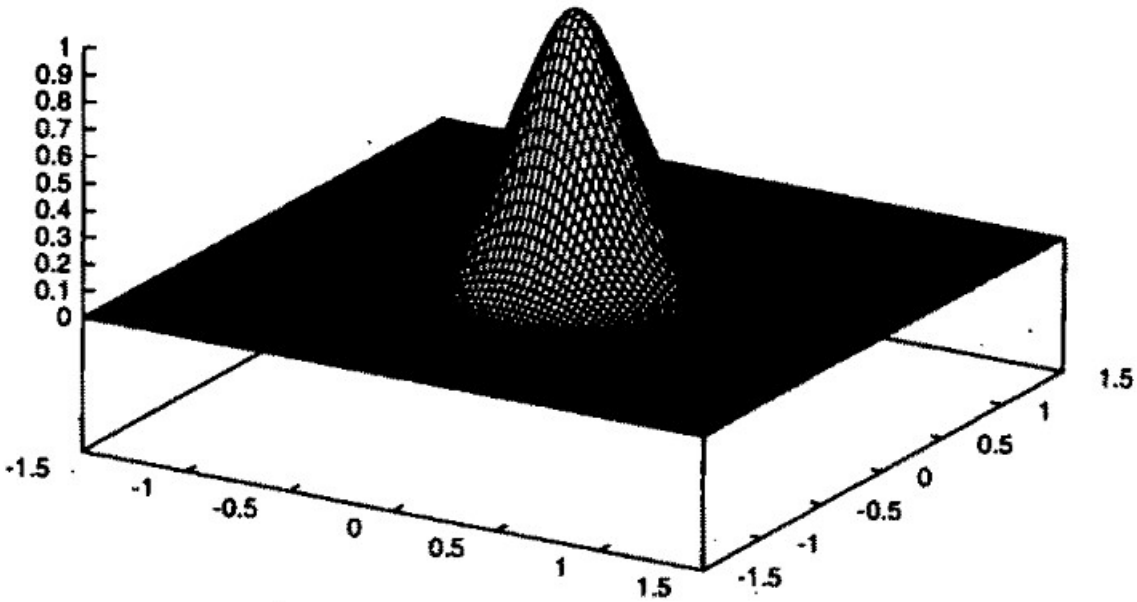


图 10.11: ϕ

我们仍不知 σ 的大小如何设定。从直觉上讲，σ 应和点密度负相关。因而论文提出一种确定 σ 的算法：使用八分树的方式将点集划分为 8 个子空间，递归进行，直至某子空间内点数 ≤ 8，此时终止划分，计算该叶子子空间中的点的直径（两点间最大距离），取 σ 为所有叶子子空间的直径平均值 $* \ c$（c 为某常数）。

10.7.3 多层插值

对全图使用相同半径的 RBF 并不合理：

- 在点稀疏的区域使用过小的半径可能无法实现正常插值。
- 在点秘籍的区域使用过大的半径可能导致局部细节被抹去，且增大计算量。

因而论文提出层次的方法：逐层插值，随着层数增加，点逐渐密集且 RBF 半径逐渐减小。

首先我们建立出不同层次需要插值的多个点集 $P_1, P_2, ..., P_m$，其中 P_k 包含八分树第 k 层所对应的最多 8^k 个子空间的质心，用于第 k 层插值。

σ_k 为第 k 层插值的 RBF 半径，其中 $\sigma_k = \sigma_{k-1}/2, \sigma_1 = cL$ (L 为第一层子空间内点的平均直径，c 为某常数）

第 1 层的插值和之前无异（即以 σ_1 为半径，在点集 P_1 上插值），得到函数 $f_1(x)$

第 k 层的插值以第 $k-1$ 层为基础，在其上做进一步修正，公式如下：

$$f_k(x) = f_{k-1}(x) + \sum_{p_i^k \in P^k} [g_i^k(x) + \lambda_i^k] \phi_{\sigma_k}(\|x - p_i^k\|) \tag{10.3}$$

而确定 g_k^i 和 λ_k^i 的方式与单层插值相同。当 $\sigma_m \leq \sigma_0$ 时（σ_0 即为依之前算法确定的平均 σ），不再构建更深的层。层次化方法实例如 10.12所示：

参考文献

[FS97] Y. Freund and R. E. Schapire, A Decision-theoretic Generalization of On-line Learning and an Application to Boosting, *Journal of Computer and System Sciences* **55** (1997), pp. 119–139.

[OBS03] Yutaka Ohtake, Alexander Belyaev and Hans-Peter Seidel, A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions, *Proc. Int. Conf. Shape Modeling and Applications (SMI)* **55** (2003), pp. 153–161.

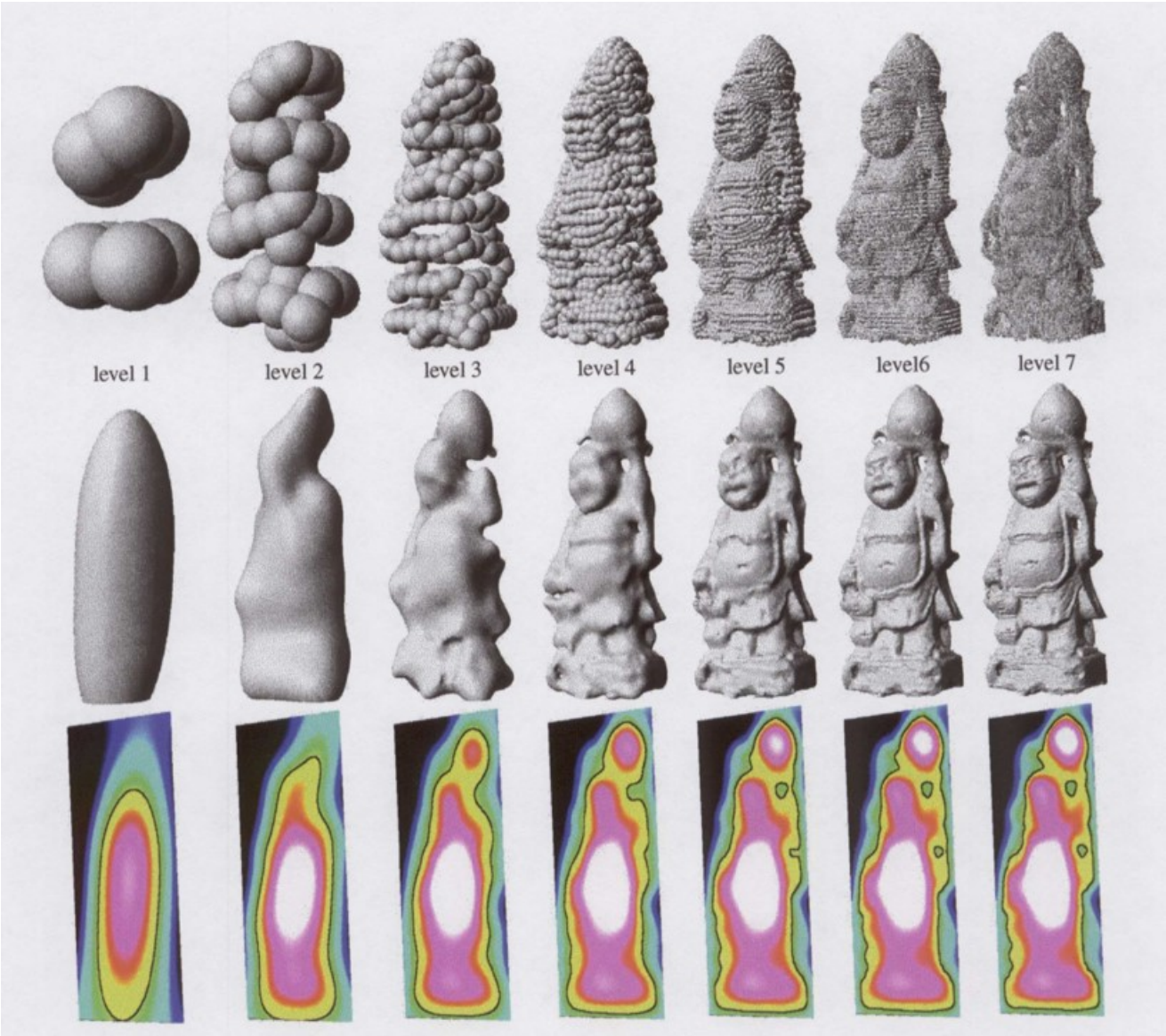


图 10.12: 第一行：各层点及其半径（为方便可视化，半径已缩小为真实值的 1/5）。后两行：各层插值出的平面的可视化。