第十一章　几何重建

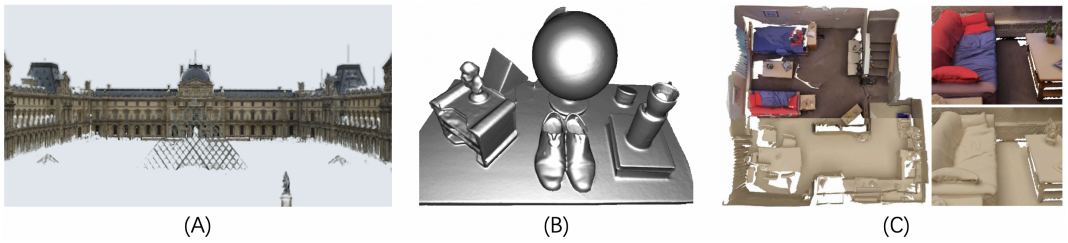

图 11.1: 现实场景的三维重建．A：COLMAP 基于彩色（RGB）图像重建室外场景．B：KinectFusion 基于深度（Depth）图像重建桌面物体．C：BundleFusion 基于 RGB-D 图像重建室内场景．

几何重建 (如图11.1) 是计算机视觉和计算机图形学中的一个经典问题，是实现虚拟现实、增强现实、和无人驾驶等技术的基础．几何重建在机器人领域中也有重要的应用价值，是实现机器人定位和导航的关键技术．在计算机视觉和图形学中，几何重建研究如何恢复目标物体（场景）的三维几何形状．

点云 (Point Cloud) 是几何重建的一种主要的三维表示方法 (3D representation)．形式上来说，点云是指一组三维空间中坐标的集合：$\{(x_i, y_i, z_i)|i = 1, 2, \ldots, N\}$ ．在几何重建中，点云数据一般直接来源于 Lidars 和 RGBD 相机，或者从 RGB 图像开始，通过一些计算机视觉的算法（例如三角化、束调整和深度学习）生成．

这一节中，我们会重点介绍处理点云这一关键步骤．将从以下这三部分展开：

1. 点云的注册
2. 点云的表面重建
3. 点云的模型拟合

11.1　点云的注册

在点云处理中，一项重要的内容就是点云注册．在几何重建中，我们往往会对同一个场景采集多组点云数据，最终需要将这些局部点云通过点云注册，如图11.2，对齐成为一个最终的完整点云．

下面我们给出一种最经典的点云注册算法：Iterative Closest Point(ICP)

11.1.1　ICP 流程

1. 通过主成分分析 PCA，**初始化**一组旋转平移变换 R, t

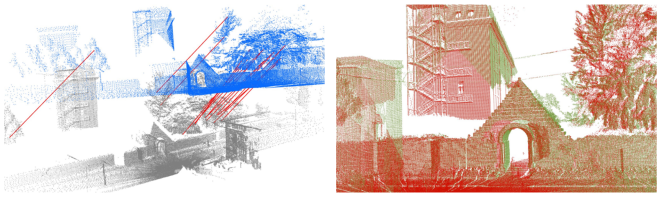

图 11.2: 左图：同一场景中的两组待对齐的点云；右图：通过 ICP 算法成功对齐后的结果

2. 将这组变换应用于其中一组点云，然后找到其中的每个点在另一组点云中的最近点，如此两两**匹配**
3. 对于相距太远（点对距离大于中位数的 k 倍）的点对，**丢弃**
4. 构造**误差函数** $E = \sum |Rp_i + t - q_i|^2$
5. 通过奇异值分解 SVD 来**最小化**误差
6. 重复 2-5 步直到误差小于特定阈值

下面我们将具体阐述算法中两个关键的步骤：PCA 初始化和 SVD 最小化误差.

11.1.2　通过 PCA 初始化

PCA 全称主成分分析 (Principal Component Analysis)，可以用来求解数据的“轴”.

我们考虑一组点 $p_1, ..., p_n$ 及其中心坐标 c.

构造矩阵 $P_{3\times n}$，使其第 i 列为 $p_i - c$

构造协方差矩阵 $M = P \times P^T$，M 的特征向量即代表了数据的“主成分”. 对于二维数据，M 的两个特征向量表示数据的两条轴线：对应最大特征值的那个特征向量表示，数据在这个方向上的分布最分散（数值变化最大）；对应最小特征值的那个特征向量表示，数据在这个方向上的分布最集中（数值变化最小）. 对于三维数据亦是同理.

由于这些轴都是正交的，因此我们可以通过分别对两组点云进行主成分分析，然后寻找一个变换来对齐它们的这些轴线，即能得到一组粗略的初始化变换.

具体地，对于两组点云 S, T，它们的中心分别是 c_S, c_T，首先我们将点云 S 平移 $c_T - c_S$，然后找到一个旋转矩阵 R' 使得两组点云的轴分别对齐，则最终得到的变换就是：

$$R = R', t = c_T - R' \times c_S$$

即，对于 S 中的每个点 p_i，我们都可以通过应用这个变换得到对齐后的位置：

$$p_i' = R \times p_i + t = c_T + R \times (p_i - c_S)$$

11.1.3　通过 SVD 最小化误差

当我们已经找到了一系列的点对 (p_i, q_i) 时（其中 p_i 来自点云 S，q_i 来自点云 T），如何找到一个更好的 R, t 来最小化误差函数？这就需要用到 SVD 分解：

构造矩阵 P，使得其第 i 列为 $p_i - c_S$

构造矩阵 Q，使得其第 i 列为 $q_i - c_T$

然后构造协方差矩阵：

$$M = P \times Q^T$$

计算 SVD：

$$M = U \times \Sigma \times V^T$$

则旋转矩阵就是：

$$R = V \times U^T$$

最终的变换为：

$$R = V \times U^T, t = c_T - R \times c_S$$

11.1.4　相关资料

我们可以找到一些可视化的例子，例如这个链接是一个 ICP 变种算法的可视化：`https://laempy.github.io/pyoints/tutorials/icp.html#References`；在经典的 ICP 算法之外，也有各种各样在效率和稳定性上的改进，这部分内容可以参考资料`https://gfx.cs.princeton.edu/proj/iccv05_course/iccv05_icp_gr.ppt`的介绍.

11.2　点云的表面重建

有时，点云会具有一些额外信息，例如每个点的法向量和颜色．图11.3为北京大学静园五院大门处重建得到的三维点云．这些点具有位置信息和颜色信息，因此可以被直接渲染成为左侧所示的图像．该点云包含超过 10,000 个点，模型大小大约 150MB．然而对于点云来说，这样的数量也远远称不上庞大．在图像中仍然存在明显的、因为点云不够致密而产生的孔隙．

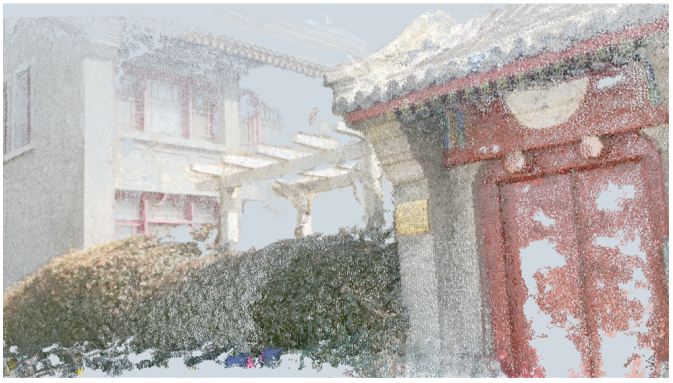

图 11.3: 北京大学静园五院大门处重建得到的三维点云

尽管对于几何重建而言，点云这种表示方式是易于处理和分析的，但是正如例子所示，即使一个并不够精致的点云模型也将占用很大的存储空间；而且数以万计的三维点也给我们编辑调整三维模型带来了巨大的困难．再者，随着我们将模型放大，点云的渲染质量将会迅速下降：点会变得稀疏，因而在物体表面露出孔隙．

因此，点云通常经过表面重建 (Surface Reconstruction) 的方法转化为网格 (Mesh) 的形式来表示．有时点云的表面重建也称作网格化 (Meshing)．我们将介绍两种主要的表面重建方法：

1. 德劳内三角剖分 (Delaunay Triangulation)
2. 泊松表面重建 (Poisson Surface Reconstruction)

11.2.1　德劳内三角剖分

在三维空间中,三角剖分 (Triangulation) 是指将点云转化为三角形网格 (Triangle Mesh) 的过程．如图11.4，在给定的点集上构建三角形网格，并没有唯一的方法．

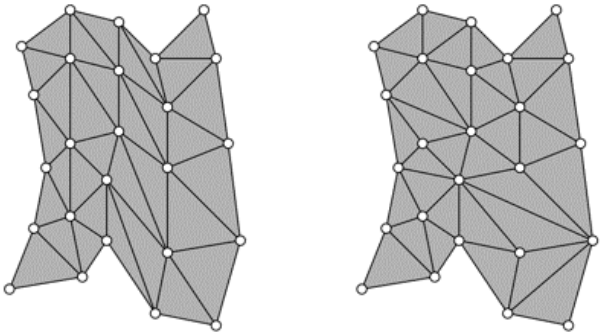

图 11.4: 对同样点集进行不同的三角剖分

倘若我们希望有一种相对“合理”的三角剖分方法，就需要提出某种标准来评价三角剖分的质量．一种直观的想法就是：尽量避免生成过于狭长的三角形．德劳内三角剖分就适用于此，它能够找到一种方法，最大化所有三角形中最小的角．

简便起见，我们以二维情况下的德劳内三角剖分为例．严格意义上来说，德劳内三角剖分并不是一种算法，而是指一类满足了特定条件的三角剖分．这个条件就是：生成结果中，任何三角形的外接圆内，都不包含任何点．可以证明，这样的三角剖分最大化了“所有三角形中最小的内角”.

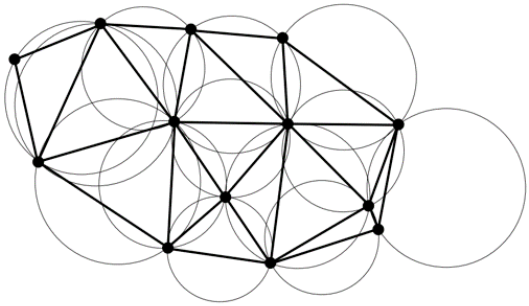

图 11.5: 平面上十三个点的德劳内三角剖分结果，通过画出每个三角形的外接圆，可以看到任何一个外接圆内都不包含点．

根据德劳内三角剖分的性质，结合三角形外接圆的性质，我们很容易可以得出一个准则：任何一对存在共用边的三角形（如图11.6），必须满足 $\alpha+\gamma<180\,\mathrm{deg}$，否则就不满足德劳内三角剖分的性质．此外，我们还可以得到一个推论：为了恢复这种性质，我们只需要把 BD 之间的边删去，加上 AC 之间的边．

这种“翻转”的思路就产生了一个非常简单的算法：

1. 构造任意三角剖分，然后检查共边三角形．
2. 如果不满足性质，就进行翻转，重新连接边．
3. 重复检查其他三角形对，直到所有三角形对都满足性质．

遗憾的是这种最简单的算法并不高效，可能需要检查 $\Omega(n^2)$ 次．一个可能的改进是将

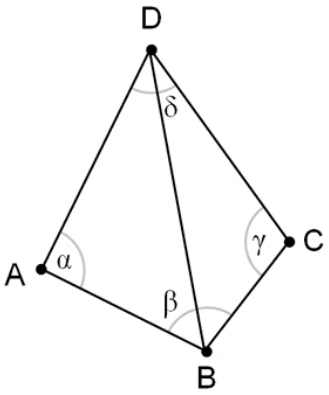

图 11.6: 共边三角形

算法改为增量式进行，每次向当前集合中添加一个点并连接一些边组成三角形，随后检查受到影响的周围的三角形是否满足德劳内性质，并对不满足性质的三角形进行翻转．这样的算法仍然需要检查 $O(n^2)$ 次．

这些基于翻转的算法始终难以并行化．另一种进行德劳内三角剖分的思路是采用分治算法：在这种算法中，我们每次递归地画一条线，将点集分成两个子集．为每一个子集递归地进行三角剖分，然后沿分割线合并这两组．利用一些巧妙的技巧，合并操作可以在 $O(n)$ 时间内完成，所以总的运行时间是 $Omega(n \log n)$．这种分治算法已经被证明是最快的德劳内三角剖分方法，如感兴趣可以参见 `https://link.springer.com/article/10.1007/BF01840356`

11.2.2　泊松表面重建

与德劳内三角剖分这种从点云直接构造三角网格的方法相比，泊松表面重建则另辟蹊径，采用了一种相对间接的方式来完成点云到网格的转换．

不难想见，德劳内三角剖分这一类直接方法存在一定的隐患：当输入的点云数据中包含噪声，或者含有一些错误的点集时，由于三角剖分必然会构造以噪点和错误点为顶点的三角形，这将不可避免地导致视觉上异常的结果．

泊松表面重建则并不直接构造三角网格，而是首先通过构建符号距离函数 (Signed Distance Function) 这一隐式表示，然后再通过行进立方体 (Marching Cubes) 算法，从 SDF 中提取出三角网格．

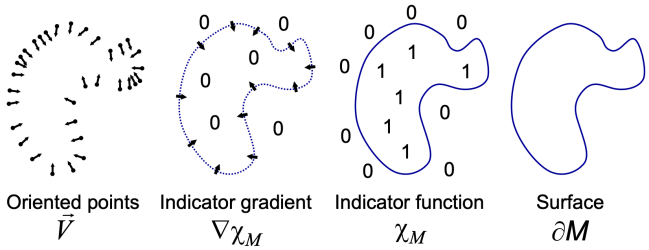

图 11.7: 泊松表面重建

具体来说，泊松表面重建基于这样一个观察：如图11.7，对于一个空间中的实体 M，如果我们定义一个指示函数 χ_M 来标识其表面 ∂M，内部的点都具有 1 的函数值，外部的点都具有 0 的函数值，那么该指示函数的梯度场 $\nabla\chi_M$ 就只在 ∂M 上具有非零梯度．而表面

上某一点梯度的方向与该点的法向量恰好反向，因此一个带有法向量的点云 $\vec{V}$ 可以被看作是指示函数梯度场在边界上的一个采样．因此，当我们可以通过 $\vec{V}$ 估计出指示函数的梯度场时，问题就转化为如何从一个函数的梯度场恢复出原函数，即从下式中估计 χ_M：

$$\nabla \chi_M = \vec{V}$$

然而由于等式右端的向量场不一定可积（例如有可能并不是无旋的），因此我们无法找到一个精确解，而需要将上式通过泊松方程转化为可以估计最小二乘解的形式：

$$\nabla \chi_M = \vec{V} \Leftrightarrow \Delta \chi_M = \nabla \cdot \vec{V}$$

最终得到指示函数的近似估计．因此，泊松表面重建依赖带有法向量信息的点云数据作为输入．算法通过八叉树 (Octree) 组织这些带法向量信息的点并在八叉树节点上定义一系列基函数，通过求解上述泊松方程得到基函数之间组合的系数，从而最终拟合出指示函数．由于使用了平滑滤波器来进行近似求解，得到的指示函数在表面附近具有类似符号距离函数 (SDF) 的性质，因此通过 Marching Cubes 算法可以从等值面中提取出三角网格，就得到了最终的网格表示．

如图11.8，泊松表面重建具有鲁棒性好、生成表面平滑的优点．如对算法细节感兴趣可以参见 `https://hhoppe.com/poissonrecon.pdf`．

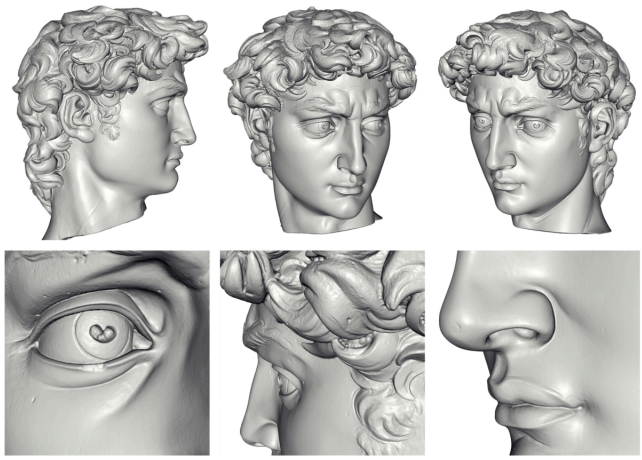

图 11.8: 泊松表面重建

11.2.3　行进立方体算法

行进立方体算法，即 Marching Cubes 算法，是一种从符号距离函数来重建表面网格的方法．

1. 首先，将空间划分为固定大小的立方体体素．然后根据每个顶点的 SDF 值，确定这个顶点在对象的内部还是外部．
2. 然后，对于每个体素立方体的八个顶点的 SDF 值正负性，我们将其编码成 $2^8 = 256$ 种情形并用一个 8-bit 二进制索引表示，如图11.9中，索引 (Vertex bit mask) 的每一位为 0 或 1 分别表示该顶点的 SDF 值小于 0 和大于等于 0．全部 256 个索引构成了 256 种拓扑关系，并可以根据对称性简化为 15 种；这 15 种拓扑关系通过旋转平移等对称性操作就可以覆盖全部 256 种情形．如图11.10．

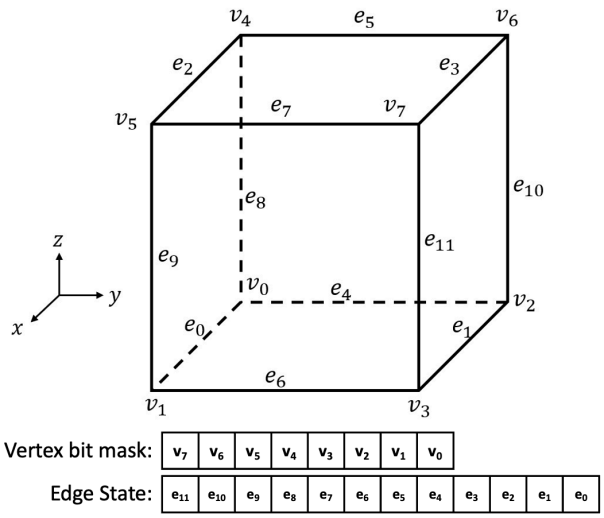

图 11.9: Marching Cubes 算法

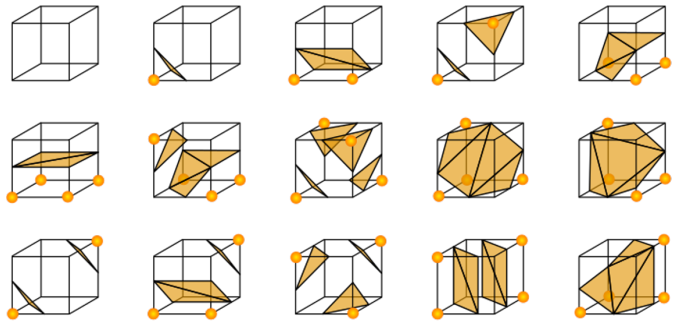

图 11.10: Marching Cubes 中 15 种拓扑关系

3. 算法维护了两个查询表．第一个表根据顶点编码成的索引查询关于边状态的编码 (Edge State)，这是一个 12-bit 的二进制串，其中每一位为 0 或 1 分别表示该边与等值面相交和不相交．根据这些信息可以在立方体的边上生成最终需要的网格顶点；这些网格顶点的位置由该边的两个端点按照 SDF 值插值得到，即该边与等值面相交的估计位置．例如边的两个端点分别具有 p_1, p_2 的位置属性和 v_1, v_2 的 SDF 值时，通过下式估计等值面（假设等值面处 SDF 值为 v^*）与边的交点，生成所需的顶点网格：

$$p^* = \frac{v_2 - v^*}{v_2 - v_1} p_1 + \frac{v^* - v_1}{v_2 - v_1} p_2$$

4. 算法维护的第二个查询表包含关于这些生成的网格顶点之间如何连接的信息；根据每个体素的拓扑关系，可以在表中查询等值面与边交点的连接方式，根据查表得到的连接方式连接对应的边，就得到了面片．
5. 最后，为每个等值面与边的交点确定法向量，法向量为边的两个顶点法向量按照同样方式进行插值的结果；而体素顶点的法向量，则由 SDF 的梯度确定：通过在 x, y, z 三个方向上计算相邻顶点 SDF 值的差，可以估计在 x, y, z 三个方向上分别的梯度，将它们组合后归一化，就得到了顶点的法向量．
6. 算法最终输出顶点的位置和法向量，以及组成顶点的网格．

11.2.4　相关可视化资料

1. 德劳内三角化的 2D 可视化例子：
   - https://cartography-playground.gitlab.io/playgrounds/triangulation-delaunay-voronoi-diagram/
   - https://travellermap.com/tmp/delaunay.htm
2. 行进立方体算法的可视化例子：https://www.willusher.io/webgl-marching-cubes/

11.3　点云的模型拟合

模型拟合是点云的另一个重要课题. 具体来说，这指的是从给定的点云中提取出平面或球体、圆柱体等几何基元的过程. 它在低层次的三维数据（无序的点集）和高层次的三维形状的结构信息之间，架起了一座桥梁，为许多下游的应用提供基础. 例如，如何从图11.11所示的点云中提取出墙面或地面的位置，就是几何拟合所需要解决的问题.

图 11.11: 北京大学治贝子园处重建得到的三维点云

几乎所有的几何形状都可以用一个确定的方程来描述它.

1. 对于实体：$F(x,y,z) \geq 0$
2. 对于表面：$F(x,y,z) = 0$

因此，我们可以形式化地定义几何拟合问题（以表面为例）：对于给定的点云 $P = \{(x_i,y_i,z_i)|i = 1,2,\ldots,N\}$ 和给定的形状 F，找到一组 P 的子集 $P' \subset P$，使得 $\forall (x,y,z) \in P', F(x,y,z) = 0$.

由于实际上不可能做到完美的拟合，因此我们往往定义一个误差函数，然后求解一组点云的子集，使得误差最小化.

11.3.1　平面的拟合

首先我们考虑一个最简单的例子，以平面的拟合为例，输入的点云也基本在同一个平面上，即不存在任何噪点、错误点和无关点. 给出三维空间中平面的方程：

$$Ax + By + Cz + D = 0$$

我们需要求解 A, B, C, D.

对于点云中的某个点 (x_i, y_i, z_i)，我们定义误差：

$$L_i = (Ax_i + By_i + Cz_i + D)^2$$

则总误差：

$$L = \sum_{i=1}^{N} L_i = \sum_{i=1}^{N} (Ax_i + By_i + Cz_i + D)^2$$

写成矩阵的形式：

$$P = \begin{bmatrix} x_1 & y_1 & z_1 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & z_N & 1 \end{bmatrix}, M = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}, L = \|PM\|_2$$

使用奇异值分解（SVD）解最小二乘拟合问题：

$$P = U\Sigma V^T$$

分解得到的矩阵 V 最后一列即为所求的 M，它将使误差 L 最小化.

实际中遇到的问题远比我们刚才遇到的更加复杂. 就像治贝子园的例子那样，给定的点集中可能不全是待求平面上的点，而有很大一部分其他点集（例如建筑的房檐、墙檐，门前的孔子像等等）. 这就使得刚才使用的最小二乘拟合失效了，因为最小二乘拟合对于误差越大的数据越敏感；当完全不在墙面上的点参与到平面参数的估计中时，最小二乘得到的结果也会被极大地影响.

此时，我们不得不使用新的一类方法，它们能够很好地利用采样一致性 (Sample Consensus) 来完成对房檐、墙檐等无关点的过滤. 我们将讨论其中最负盛名的一种：随机抽样一致性算法 (RANSAC).

11.3.2　随机抽样一致性算法

RANSAC 算法基于这样的一个思想：

假设我们希望估计刚才的点云中墙面的位置，而待求墙面上的点，已知其数量至少占整个点云的 30%，而且点云中不存在拥有更多点的平面.

现在我们每次随机地挑选点云中的三个点（三个点就能确定一个平面），然后估算平面的位置，并且观察整个点云是否至少有 40% 的点落在这个平面上（如果满足，说明我们找到了墙面）.

只要我们尝试足够多次，总存在某一次随机挑选的三个点，恰好都在墙面上；而且我们发现，至少有 30% 的点落在其上. 这样一来我们就确定，找到了墙面的位置.

从上述过程可以看出，即使存在大量（70%）的无关点，也仍然能够利用随机抽样的方式，来对点云进行几何拟合.

TIPS：事实上，这种“恰好”的概率是可以估算的. 在本例中，恰好挑选了三个墙面上的点，其概率为 $0.3^3 = 2.7\%$；只要重复 200 次抽样，就有 99.5% 以上的概率能恰好选到.

下面我们系统地描述 RANSAC 算法的流程：

1. 随机选择一个用于估算的最小子集（对于估计平面来说，该子集包含 3 个点）

2. 用这个子集估算待求的参数

3. 用估算的参数测试整个数据集，将误差在一定范围内的数据点作为“内点”(inliers)

4. 如果内点的数量超过一定比例，就以全部内点为输入，使用最小二乘法重新估计一组更精确的待求参数

5. 重复 1-4 步若干次，最后将“用最多内点估计出的参数”作为最终结果.

RANSAC 算法具有鲁棒性好、效率高的特点（不难看出它可以并行化执行），因此长久以来是计算机图形学和计算机视觉中广泛应用的拟合手段.

11.3.3 相关可视化资料

模型拟合的可视化动画:https://github.com/leomariga/pyRANSAC-3D/tree/Animations