血管的三维重建

摘要

地球上有众多的生物体，在这些生物体内有非常多的组织、器官，正是它们的正常运转维持了生物体的日常生活需求。因此，这些组织和器官起着尤为重要的作用。而通过了解这些组织、器官是以何种形态存在以及我们能否在生物体外建立起他们的三维立体模型，则需要通过切片技术实现。

在本论文中将样本进行染色处理后利用切片机技术切割成均匀薄片，以此为观察样本。题中给出所得切片在显微镜下的图像及三维坐标，需要求取对应中轴线及半径、绘制投影平面图以及重新模拟切片。因为假设血管为一特殊管道，其表面是由球体沿一固定曲线滚动包络而成，由球体各个截面均为圆形，且球半径所在平面为切割最大平面这一特性可得，题中给出切片图像内必定存在一内切圆，半径即为球半径。而球体在滚动过程中，球心运动轨迹即为所求中轴线。得到中轴线后直接绘制即可得到三维投影图。

对于血管的三维重建问题，通过二分法原理绘制切片内部最大最小圆进行同时逼近，当最大最小圆重合时，得到的圆形即为存在于不规则切片图形上最大圆。为确定所的图形全部处于切片内部，从而符合最大内切圆定义，采用了蒙地卡罗取样算法，利用分层取样得到八个样本进行分析，最终确定最大内切圆位置。编写程序求得每张切片对应半径，并进行比较迭代，最终得到 100 张切片中最大半径值，即为球半径。同时，确定了最大内切圆后可得知其圆心坐标，将这一系列的的点进行拟合后得到离散曲线，即为中轴线轨迹。再进行三维平面上的投影绘制，利用插值方法进行切片的重新切割，得到散点图，而后对散点图进行中轴线的绘制以及三维血管图的重建。

本论文针对血管的三维重建问题建立了模型，通过切片可以得到样本本身的三维形态图，使得我们对于样本有了更清晰直观的了解。由于样本大多为生物体内组织、器官，所以在生物学、病理学、临床学等多方面会有更大的帮助。

关键词：切片技术、二分法、蒙地卡罗取样、迭代、拟合、一次样条插值

一、问题重述

血管的三维形态重建是利用切片技术进行样本分析后根据所得数据进行的模型重建，先通过采取部分组织样本，进行简单染色处理后开始切割处理。使用切片机对处理好的样本进行连续不断地平行切割，得到了数十、成百的厚约$1\mu m$的平行切片，再利用显微镜进行依次逐片观察。通过观察我们可以清晰地看到所观察样本的形态，再进行下一步的研究。

现对一血管样本进行处理观察，并需要通过题中所给的切片图像进行逆向血管的三维重建。题中给出了 100 张垂直于坐标系 Z 轴进行切割的切片平面图及其坐标，现假设如下：

假设该血管可视为一类特殊的管道，该管道的表面是由球心沿着某一曲线（称为中轴线）的球滚动包络而成，例如圆柱。且管道中轴线与每张切片有且只有一个交点；球半径固定；切片间距以及图象象素的尺寸均为 1，题中所给数据真实有效，则需要解决如下问题：

（1） 通过所给切片图求取该血管管道的中轴线及其半径。

（2） 绘制中轴线在 XY 、YZ 、ZX 平面上的投影图。

（3） 利用所求的（1）、（2）问结果进行重新模拟切片，并评价模型及结果优劣。

二、问题分析

血管的三维重建问题求解，是通过对所给切片数据图形的算法处理，根据假设情况得到，建立管道三维模型的关键是在于求取管道中轴线及其半径，从而计算机模拟建立血管模型。

2.1 问题一分析

计算管道的中轴线与半径，需要从题中所给切片数据入手计算。假设管道是以中轴线为直线，半径固定的球滚动包络形成，因此血管通道类比为球体滚动轨迹模型。球体的特质导致从任何角度进行切割得到的截面均为圆形，因此在对血管进行切割后得到的切片图像一定存在一个圆形，我们称之为切片图形的最大内切圆，它是球体滚动时进行切割所形成平面。由于对管道进行自下而上的垂直坐标系 Z 轴的切割时，所得到的圆形半径必定是小于等于实际球体半径，所以在众多的切片图像当中必定存在一个最大的圆形半径，其对应的就是球体半径。而找到最大的圆形半径则需要对切片图像进行处理，首先找到每一张切片对应的最大内切圆，再进行比较得出这些最大内切圆当中半径最大的，即为所求的球体半径。

球体还有一特质是，球心到表面的距离处处相等，因此在球体沿一直线（中轴线）包络形成血管通道时，球心的运动轨迹曲线实际就是球体所沿直线。求取中轴线只需要计算每张切片内最大内切圆的圆心坐标，再对众多坐标进行串联即可得到轨迹曲线，也就是中轴线。

2.2 问题二分析

关于中轴线在 XY 、YZ 、ZX 平面上的投影图绘制，在问题一中轴线曲线轨迹已得的基础上进行降维绘制即可。

2.3 问题三分析

对于利用结果重新模拟切片进行血管的再次切割，我们需要求出每一点在切面上形成的形状，并进行重叠即可。

三、符号说明

符号	含义	符号	含义
P	图形的点集	S	采样数
d	精度	$O(\log_2 N)$	时间复杂度
R_{max}	大圆半径	h	切片平面位置
R_{min}	小圆半径	S_i, C_i	被切面所切的球和切出的圆
R_{mid}	中间圆半径	$C(x,y,r)$	三维圆图

四、模型假设

（1） 假设血管类似于一种特殊管道，管道表面是由一个球半径固定的球沿一直线滚动包络而成。
（2） 假设进行切割时，管道中轴线与每张切片有且只有一个交点。
（3） 假设所给切片图像完整无误。
（4） 假设无异常数据需要处理。

五、模型的建立与求解

5.1 求取管道中轴线及半径

5.1.1 问题分析

血管可视为一种特殊管道，由一球体沿固定曲线（中轴线）滚动包络而成。由问题分析可知管道中轴线即为球体在滚动过程中球心坐标移动轨迹，半径为每张切片最大内切圆对应的半径最大值，因此需要求得每张切片中最大内切圆的圆心坐标及其半径。

如何对不规则的切片进行画出最大内切圆是解题关键，可以直接线性遍历每一个半径（通过一定的步长增长）来找到内切圆，因为对于每一点来说，以该点为圆心，仅存在唯一一个内切圆，我们一定可以找到它。但这样若想找到一个图形内的最大内切圆，其算法的时间复杂度很可能是三次多项式级别的，耗时巨大，我们需要寻找其他的算法。

于是我们引用二分查找来解决此问题。二分迭代法是指在一个连续或者离散的区间内，并且这个区间通常应该是有序的，在此基础上，不断地将区间一分为二，因为其有序性，我们可以很容易的将目标值与区间的中间值进行比较：如果目标值大于中间值，区间向大于中间值的方向收敛；若小于中间值，则向小方向收敛，由此便可以使区间不断逼近目标值的两侧。二分查找的时间复杂度为$O(\log_2 N)$，是对数级别的算法，在数据量比较庞大的情况下，性能远超线性算法，非常的优秀。

为进一步确定一个圆处于图形内还是图形外，我们考虑使用其他算法进行辅助。如果仍旧使用枚举，或者继续使用二分搜索算法依旧是多项式时间复杂度的。

于是我们引入蒙地卡罗方法，用取样的方式来检索圆是否在图形内。蒙地卡罗方法是一种藉由抽样来描述庞大数据的方法，我们可以认为当一个结果在每个样品中都成立时，这个结果就是成立的。在这个问题上，任取N个角度$\{\varphi_1, \varphi_2, \dots \varphi_N, \varphi \in [0, 2\pi)\}$，对于圆$C(x,y,r)$，当$(x + rcos\varphi, y + rsin\varphi) \in P(x,y)$

对任意φ都成立时，我们认为这个圆就在图形内，反之它一定在图形外。

在蒙地卡罗方法中通常有很多种取样方法，比如随机采样、抖动采用或者分层采样。由于题中数据的图形大小不大，同时图形的破碎度不是很高，我们使用分层采样，即稳定地间隔一段距离进行采样。

假设采样数为S（通常取2^n）、当前圆的半径为R，假设圆边沿像素数近似于周长，那么采样比约为$\frac{S}{2\pi R}$. 在这种方案下，整个算法的流程将提升到线性对数复杂度，约为$O(S \times Nlog_2N)$。

5.1.2 建立模型并求解

计算存在于不规则切片内部的最大内切圆需要满足以下两个条件：

（1） 内切圆内每一点都在切片内部。

（2） 所得的内切圆为切片中存在的最大内切圆。

为满足以上两个条件，引入了二分法和蒙地卡罗取样进行求取。算法思想如下：

在使用二分算法过程中，因为内切圆应是一个在图形内却又无限接近图形边沿的圆，我们用一个大圆（半径R_{max}）和一个小圆（半径R_{min}）进行迭代，若中间圆（半径R_{mid}）在图形内，那么我们扩张小圆到中间圆位置，因为不可能有内切圆半径比中间圆小了；若中间圆在图形外，那么就收缩大圆到中间圆。最终，大圆和小圆会无限的逼近图形的内切圆。

随后引进蒙地卡罗算法进行分层采样从而辅助验证所得圆形全部处于切片内部。

步骤一：求出以某点为圆心的内切圆半径

在这里先利用编程实现二分法迭代，因为二分法的终止条件是大圆和小圆重合，因此迭代结果必定是大小圆都重合在内切圆上。（源程序见附录）虽然在数学的角度上来说，大小圆是永远不会重合的，但是在计算机模拟中，我们可以取一个精度d，当$|R_{max} - R_{min}| < d$时，我们视为两圆重合，其为内切圆，停止迭代。

因为图形是作为像素离散储存的，所以我们只需要选择一个小于 1 的d值即可。

通过分析可知，对于不规则图形的每个像素点，都存在一个R_{min}与 R_{max}，我们赋予R_{max}一个足够大的初值（通常为该点到边界的最短距离），并使$R_{min} = 0$，使得小圆一定在图形内，大圆一定不在图形内，如图 1 所示。

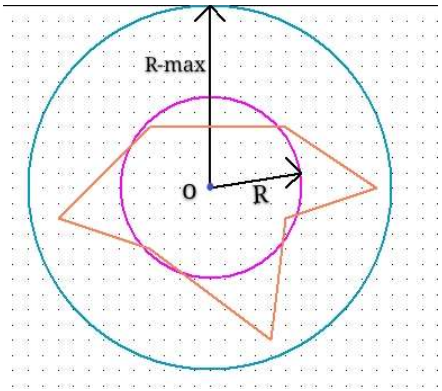

图 1 大小圆分析

在某一次迭代中，中间圆位于图形外，则大圆将收缩至中间圆位置。

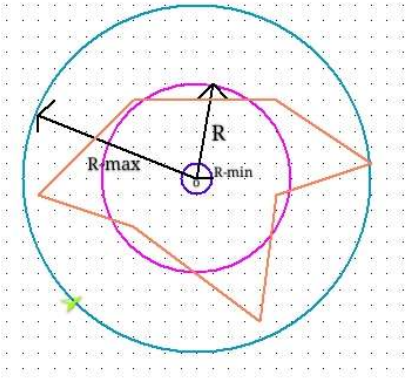

图 2 大圆进行收缩

在下一次迭代中，中间圆位于图形内，那么小圆将扩张至中间圆位置。

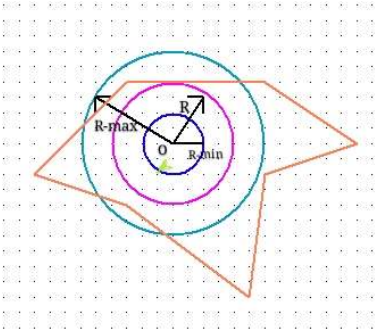

图 3 小圆进一步扩张

经过多次迭代后，$|R_{max} - R_{min}| < d$，视为两圆重合，停止迭代。

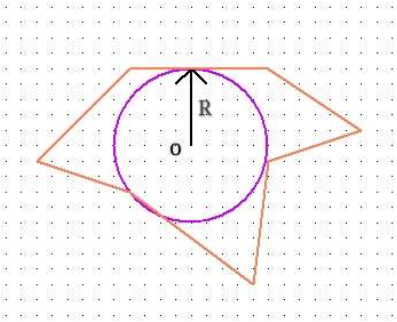

图 4 大小圆重合

对得到重合的圆形进行蒙地卡罗取样过程中，根据分层取样法选取了任意八个样本，下图 5 可以看到有部分样本不在切片内部，因此可知这种情况下的圆形并非内切圆。图 6 因所选样本均在切片内部，因此符合内切圆的标准，说明所得圆形为所求内切圆。

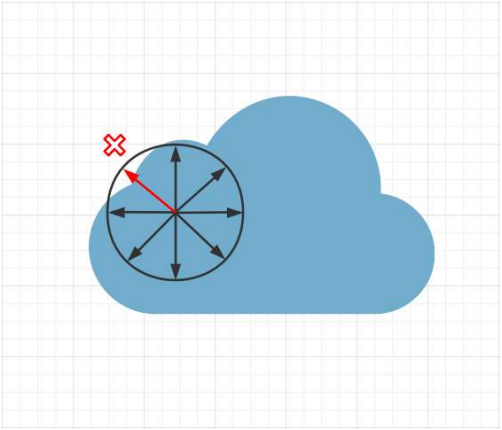

图 5 样本不在切片内部

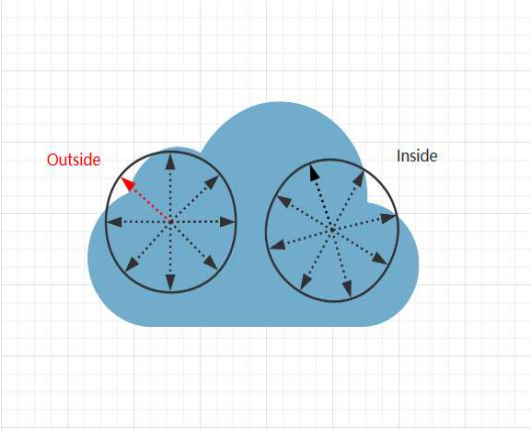

图 6 样本均在切片内部

所得结果图即为最大内切圆，依照此理可以得到其他内切圆半径。

步骤二：求解最大内切圆

以黑色轮廓内每个像素点为圆心，执行步骤一后可得该点在轮廓内的最大内切半径 R，所有像素点对应的最大内切半径里的最大值为轮廓最大内切圆半径，即为球体的最大切面。

编写程序后得到每张切片中最大内切圆的位置。图 7、8 给出了精度为 0.5 的情况下，取样 32 个角度的任意两个切片的计算结果。

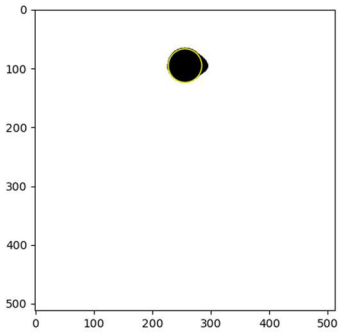

图 7 切片 1 的最大内切圆绘制

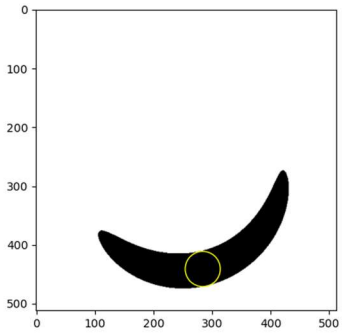

图 8 切片 100 的最大内切圆绘制

求得最大内切圆后，进行程序的编写。在程序算法中，对第一张切片下求得的半径进行存储为第一次的保存值，再对第二张切片进行计算半径，与保存的值比较。若是小于保存值则自动滤去，进行第三张切片半径求值再进行与保存值的比较，直到得到一个比保存值大的值；若是大于保存值则代替保存值。这样保证了保存值必然是最大的数值。经过计算得出球体半径R约为 30。

步骤三：求解中轴线

对每一张图片进行处理，我们将每张切片处理所得的圆心排列在空间中，得到了一个三维散点图，由问题分析可知，这些散点即为中轴线上的点，在一定的程度上，我们可以把它视为离散表示的中轴线。对坐标进行曲线拟合得到中轴线轨迹如下：

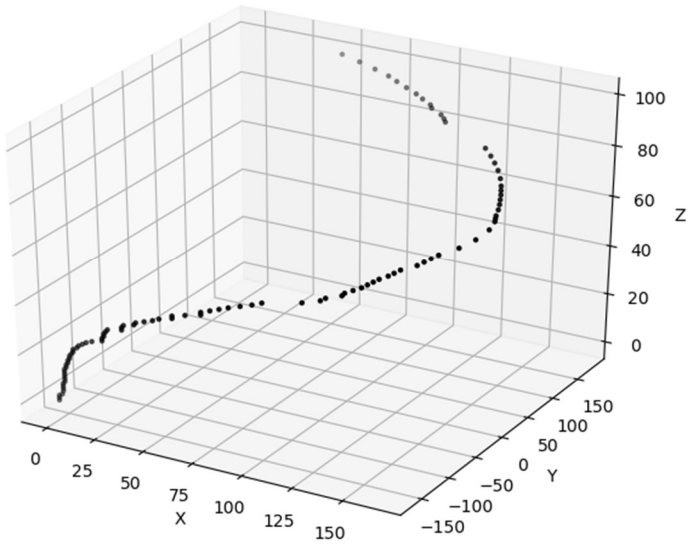

图 7 由圆心坐标绘制的中轴线

下图是另外两个视角的中轴线曲线：

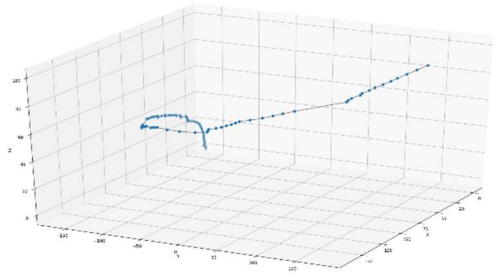

图 8 不同视角下的中轴线 1

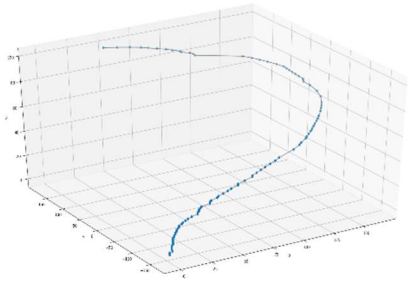

图 9 不同视角下的中轴线 2

5.1.3 结果分析

经对比我们可以发现二分法与蒙地卡罗方法组合的算法在搜索内切圆的效果十分的不错，准确度和速度都比较优秀，而生成的中轴线也较为平滑，符合自然界规律，接近问题描述和人们的现实常识。

5.2 绘制投影平面图

5.2.1 问题分析

由于第一问已得中轴线曲线，因此直接进行中轴线在 XY 、YZ 、ZX 平面上的投影图绘制即可。

5.2.2 模型建立与求解

利用软件绘图分别如下：

（1）在 XY 平面下的投影图：

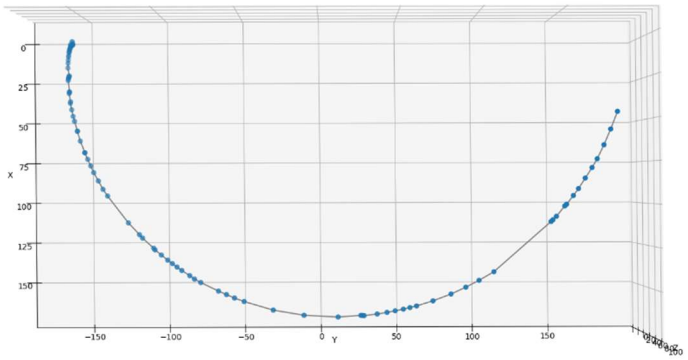

图 10 在 XY 平面下的投影图

（2）在 XZ 平面下的投影图：

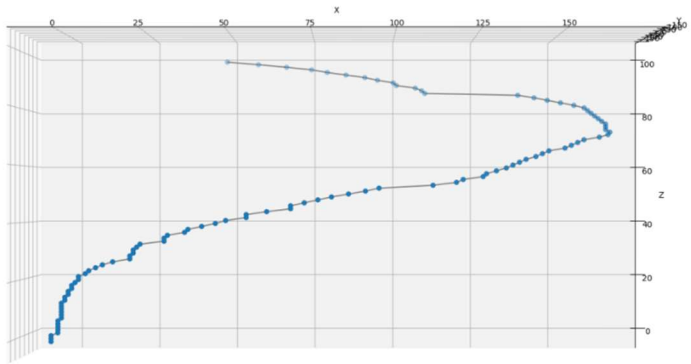

图 11 在 XZ 平面下的投影图

（3）在 YZ 平面下的投影图：

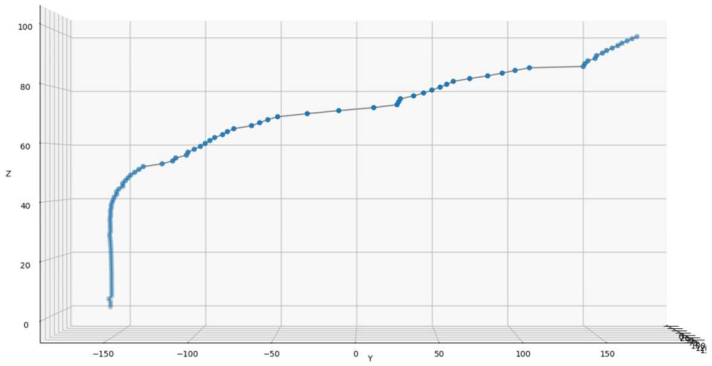

图 12 在 YZ 平面下的投影图

5.2.3 结果分析

在利用中轴线在三维空间内的轨迹进行直接投影时，并无出现太大错误，得到了中轴线分别在 XY 、YZ 、ZX 三个平面上的投影图，更有利于观察其曲线走向。

5.3 重新模拟切片

5.3.1 问题分析

在问题三中，我们需要利用结果重新模拟切片。

我们已经求得了滚动球体的半径与所有球心构成的中轴线，现在考虑切片情况：假设在$z = z_0$处做一切面，这个切面切割的球体构成一个集合，其中包含了n个不同的球体

$$S_i(x_i, y_i, z_i, r)$$
$$1 \le i \le n$$

那么那么对于集合中任何一个球体来说，S_i将在切面上留下一个圆，我们用C_i表示，而因为切割面垂直于Z轴，可得：

$$C_i(x_i, y_i, z_i, r_i)$$

其中

$$x_C = x_S$$
$$y_C = y_S$$
$$z_C = z_0$$
$$r_C \le r_S$$

现在我们需要求得切割面上的圆的半径，如图所示：

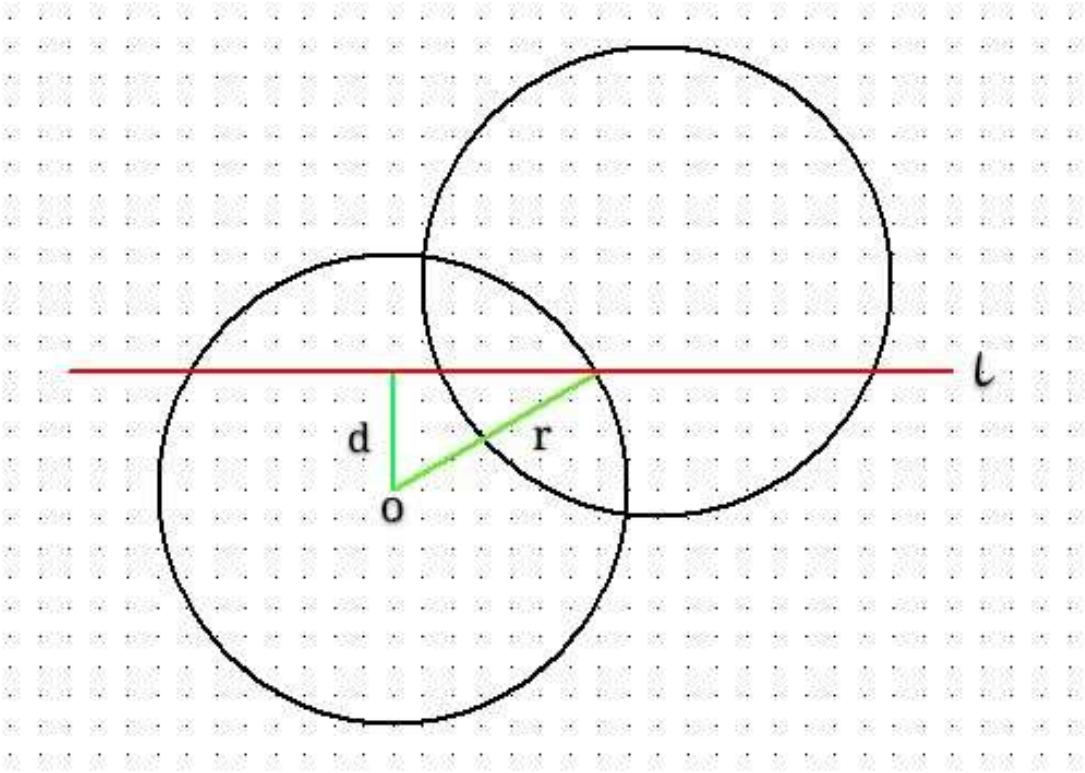

图 13 切割圆半径

可以很容易求出：

$$r_c = \sqrt{r^2 - (z_0 - z_i)^2}$$

这样我们就能对任意一个圆心，在任意一处切面上计算其留下的圆的参数，接着我们发现，当$|z_0 - z_i| > r$时，切面圆半径$r < 0$，计算无意义，所以我们当我们需要得到$z = z_0$切面的图像时，需要计算的球体集合S为：

$$S_i(x_i, y_i, z_i, r)$$

$$z_0 - r \le z_i \le z_0 + r$$

而当我们计算了所有S_i在切面上留下的圆C_i后，圆集C_i内的所有点构成的点集P，即为我们重新还原出来的切片。

5.3.2 模型建立

我们已经得到了血管在[0,99]的离散区间中的球体数据，当我们想获得任意一个z_0处的切面图像时，只需要遍历$z \in [z_0 - R, z_0 + R]$区间内的所有圆心，并且将这些园重叠起来，便可得到重新模拟的切片图像。

以下给出几个模拟结果：

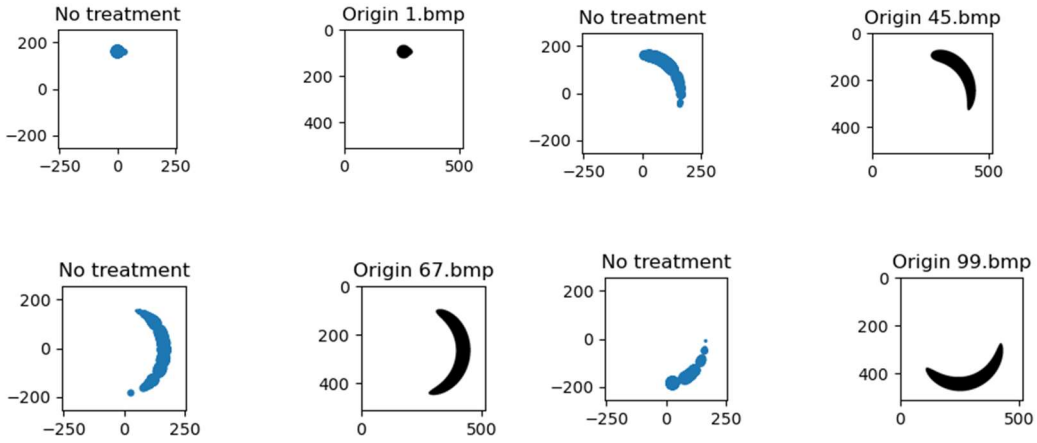

图 14 模拟切面与原图的对比

5.3.3 结果分析

利用算法，我们可以还原出血管切面图像的大致轮廓，但这样还原出来的图像同样存在着一定的缺陷，主要为以下两点：

（1）由于我们并没有 99 以上和 0 以下z值的切片图像，所以我们无从了解这些区间内的球体参数，所以根据我们推导出来的算法，在$z \ge 70$ 或$z \le 30$时，切片的形状会与原图略有出入，事实上经过验证，当z值较大时，模拟图像的确与原图有所不同。

（2）我们所用的数据为离散数据，难免会出现跨度较大的问题，因此模拟出的图像同样可能会出现一定程度的破碎，与原图有所出入。

出于这两点原因，我们需要进行一定的优化。

六、模型优缺点

6.1 模型优点

（1）该模型实用性较强，利用算法还原血管三维形态，对现实有较强的指导意义，帮助理解血管形态。

（2）建立模型的原理简单易懂，数学逻辑性强。简化了算法，并且切实可行。

（3）通过算法得到了重建后三维图的各层切片，将之与原始图片进行对比，并得到了可视化的重合度图像。

6.2 模型缺点

（1）实验中选取数据主要来自某一段血管，故模型具有一定的主观性。

（2）利用二分法法得到的拟合曲线没有经过所有的点，所以具有一定的偶然性，不能完全代表所有的数据，图像不能代表实际图像。

（3）对图像的读取得到的数据处理比较粗糙，有一定误差。

七、模型改进

在血管的三维重建过程中，我们假设了所有的数据真实有效，因此并未检查异常数据并进行处理。但在实际情况当中发现，如果有一组 XY 值都一样的圆心点，同时使用两组数据会对实验结果带来一定误差，通过分析得出应该只保留 Z 轴居中的数据进行实验。

同时因为图是离散图形，会有精度问题，所以可能出现一个切片内存在多个最大内切圆的情况。为避免大的误差出现，我们需要保证每个 XY 坐标对只有一个，而后引进插值方法，优化模型，保证实验的精准度更高。

在使用插值方法进行模型优化时，我们首先对切片本身进行了插值处理，然后对插值后得到的切片进行了对应中轴线的绘制，以及建立了血管的三维图像。优化算法如下叙述：

（1） 插值得三维中轴线：

对切片本身坐标进行处理得到了三维散点图，然后进行投影得到二维散点图，紧接着对二维散点图进行插值，得到了 $Z-X, Z-Y$ 的连续图像，利用所得连续图像重新构造出一个连续的三维中轴线。流程如下：

图 15 插值算法流程图

算法具体过程如下图所示：

在第二问中，我们求得了一组投影坐标，我们用所得的投影坐标进行插值，得到了一组连续的投影图像。

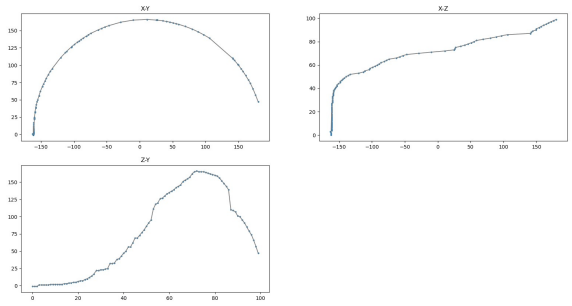

图 16 投影插值结果图

这组图像给我们提供了一组$Z-X$与$Z-Y$的坐标映射，我们便可以用这两个插值函数求出一条新的三维中轴线。

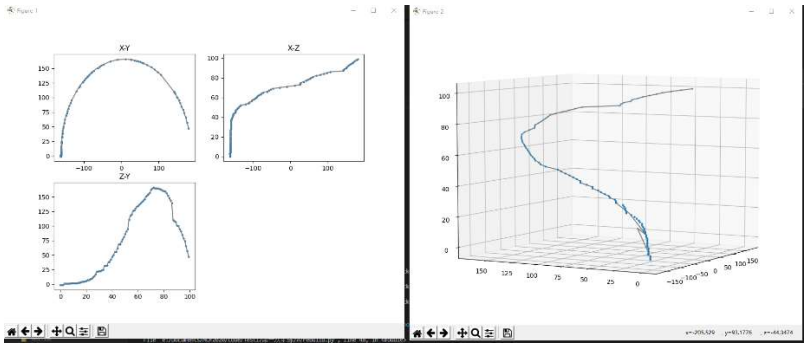

图 17 插值后重构血管效果图

（2） 重建血管管道：

有了三维中轴线和球体半径，我们可以很容易地求出整个血管的三维图像，图 18 和图 19 给出的插值前后血管的图像对比：

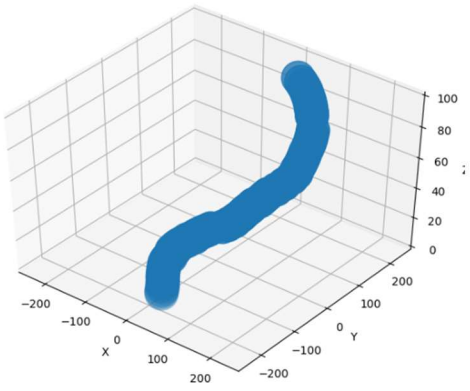

图 18 插值前三维血管图

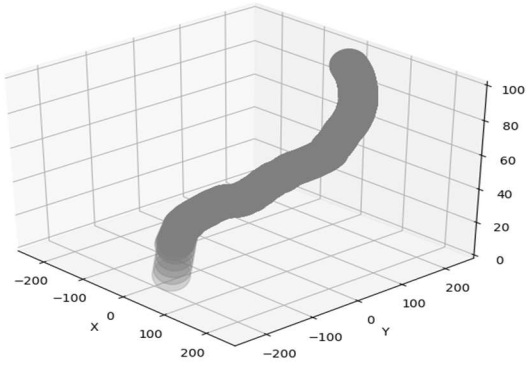

图 19 插值后三维效果图

血管的三维重建利用了插值方法进行模型的优化，弱化了若存在两组 XY 数据一样的圆心坐标点以及圆形精度本身有问题带来的实验误差。经过对切片数据的插值，重新绘制了中轴线曲线以及构建了血管三维曲线图。通过上图对比可以看出，插值前后并无明显差别，大致走向一致。但插值以后得到的曲线更加光滑，

也更为连续。

（3）重新得到切片

在之前的切片操作中，我们需要遍历$z \in [z_0 - R, z_0 + R]$区间内的所有球心坐标，在没有插值的情况下，我们只能取z为整数的坐标值，但现在我们已经拥有了一组$Z - XY$的连续函数，理论上给定任意一个z值，我们都可以求出这个z平面上的球心坐标，于是在遍历的过程中，我们可以无限细分的取值，可以很好的避免离散取值带来的图像破碎。

对切片本身插值处理后结果如图 20-23，右图为未插值的结果，下图为插值后的结果：

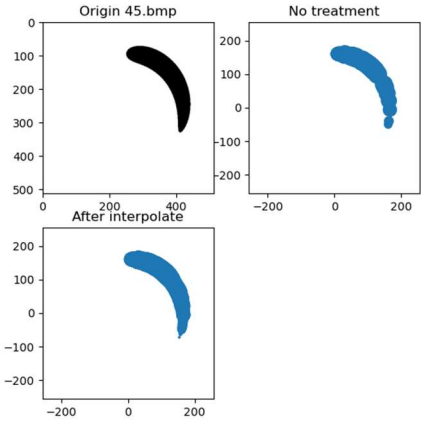

图 20 切片 45 插值前后对比图

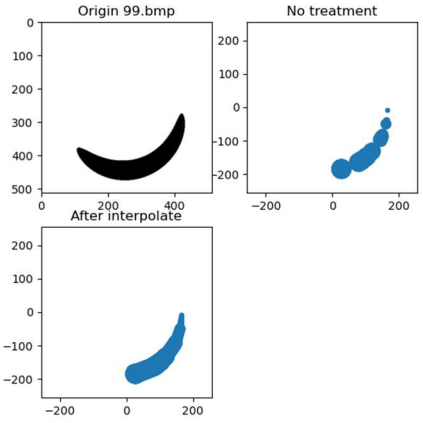

图 21 切片 99 插值前后对比图

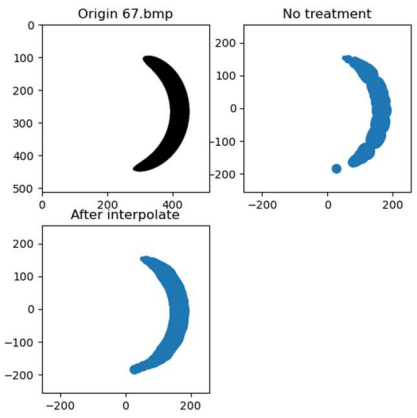

图 22 切片 67 插值前后对比图

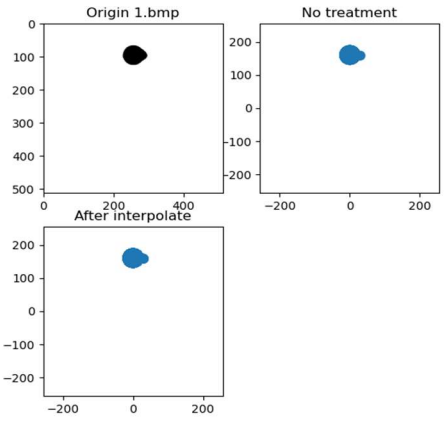

图 23 切片 1 插值前后对比图

从结果上来看，这个算法可以很好的模拟出切片的图像，并且复用性极高，可以在短时间内计算出任意$z \in \mathbb{R}, z \in [0,99]$平面的切面。

参考文献

[1] 百度百科，拟合，https://baike.baidu.com/item/拟合/7505059，2019.8.14。

[2]《现代数值计算方法》：MATLAB 版/马昌凤，林伟川编著。北京：科学出版社，2008。

[3]《MATLAB R2018a 完全自学一本通》：刘浩，韩晶编著。电子工业出版社，2019。

附录

循环代码 1：

```
"""
@Author: Hata
@Date: 2020-08-02 17:26:38
@LastEditors: Hata
@LastEditTime: 2020-08-02 17:59:53
@FilePath: \MCM2020\code\Test1\第一次训练\A 题 血管的三维重建
\circle.py
@Description:
"""


import numpy as np

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import matplotlib.patches as mpathes

from pandas import DataFrame


class InsideCircle:
    def __init__(self, pic_path: str, sample=16, precision=0.5) ->
None:
        self.bitmap = mpimg.imread(pic_path)
        self.prec = precision
        self.sample_angles = np.linspace(0, 2 * np.pi, sample)

    def isInside(self, x, y) -> bool:
        try:
            return self.bitmap[int(y), int(x), 0] == 0
        except:
            return False

    def sampling(self, x: int, y: int, r: float) -> bool:
        '''
        取样判断越界

        @param {type}
        @return: bool
        @todo: 可以尝试一下非均匀的随机取样
        '''
```

```
        # 在0到2π 间分层取样
        # sample_angles = np.linspace(0, 2 * np.pi, self.sample)
        for a in self.sample_angles:
            dx, dy = x + r * np.cos(a), y + r * np.sin(a)

            # 只要有一个点出界这个圆一定不在图形内
            # 只有所有点都处于图形内这个圆才在图形内
            if not self.isInside(dx, dy):
                return False

        return True

    def findInsideCircle(self, x: float, y: float):
        '''
        计算以一点为圆心的内接圆

        @param {type}
        @return: int
        '''
        if not self.isInside(x, y):
            return -1

        # 初始上下界，上界为图的最大像素数
        min_r, max_r = 0.0, float(np.min([x - 0, 512 - x, y - 0, 512
- y]))

        # 规定一个临界值，如果最大半径和最小半径的差小于这个临界值
时，我们视为两个值相等，不再循环。
        while np.abs(max_r - min_r) >= self.prec:

            # 每次取上下界的中间值进行检测
            mid_r = (min_r + max_r) / 2

            # 开始取样，查看以 mid_r 为半径的圆是否被包含在图形中
            # 如果包含，说明内切圆的半径一定大于等于 mid_r，mid_r 以下
的半径我们就不用检查了，于是把下界提升到 mid_r
            if self.sampling(x, y, mid_r):
                min_r = mid_r
            # 如果不包含，说明内切圆的半径一定小于等于 mid_r，于是我
们把上界降低到 mid_r
            else:
                max_r = mid_r

        # min_r 和 max_r 已经在某种意义上相等了，这就是内切圆半径
```

```
        return min_r

    def findMaxInsideCircle(self):
        res = (-1, -1, -1)
        y = 0
        for i in self.bitmap:
            x = 0
            for j in i:
                if self.isInside(x, y):
                    r = self.findInsideCircle(x, y)
                    if r > res[2]:
                        res = (x, y, r)
                x = x + 1
            y = y + 1
        return res

    def imageShow(self):
        plt.imshow(self.bitmap)


def posTranslate(pos: tuple):
    x, y = pos
    return x - 256, y - 256


if __name__ != "__main__":
    main_dir = {
        'Y': [],
        'X': [],
        'Z': [],
        'R': [],
    }
    sample = 32

    path = "code\\Test1\\第一次训练\\A\\A01bmp\\%d.bmp"
    for z in range(100):
        pic = path % z
        pixelMap = InsideCircle(pic, sample)
        x, y, r = pixelMap.findMaxInsideCircle()
        x, y = posTranslate((x, y))
        print("z = %d\nCircle center: (%d, %d)\nRadius = %f\n" % (z,
x, y, r))
        main_dir['Y'].append(y)
        main_dir['X'].append(x)
```

```
            main_dir['Z'].append(z)
            main_dir['R'].append(r)

        df = DataFrame(main_dir)
        print(df)
        df.to_csv("code\\Test1\\第一次训练
\\A\\circle_datas\\circles_s%d.csv" % sample)
else:
    sample = 32
    fig, ax = plt.subplots()
    pixelMap = InsideCircle("code\\Test1\\第一次训练
\\A\\A01bmp\\0.bmp", sample)
    pixelMap.imageShow()
    x, y, r = pixelMap.findMaxInsideCircle()
    print("Circle center: (%d, %d)\nRadius = %f\n" % (x, y, r))
    ax.add_patch(mpathes.Circle(
        (x, y), r, color='yellow', fill=False, linewidth=1))

    plt.show()
```

代码 2：

```
import numpy as np
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

from scipy import optimize

image = mpimg.imread("code\\Test1\\第一次训练\\A 题 血管的三维重建
\\A01bmp\\99.bmp")
COUNT = 256
STEP = image.shape[1] // COUNT

plt.imshow(image)

high_x, low_x = [], []
high_y, low_y = [], []


def forward(xx):
    for i in range(image.shape[0] - 1):
        if image[i, xx, 0] == 255:
            if image[i + 1, xx, 0] == 0:
                high_x.append(xx)
                high_y.append(512 - i)
```

```
        elif image[i + 1, xx, 0] == 255:
            low_x.append(xx)
            low_y.append(512 - i - 1)
            break


for i in range(COUNT):
    forward(i*STEP)

high_x = np.asarray(high_x)
high_y = np.asarray(high_y)

low_x = np.asarray(low_x)
low_y = np.asarray(low_y)


def func(x, p):
    res = 0
    j = 0
    for i in p:
        res = res + i*x**j
        j = j + 1
    return res


def dis(p, x, y):
    return func(x, p) - y

num = 4
low_p = optimize.leastsq(dis, np.ones(num), args=(low_x, low_y))[0]
high_p = optimize.leastsq(dis, np.ones(num), args=(high_x,
high_y))[0]

print(high_p)
print(low_p)

x = np.arange(0, 512)
plt.plot(x, 512-func(x, low_p))
plt.plot(x, 512-func(x, high_p))

plt.show()
```

代码 3：

```
import numpy as np
```

```
import matplotlib.pyplot as plt
import matplotlib.patches as mpathes
import matplotlib.image as mpimg

from scipy import interpolate
from pandas import DataFrame, read_csv
from matplotlib.collections import PatchCollection

def getSlice(df: DataFrame, h: int, r: int):
    selected = df[(df['Z'] < h + r) & (df['Z'] >= h -
r)].to_numpy()[:, 1:-1]

    double_r = float(r ** 2)
    patches = []

    for y, x, z in selected:
        sub_r = np.sqrt(double_r - np.abs(h - z)**2)
        patches.append(mpathes.Circle(
            (x, -y), sub_r, color='black', fill=True, linewidth=1))

    return PatchCollection(patches)


def getSliceByInter(funcs: tuple, h: int, r: int):
    fzx, fzy = funcs
    low = 0 if h - r <= 0 else h - r
    high = 99 if h + r >= 99 else h + r

    double_r = float(r ** 2)
    patches = []

    for z in np.linspace(low, high, 200):
        x, y = fzx(z), fzy(z)
        sub_r = np.sqrt(double_r - (h - z)**2)
        patches.append(mpathes.Circle(
            (x, -y), sub_r, color='black', fill=True, linewidth=1))

    return PatchCollection(patches)

def addSubPlot(fig, num, title):
    ax = fig.add_subplot(num)
    ax.set_xbound(-256, 256)
    ax.set_ybound(-256, 256)
    plt.gca().set_aspect('equal', adjustable='box')
```

```
    ax.set_title(title)
    return ax

def showSolve(p, pic):
    df, fzx, fzy = p
    path = "code\\Test1\\第一次训练\\A\\A01bmp\\%d.bmp"
    fig=plt.figure(figsize=(6, 6))

    addSubPlot(fig, 222, "No treatment").add_collection(getSlice(df,
pic, 30))

    addSubPlot(fig, 223, "After
interpolate").add_collection(getSliceByInter((fzx, fzy), pic, 30))

    addSubPlot(fig, 221, "Origin %d.bmp" %
pic).imshow(mpimg.imread(path % pic))

    plt.savefig("code\\Test1\\第一次训练\\A\\pic\\TEST-%d.png" % pic)


df = read_csv("code\\Test1\\第一次训练
\\A\\circle_datas\\circles_s32.csv")
# df = read_csv("code\\Test1\\第一次训练\\A\\circles.csv")

y, x, z = df.to_numpy().T[1:4]
fzx = interpolate.interp1d(z, x, kind='slinear')
fzy = interpolate.interp1d(z, y, kind='slinear')

if __name__ == '__main__':
    showSolve((df, fzx, fzy), 1)
    showSolve((df, fzx, fzy), 14)
    showSolve((df, fzx, fzy), 45)
    showSolve((df, fzx, fzy), 50)
    showSolve((df, fzx, fzy), 67)
    showSolve((df, fzx, fzy), 99)


plt.show()
```

代码 4：

```
import matplotlib.pyplot as plt
import numpy as np

from mpl_toolkits.mplot3d import Axes3D
```

```
from pandas import read_csv
from scipy import interpolate

def setAx(ax):
    ax.set_xlim(-256, 256)
    ax.set_ylim(-256, 256)
    ax.set_zlim(0, 100)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

ax = Axes3D(plt.figure())
df = read_csv("code\\Test1\\第一次训练\\A\\circles.csv")

ax.plot3D(df['X'], df['Y'], df['Z'], 'gray')
ax.scatter3D(df['X'], df['Y'], df['Z'], cmap='b', s=900, marker='o')

setAx(ax)

datas = read_csv("code\\Test1\\第一次训练
\\A\\circles.csv").to_numpy().T
y, x, z = datas[1:4]

yy = np.linspace(y[0], y[-1], 2000)

ax = Axes3D(plt.figure())
fyz = interpolate.interp1d(y, z, kind='slinear')
fyx = interpolate.interp1d(y, x, kind='slinear')
ax.scatter3D(fyx(yy), yy, fyz(yy), s=900, c='gray')

setAx(ax)
```

代码 5：

```
import os
import re
import random

import matplotlib.image as mpimg
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D


def sample(): return random.randint(0, 100) > 99
```

```
path = os.path.dirname(__file__) + "/A01bmp/"
bit_size, bit_color = 1, 'black'
pos_range = (-256, 256)

ax = Axes3D(plt.figure())

X, Y, Z = [], [], []

for pic in os.listdir(path):
    if not os.path.isdir(pic):
        print(pic)

        z = int(re.findall(r'\d+', pic)[0])
        y = pos_range[0]

        for i in mpimg.imread(path+pic):
            x = pos_range[0]
            for j in i:
                if j[0] == 0 and sample():
                    X.append(x)
                    Y.append(y)
                    Z.append(z)
                x = x + 1
            y = y + 1

print("Points count: %d" % len(X))

ax.scatter3D(X, Y, Z, s=bit_size, c=bit_color)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```