城市空间建模与仿真

第五讲　城市空间三维数据表达和重建-3D数据平滑与去噪

任课教师：汤圣君
建筑与城市规划学院　城市空间信息工程系

其中部分图片来自互联网和同行专家

目录

CONTENTS

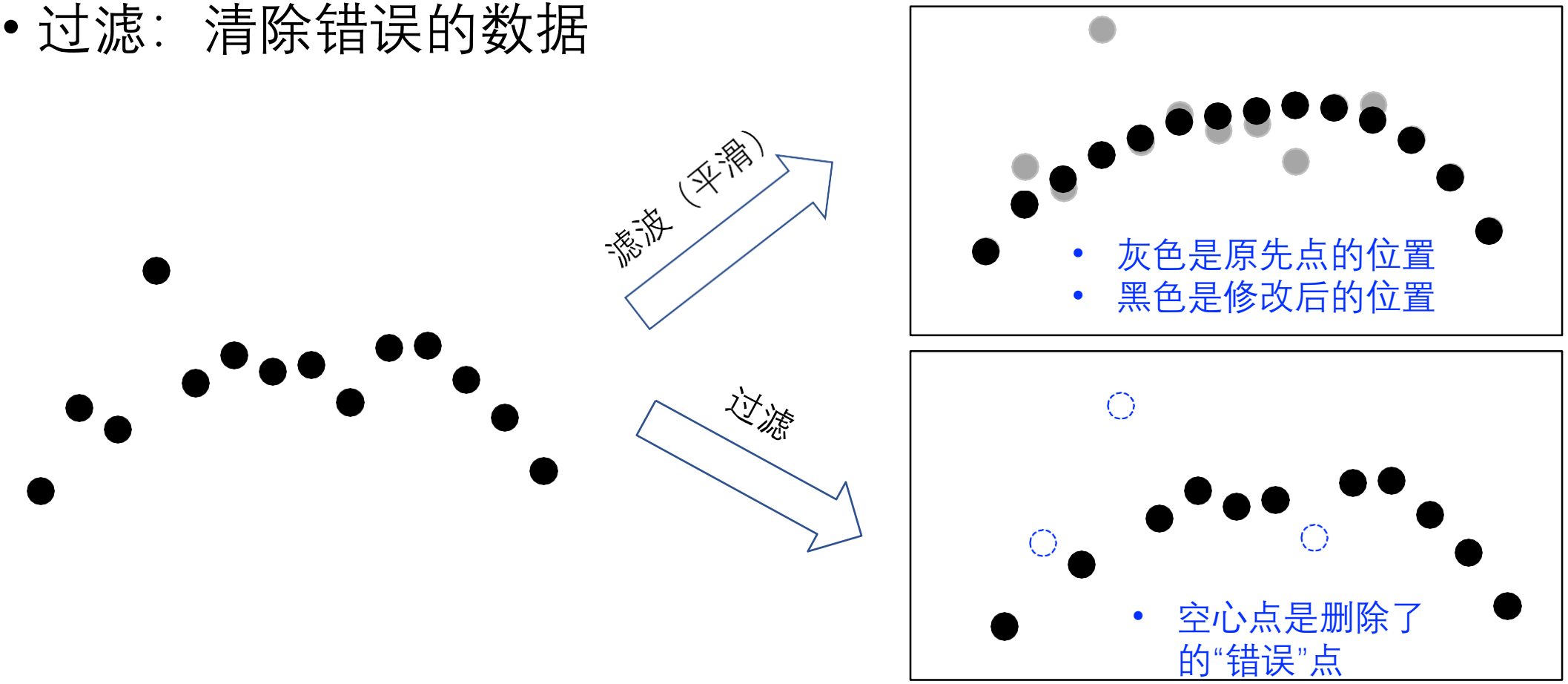
目录

CONTENTS

3D数据滤波和过滤

英文同为filter，但涉及两个容易混淆的概念

- 滤波（平滑）：叠加噪声的数据的值的修正
- 过滤：清除错误的数据



3D数据滤波

从数据源角度分类

- 基于深度图的滤波
  -  可以使用传统2D滤波算法
  -  但2D深度图的像素距离无法直接对应空间距离
- 基于点云的滤波
  -  基于3D空间的距离
  -  在无序排列的点中根据距离进行数据查询困难

从时空角度分类

- 空域滤波
- 时域滤波
- 时空域滤波

3D数据滤波

- 基于深度图的滤波

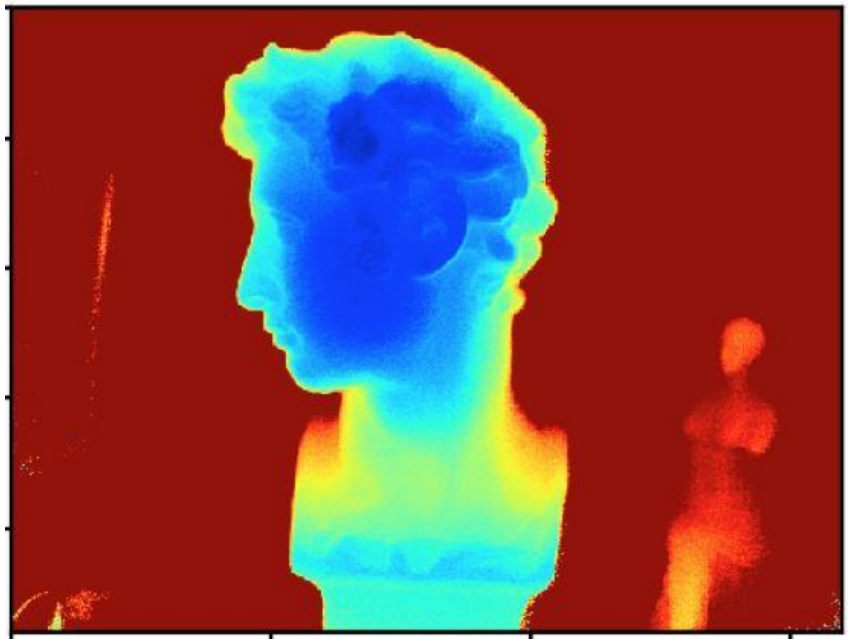
  - 可以使用多个CV的算法
  - 有2D图像处理理论支撑

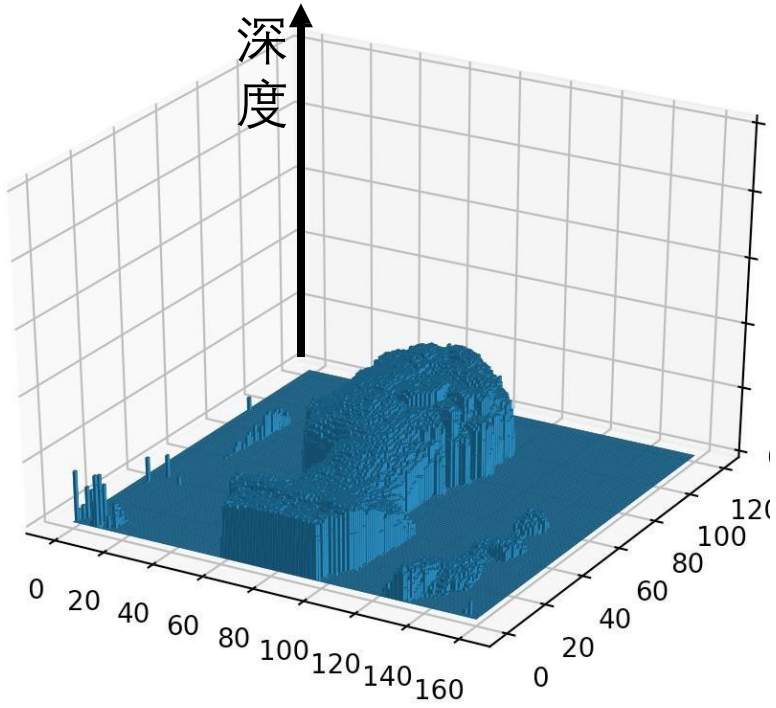
  - 深度图对物体表面的采样是非均匀的，导致传统2D滤波对他们的作用是“不均匀”的，并且在前后景交界处的失真会对后续处理带来影响

- 算法折衷
  - 注意边界的变化，不滤波甚至在后续处理时切除边界
  - 根据曲面局部法向量修改滤波核

3D数据滤波

把深度图看成2D灰度图像数据

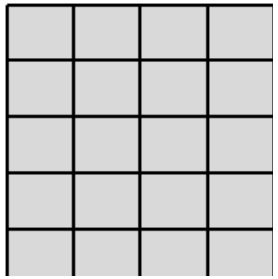




注意：在后面PPT中深度图通常以伪彩色形式出现，但滤波算法处理的是每个点的强度值

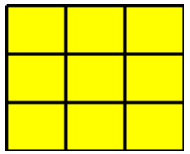
3D数据滤波和过滤

- 用2D滤波算法处理深度图，
- 下面给出2D卷积的“滑动窗口”解释



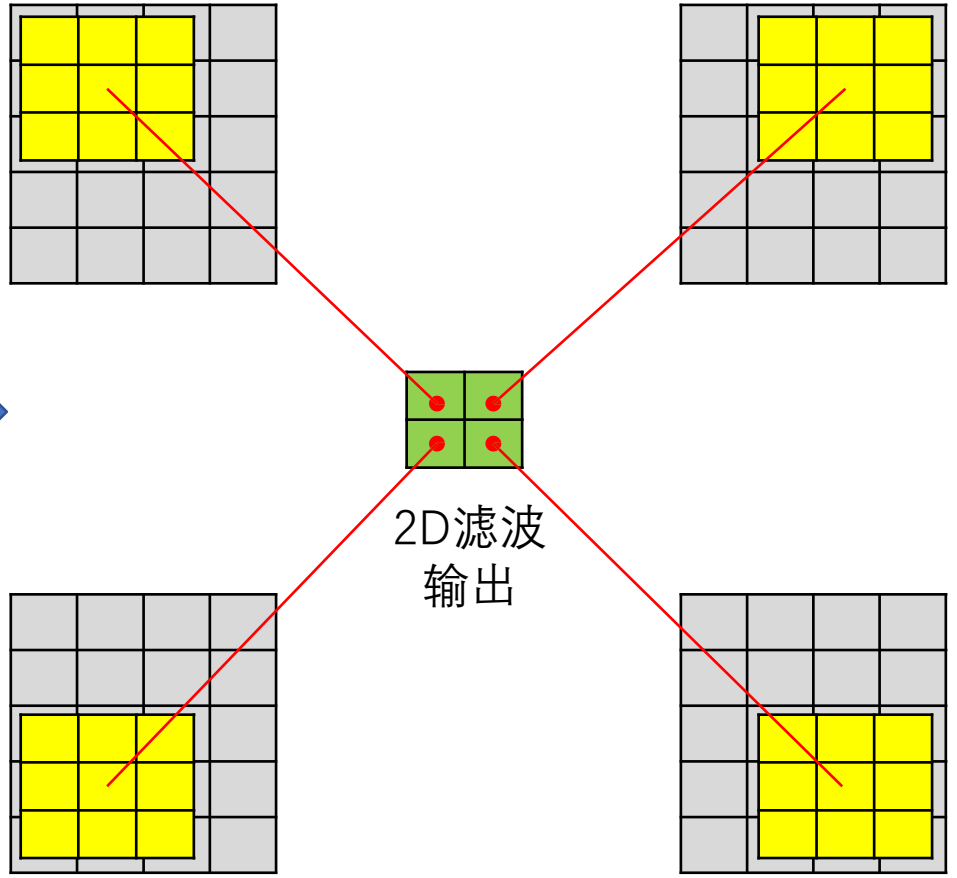
待滤波（深度）图像

滤波过程看成滤波核在图像中滑动，在滑动到的每个位置都计算一个对应的滤波输出



滤波核

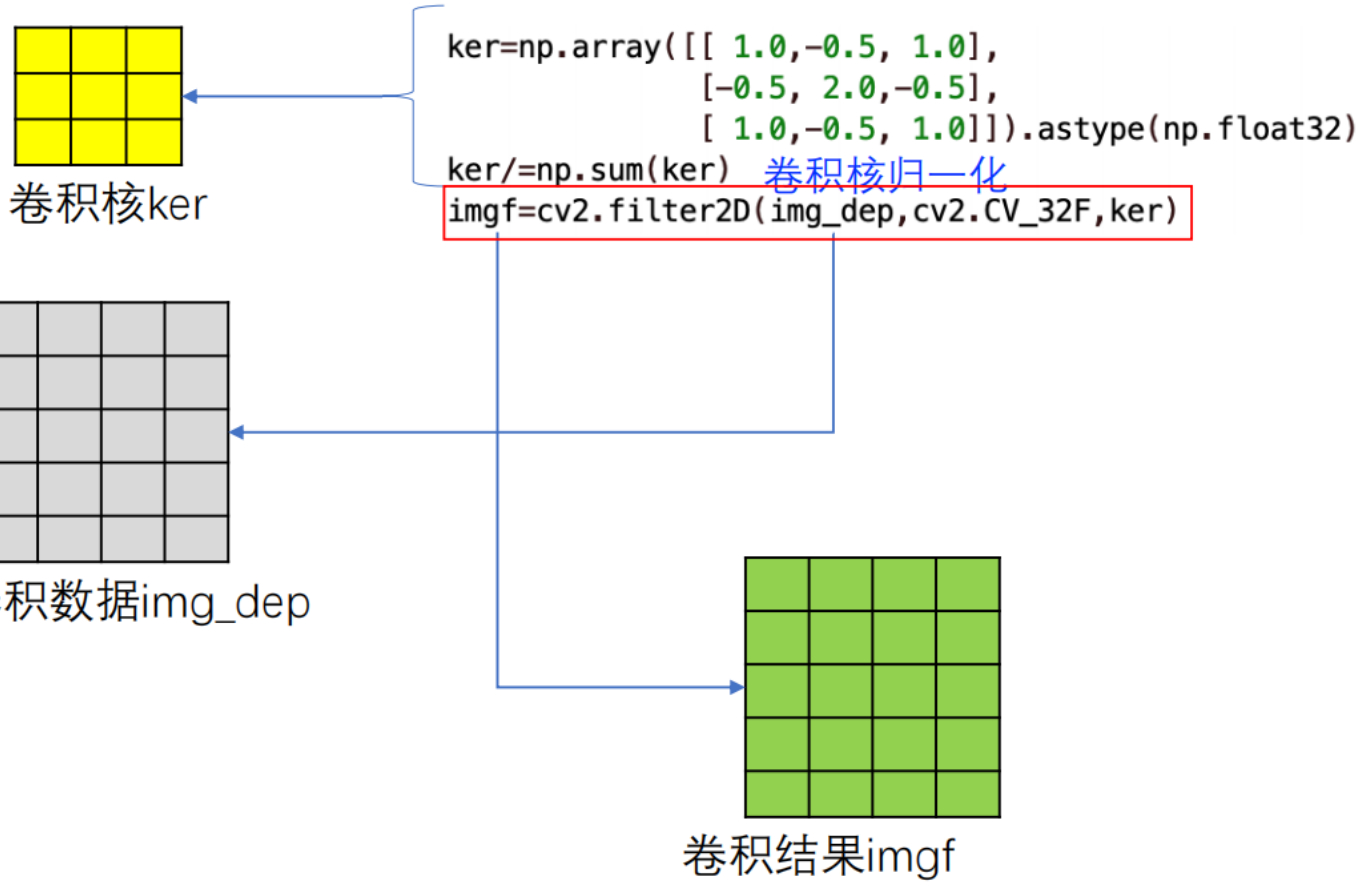
- 存放加权系数
- 通常具有归一化要求：加权系数和等于1



2D滤波过程的示意图

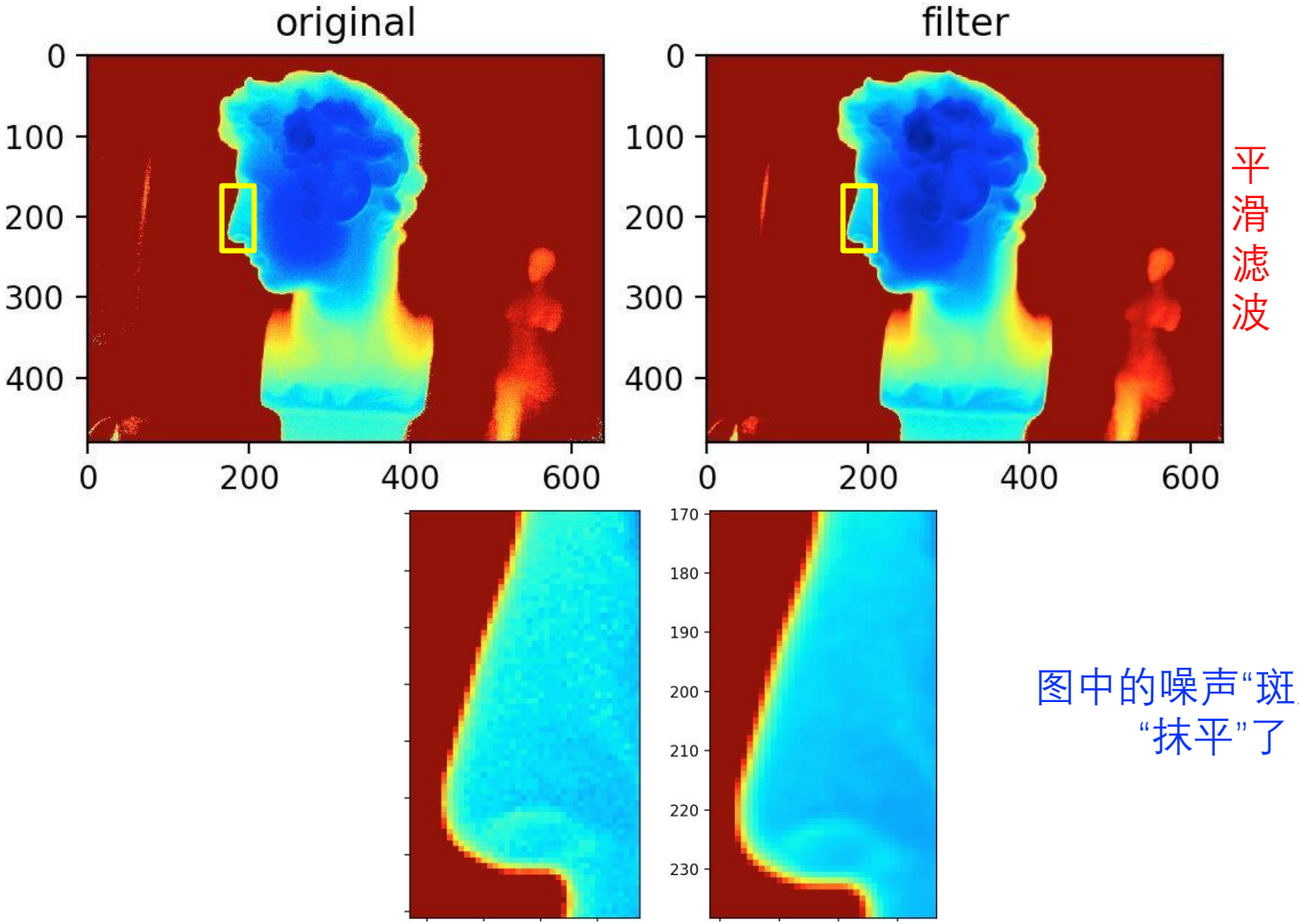
3D数据滤波和过滤

- opencv内置API实现2D滤波
- filter2D(*)



3D数据滤波

比较滤波前后的差别



平滑滤波

图中的噪声“斑点”被“抹平”了

3D数据滤波

- “方形”滤波核

$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

```
imgf=cv2.blur(img_dep,(3,3))
```

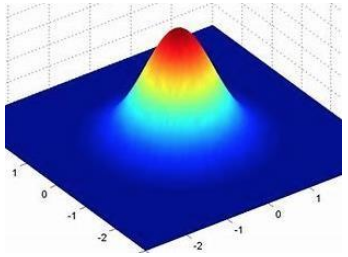
归一化核

$$K = \alpha \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ & & \cdots\cdots\cdots & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \quad \alpha = \begin{cases} \frac{1}{\texttt{ksize.width*ksize.height}} & \text{when } \texttt{normalize=true} \\ 1 & \text{otherwise} \end{cases}$$

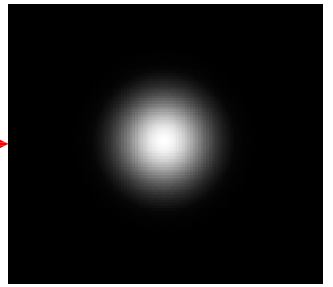
非归一化核

```
cv2.boxFilter( … )
```

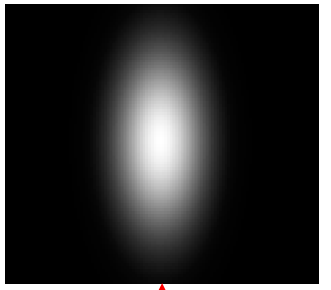
- 高斯滤波核



```
imgf=cv2.GaussianBlur(img,ksize=(127,127),sigmaX=10)
```



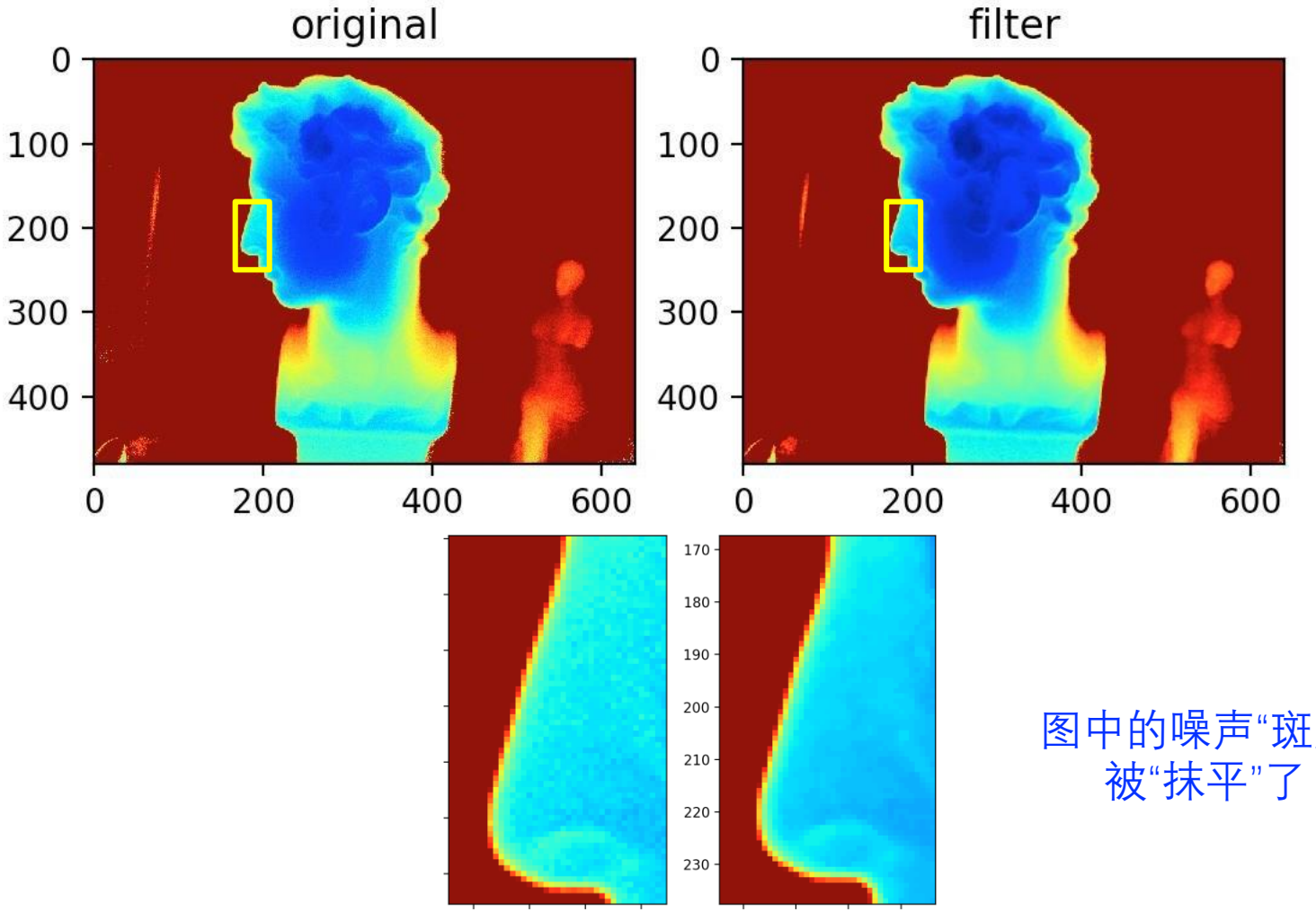
```
imgf=cv2.GaussianBlur(img,ksize=(127,127),sigmaX=10,sigmaY=20)
```



3D数据滤波

比较滤波前后的差别

```
imgf=cv2.medianBlur(img_dep,3)
```

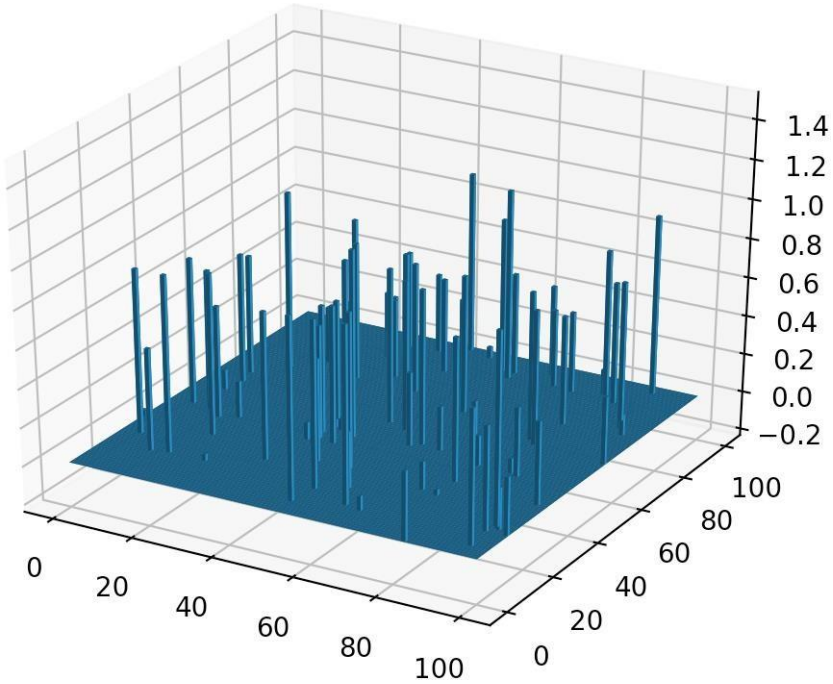


中值滤波

图中的噪声“斑点”被“抹平”了

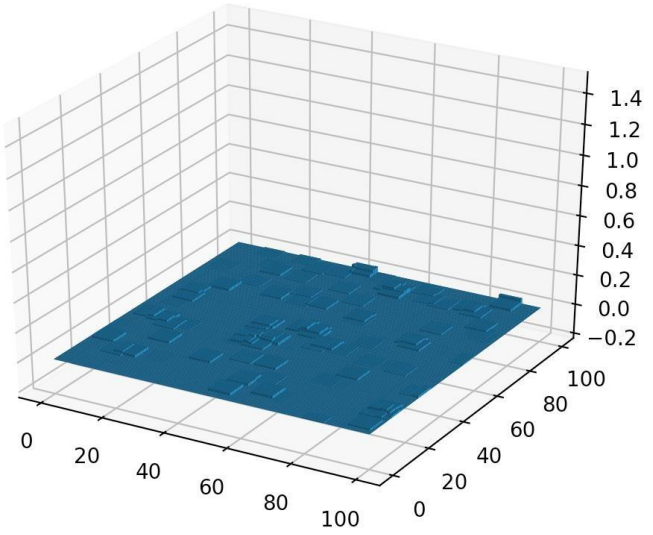
3D数据滤波和过滤

中值滤波修正深度图中的离群点



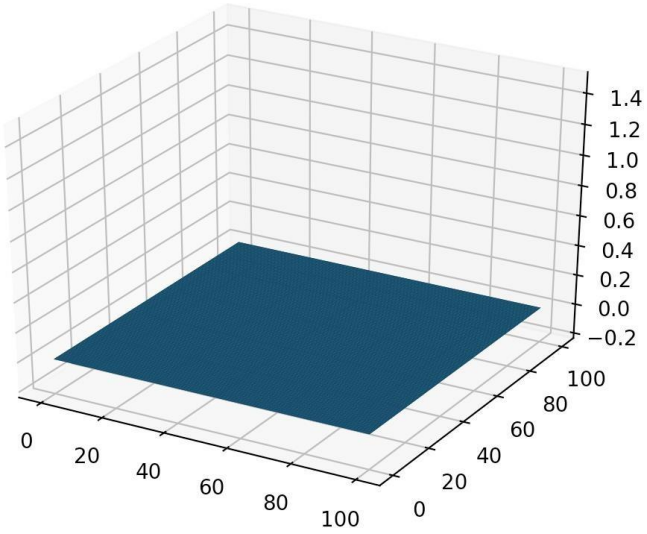
深度图上散布着“椒盐噪声”（离群点）

平滑滤波



平滑滤波降低了噪声幅度，但没有完全消除

中值滤波

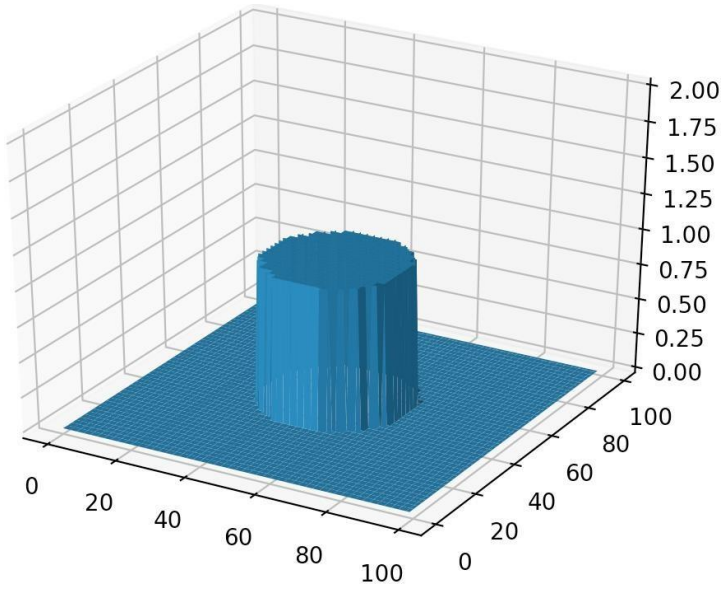


中值滤波去除了全部噪声点

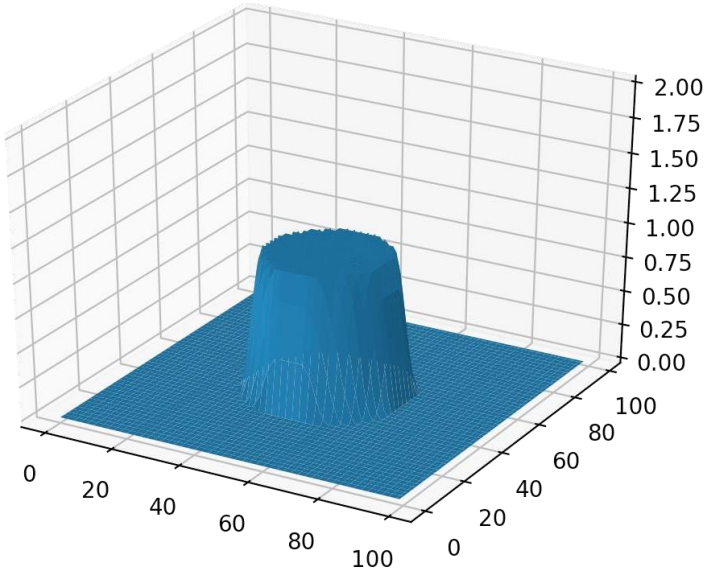
3D数据滤波

平滑滤波器

- 局部深度信息被平滑
- 不同距离邻近像素相互影响，造成点云物理距离的误差

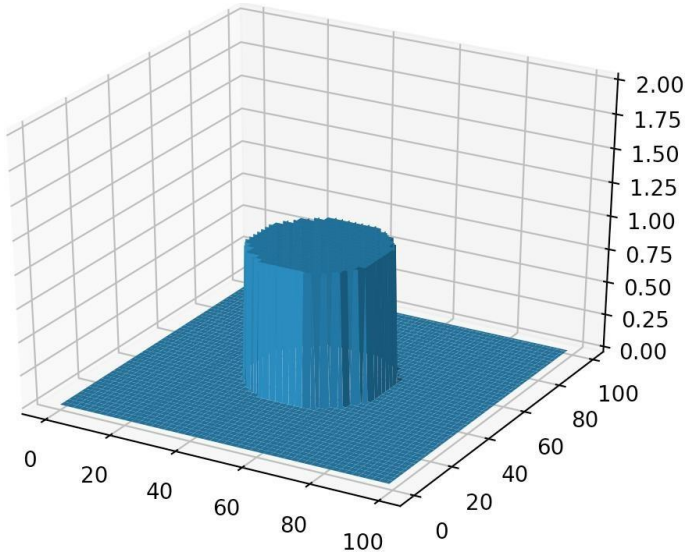


平滑滤波



平滑滤波导致边界变成“缓坡”

中值滤波



中值滤波边界保留较好

3D数据滤波-双边滤波

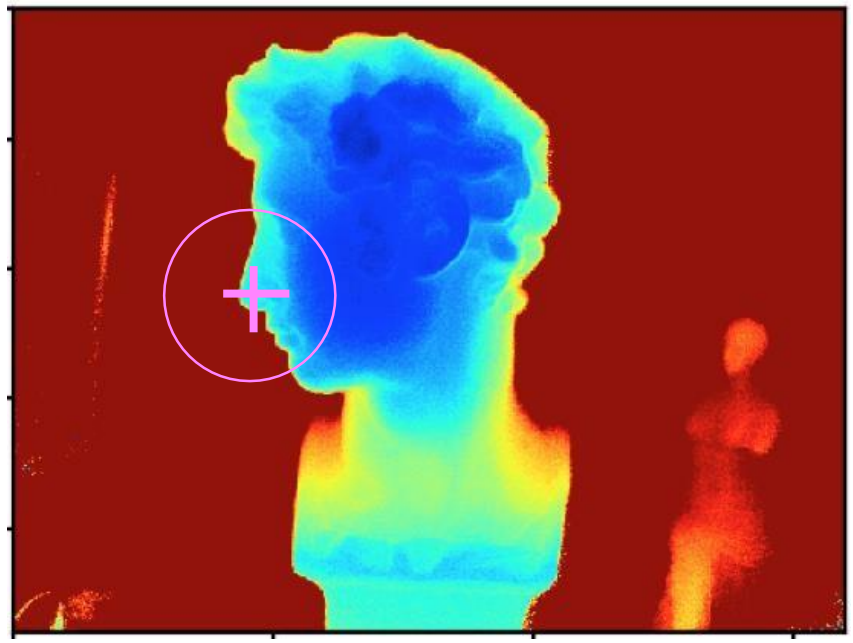
双边滤波——解决边界平滑和噪声平滑之间的矛盾

特点概述：

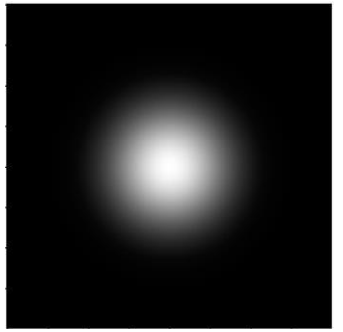
- 考虑噪声滤除，同时保留边界
- 特殊的2D-FIR滤波
- 滤波核的形状不再固定（无法看成传统的线性滤波）

3D数据滤波-双边滤波

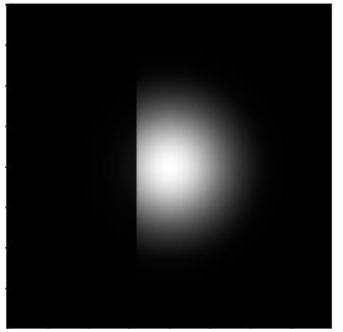
双边滤波的基本思想



- 考虑在图中十字中心点进行滤波
- 常规的滤波思想是用它邻域（图中圆圈）内部像素点的加权和作为滤波结果
- 传统方案使用“各向”一致的滤波核（加权系数）



- 考虑被滤波点在近处景物（石膏像）上，滤波结果应该使用石膏像上的像素，而不使用背景的像素
- 我们修改原先的滤波核，保留石膏像上的对应的加权系数，去除背景部分加权系数



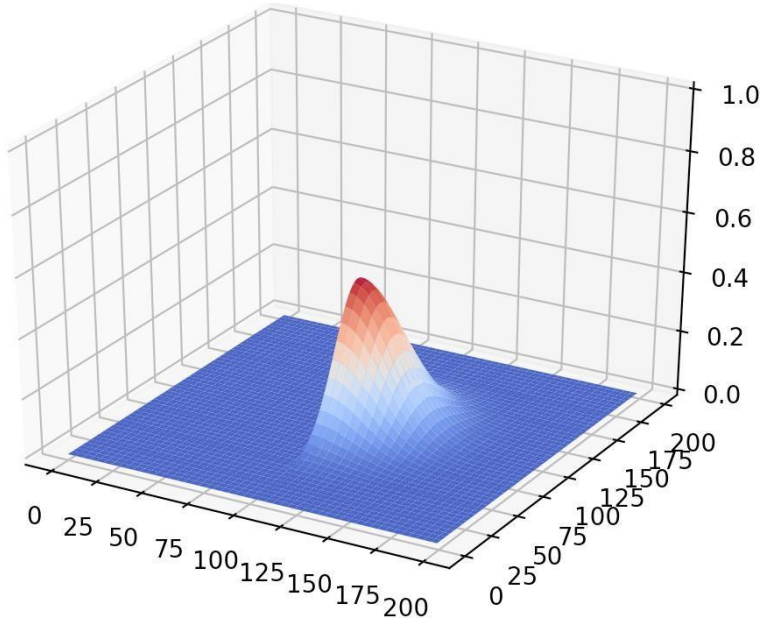
- 如何定量地设计这样的滤波核?

3D数据滤波-双边滤波

双边滤波的滤波核构成

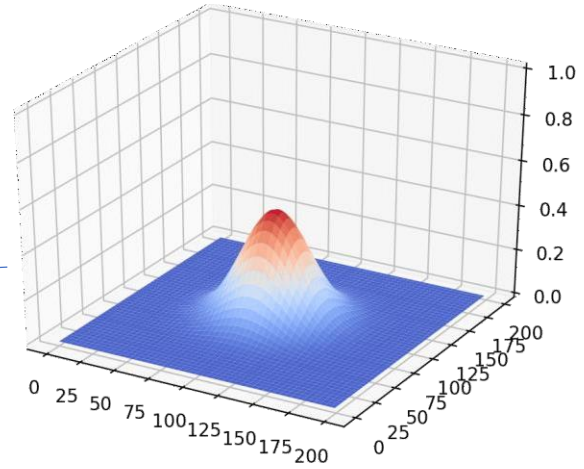
- 滤波核可以看成是两个2D“卷积核” 逐元素乘积
- 第一个“卷积核” 固定
- 第二个“卷积核” 取值随滤波滑动窗口位置改变

  值由滤波中心和周围各点的“差异”计算得到

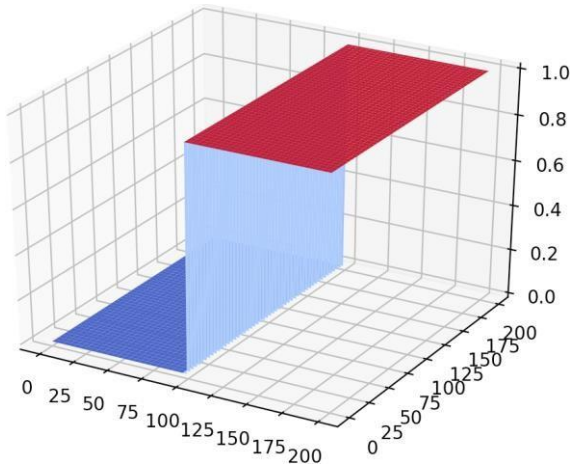


两个卷积核的乘积

逐点
乘积



第1个卷积核矩阵，固定取值的滤波和



第2个卷积核矩阵，随着滑动窗口位置改变

3D数据滤波-双边滤波

卷积核的具体构造例子

P点滤波输出　　距离加权项　　灰度差异加权项

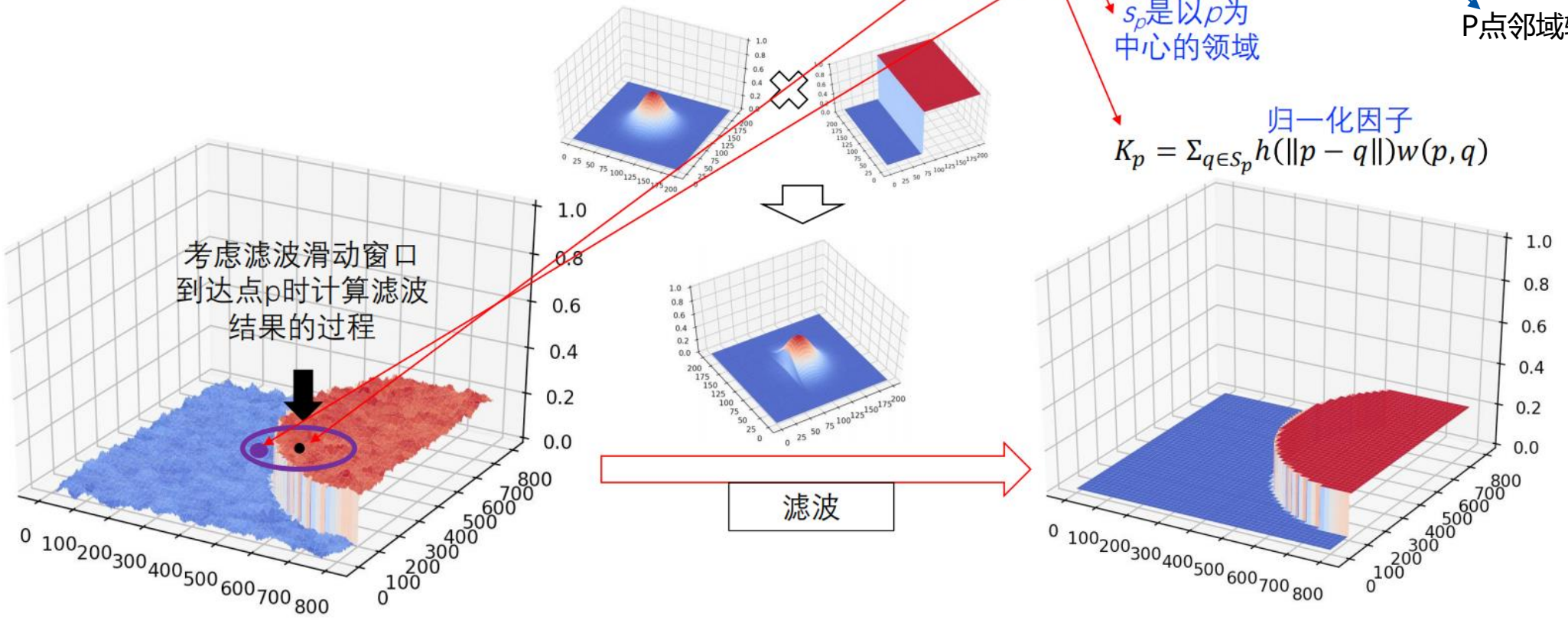
$$I_{out}(p) = \frac{1}{K_p}\Sigma_{q\in S_p} h(\|p-q\|)w(p,q)I_{in}(q)$$

S_p是以p为中心的领域

P点邻域输入

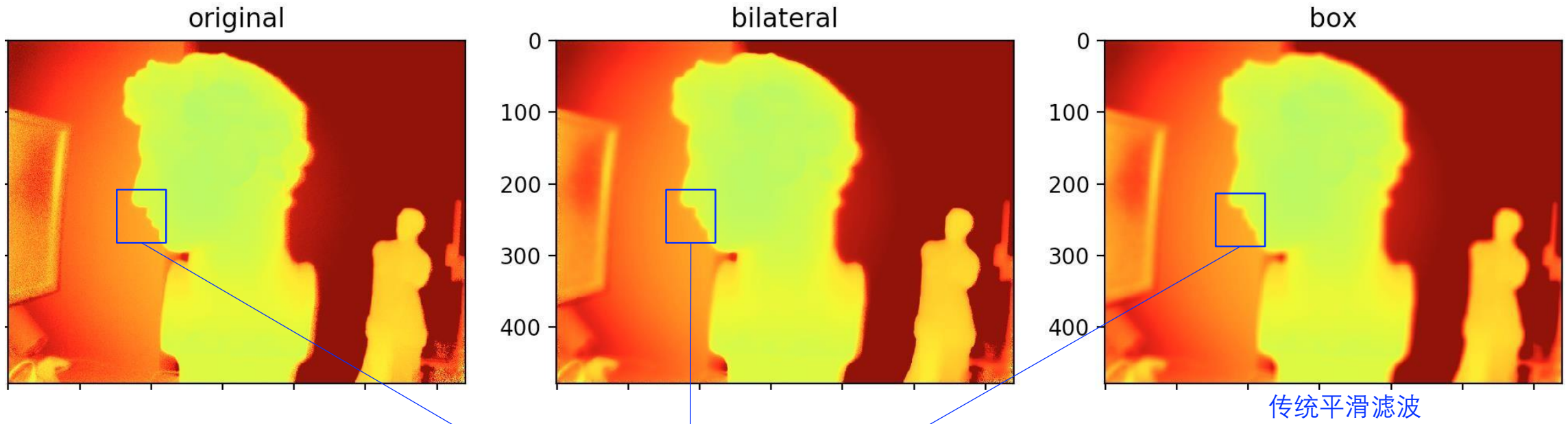
归一化因子

$$K_p = \Sigma_{q\in S_p} h(\|p-q\|)w(p,q)$$

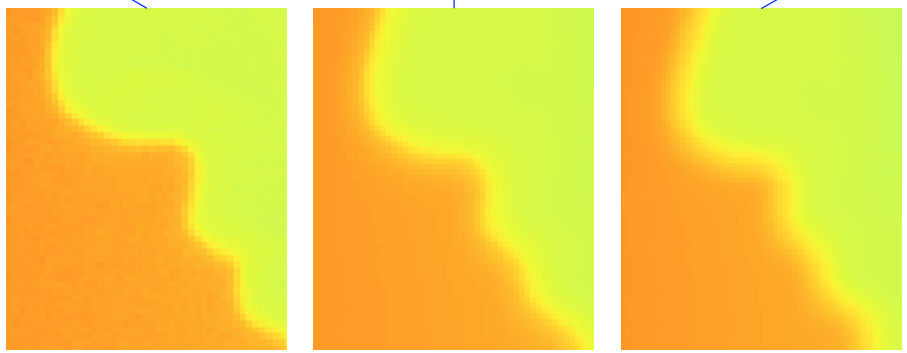


3D数据滤波-双边滤波

滤波效果比较



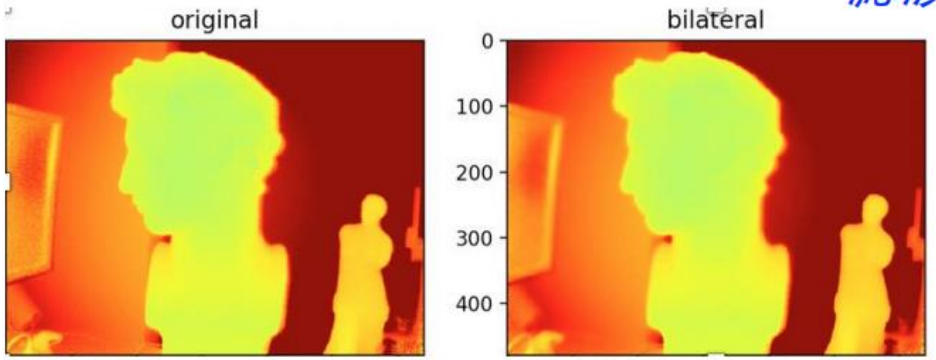
传统平滑滤波



相比右图更好地
保留边界信息

3D数据滤波-双边滤波

滤波过程软件实现（基于opencv API）



使用opencv内带API实现双边滤波

```
imgf=cv2.bilateralFilter(img_dep,5,1,1)
```

滤波半径（滑动窗口尺寸）

深度差异对应的权重衰减程度σ_r → $e^{-\frac{|I_{in}(p)-I_{in}(q)|^2}{2\sigma_r^2}}$

像素距离的权重衰减程度σ_d → $e^{-\frac{\|p-q\|^2}{2\sigma_d^2}}$

$$I_{out}(p)=\frac{1}{K_p}\Sigma_{q\in S_p}h(\|p-q\|)w(p,q)I_{in}(q)$$

滤波公式，I_{in}是输入深度图，I_{out}是滤波输出

滤波过程手工软件实现

```
150     ker1=np.ones((2*W,2*W))
151     for y in range(W,IMG_HGT-W):
152         for x in range(W,IMG_WID-W):
153             win_dep=img_dep[y-W:y+W,x-W:x+W]
154             ker2=np.exp(-(win_dep-img_dep[y,x])**2/0.02)
155
156             ker=ker1*ker2
157             ker/=np.sum(ker)
158             imgf[y,x]=np.sum(ker*win_dep)
```

手动编写的双边滤波可以灵活改变滤波器的构成

3D数据滤波-双边滤波

距离加权项　差异加权项

$$I_{out}(p) = \frac{1}{K_p}\Sigma_{q\in S_p} h(\|p-q\|)w(p,q)I_{in}(q)$$

构建$w(p,q)$的不同方案：

- 使用红外强度图 → $w(p,q) = e^{-\frac{|G(p)-G(q)|^2}{2\sigma_r^2}}$　G是强度图，$G(p)-G(q)$指两个位置强度差别
- 使用RGB图 → $w(p,q) = e^{-\frac{|r(p)-r(q)|+|g(p)-g(q)|+|b(p)-b(q)|}{2\sigma_r^2}}$　$r(*)$、$g(*)$、$b(*)$是三个颜色图图
- 使用0/1截断函数 → $w(p,q) = \begin{cases} 1 & |I_{in}(p)-I_{in}(q)| \le \theta \\ 0 & otherwise \end{cases}$　θ是给定“截0”门限
- 使用平面方向向量 → $w(p,q) = |<\mathbf{n}(p),\mathbf{n}(q)>|$　$\mathbf{n}(*)$是法向量方向（矢量）

3D数据滤波-时域平均滤波

时域平均滤波，使用T帧数据

输出深度图　　输入深度图

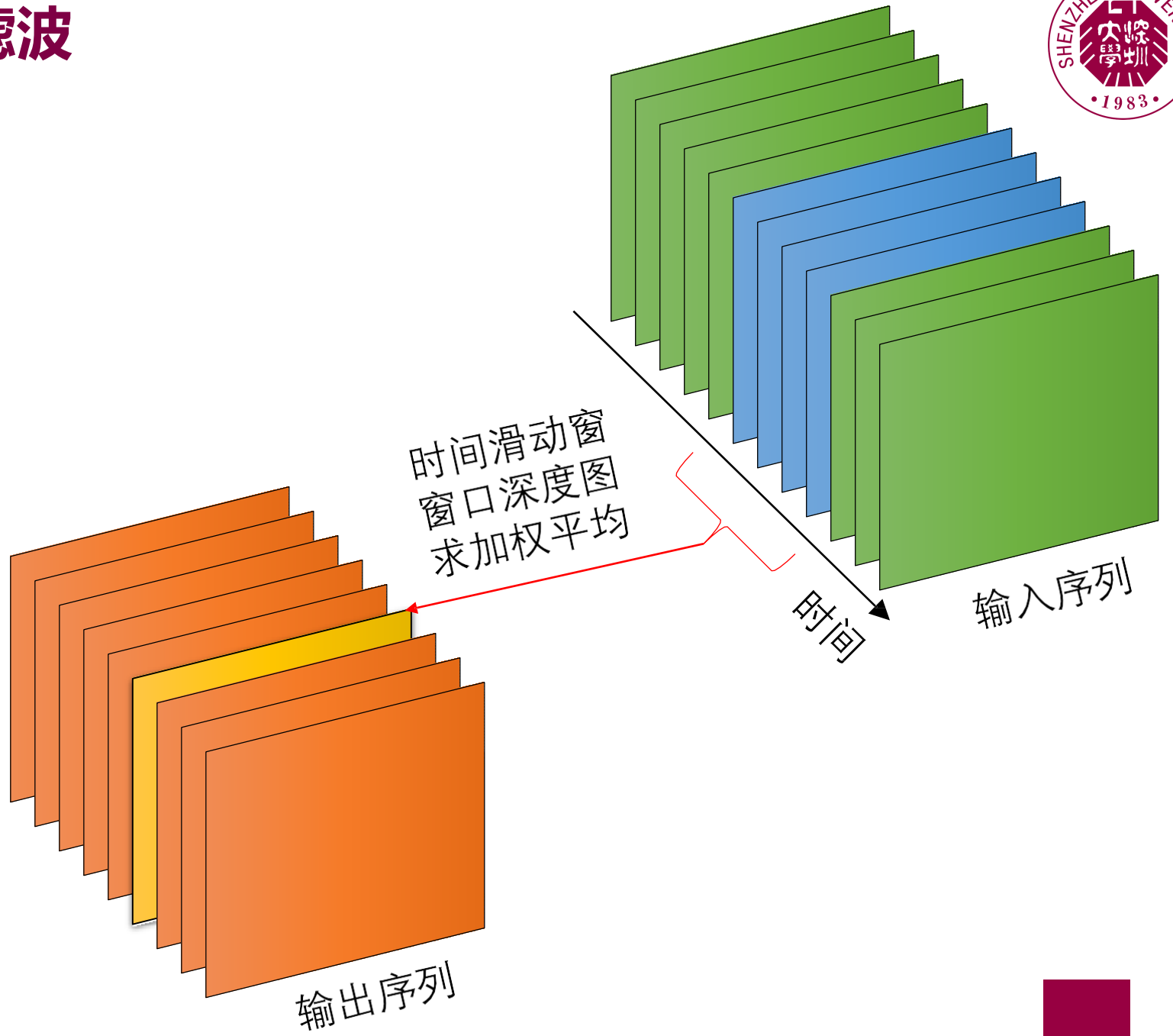
$$I_{out}(t) = \frac{1}{T}\Sigma_{\tau=0}^{T-1} I_{in}(t-\tau)$$

输出深度图　　输入深度图

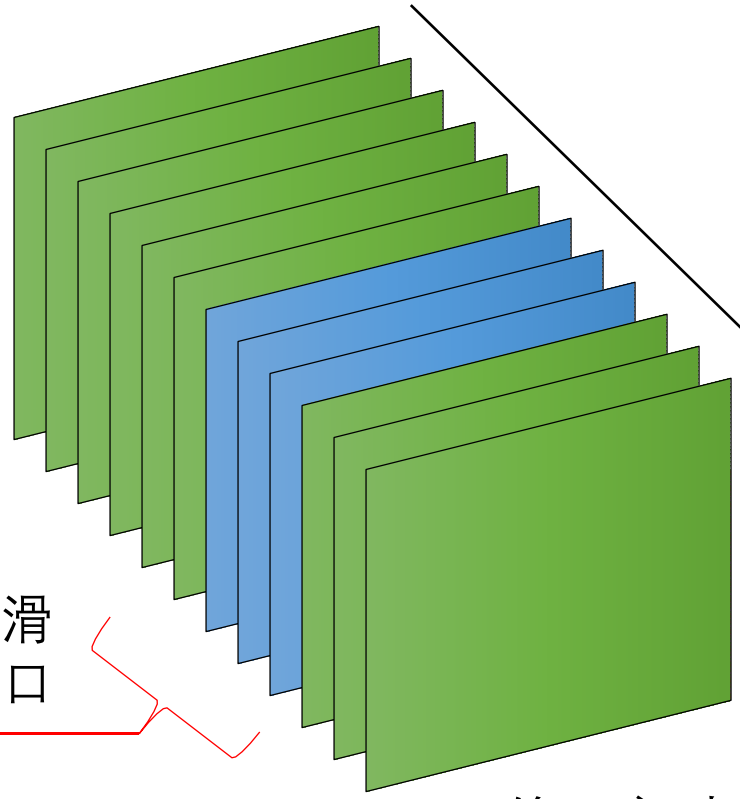
$$I_{out}(t) = \Sigma_{\tau=0}^{T-1} w(\tau) I_{in}(t-\tau)$$

加权系数

（通常要求$\Sigma_{\tau=0}^{T-1} w(\tau) = 1$）

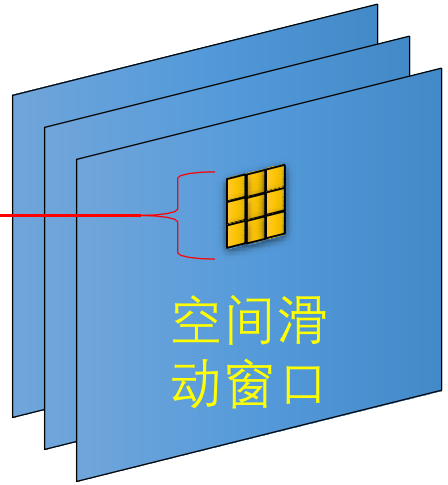


3D数据滤波-时空滤波



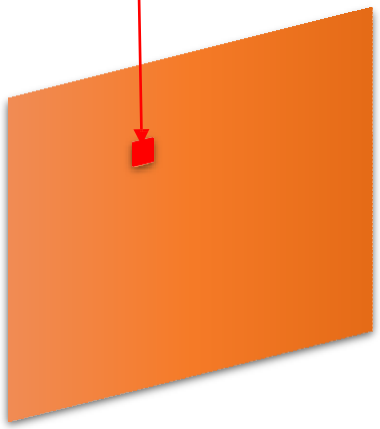
时间滑
动窗口

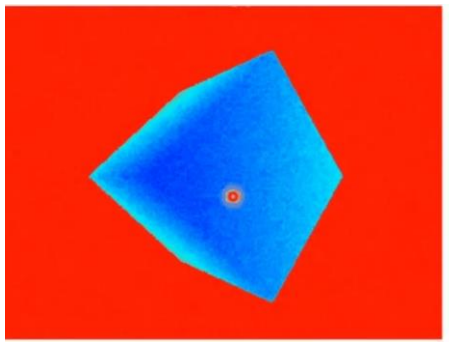
输入序列

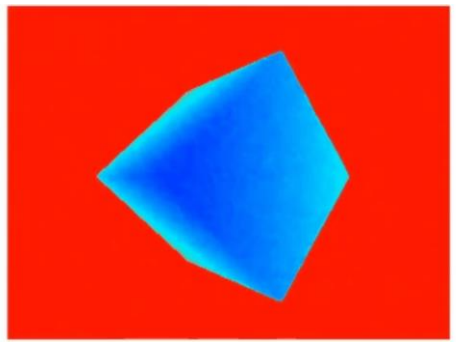


所有这些像素合
并得到滤波结果

- 计算加权平均
- 计算中值



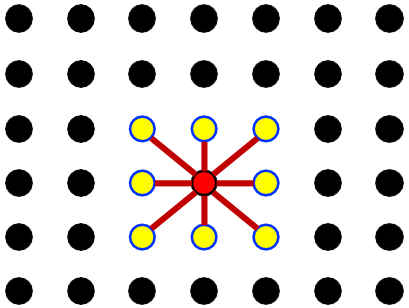




3D数据滤波-时空滤波

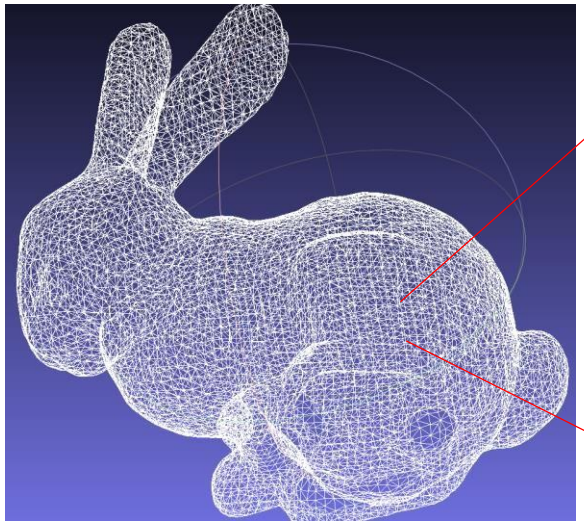
- 能够类似深度图计算点云的加权和吗？
  - 深度图滤波的“邻域”基于2D网格查找
  - 点云的“邻域”根据物理距离查找

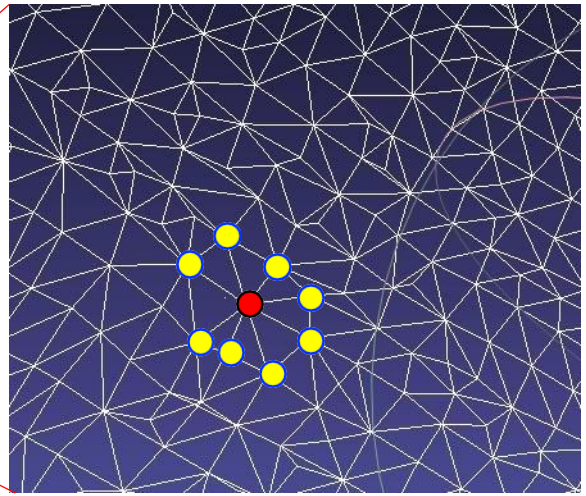
- 三角化的3D物体表面平滑滤波
  - 使用三角形连线直接找到“邻域”



深度图按2D像素位置找到“邻域”点，用于滤波

网格化





三角网格化后，直接通过网格连线找到“邻域”点

3D数据滤波-Laplacian点云平滑滤波

- 对网格中的每个顶点的坐标，用它邻近连接点的坐标的平均值代替

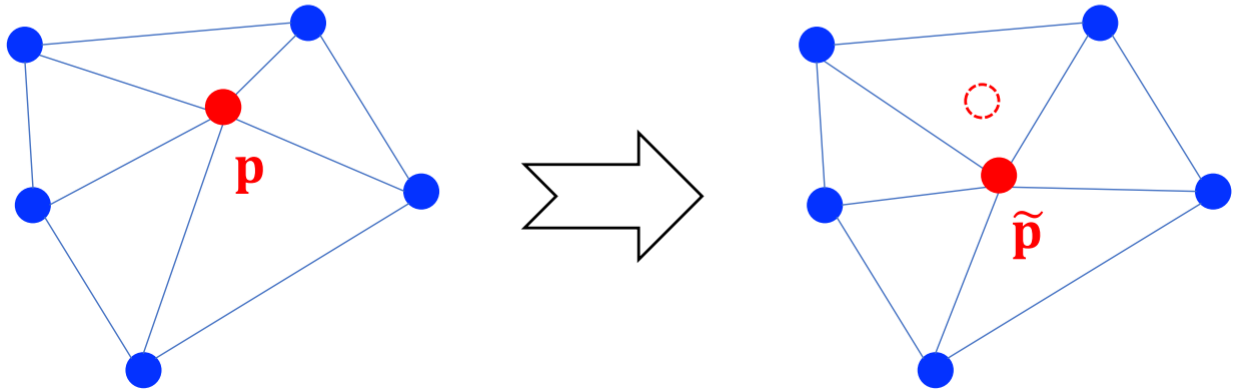
$$\tilde{\mathbf{p}} = \frac{1}{|S_{\mathbf{p}}|} \sum_{\mathbf{q}\neq\mathbf{p},\mathbf{q}\in S_{\mathbf{p}}} \mathbf{q}$$

$\tilde{\mathbf{p}}$：滤波修正的点的位置

$\mathbf{p}$：滤波前点的位置

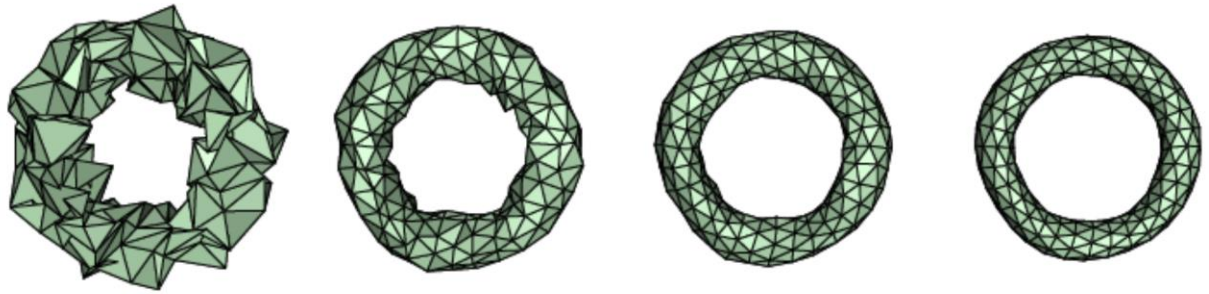
$\mathbf{q}$：$\mathbf{p}$的领域内的点，不包括$\mathbf{p}$

- $S_{\mathbf{p}}$：点$\mathbf{p}$的领域
- $|S_{\mathbf{p}}|$：点$\mathbf{p}$的领域点的数量

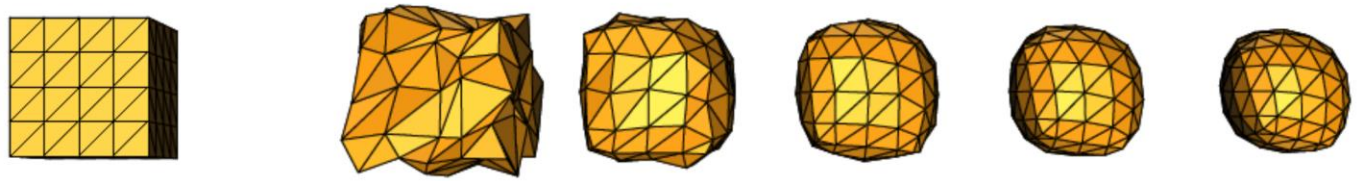


3D数据滤波- Laplacian点云平滑滤波

- 可以在网格上重复运行几次，每次比之前更加光滑



- 缺点——边沿损失，尺寸总体变小



3D数据滤波- Laplacian点云平滑滤波

滤波计算的几种变化形式

- 允许“固定点”集合S_{fixed}
- 固定点集合内的点坐标不被修正

$$\tilde{\mathbf{p}} = \begin{cases} \frac{1}{|S_{\mathbf{p}}|} \sum_{\mathbf{q} \neq \mathbf{p}, \mathbf{q} \in S_{\mathbf{p}}} \mathbf{q} & \mathbf{p} \notin S_{fixed} \\ \mathbf{p} & \mathbf{p} \in S_{fixed} \end{cases}$$

- 对原始点的位置部分修正，而不是替换

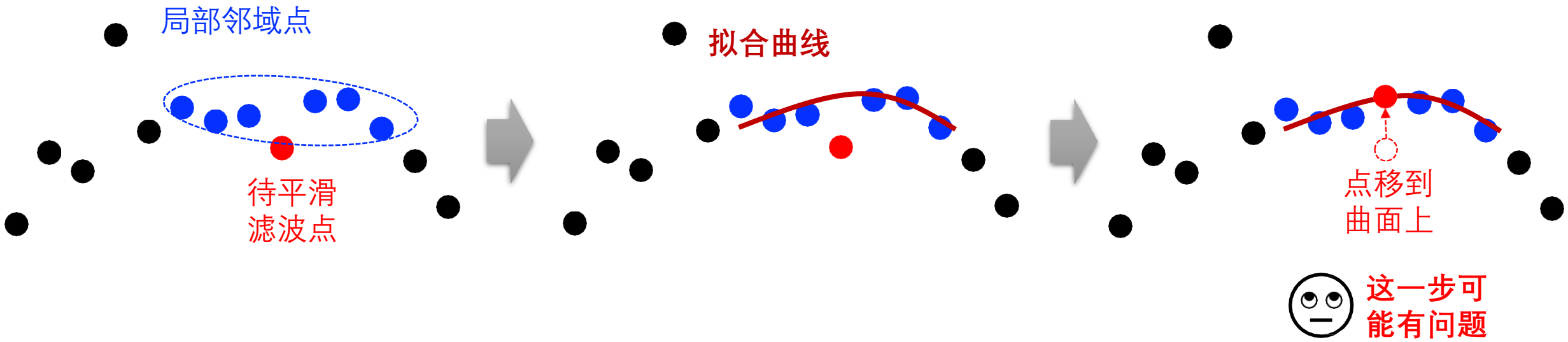
$$\tilde{\mathbf{p}} = \begin{cases} \alpha\mathbf{p} + \frac{1-\alpha}{|S_{\mathbf{p}}|} \sum_{\mathbf{q} \neq \mathbf{p}, \mathbf{q} \in S_{\mathbf{p}}} \mathbf{q} & \mathbf{p} \notin S_{fixed} \\ \mathbf{p} & \mathbf{p} \in S_{fixed} \end{cases}$$

数据点的更新的两种方式

- 同步更新：计算结果放置在新的数据区，滤波计算用的数据固定，由于需要两个存储区域，占用空间大。
- 连续更新：计算过程中，直接更改被滤波的点。计算过程中会用到之前刚修改过的点，计算结果和顶点的访问次序有关

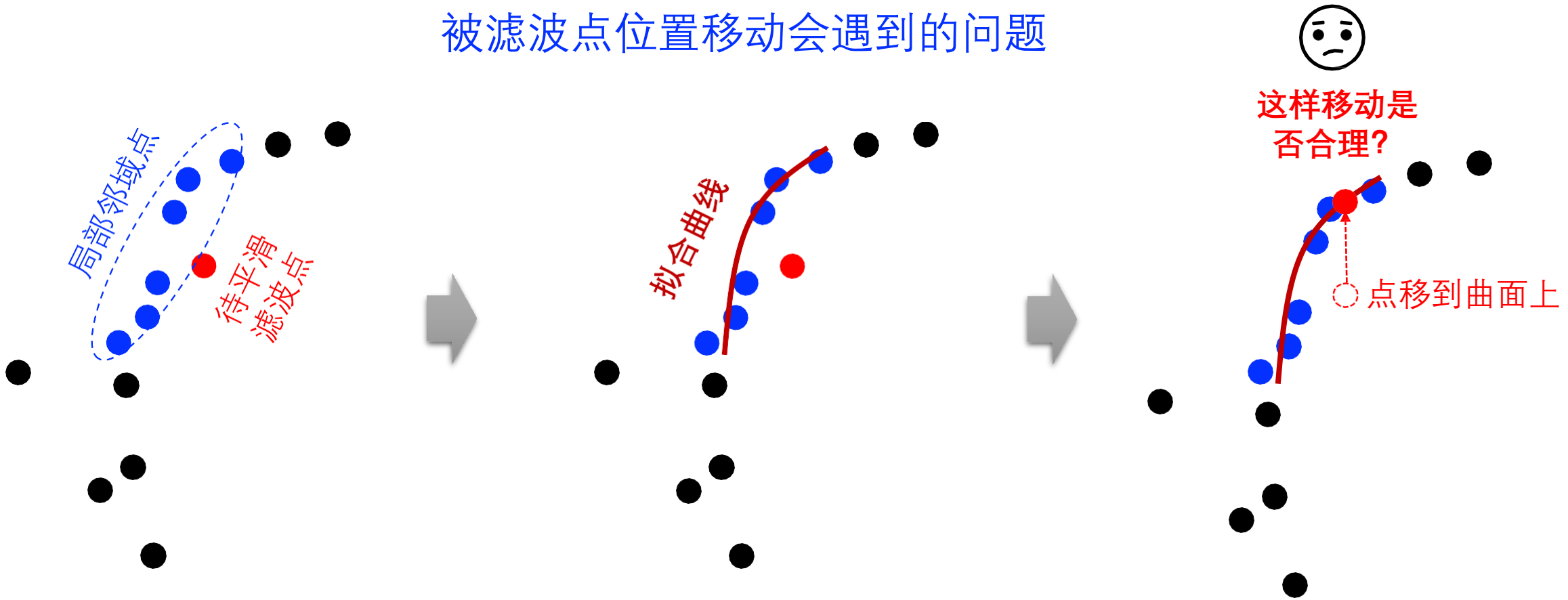
3D数据滤波- 移动最小二乘滤波

- 基本思想——对待平滑的点，利用它的邻域点云拟合光滑曲面， 并用曲面上的点作为平滑结果
- 一维的例子：



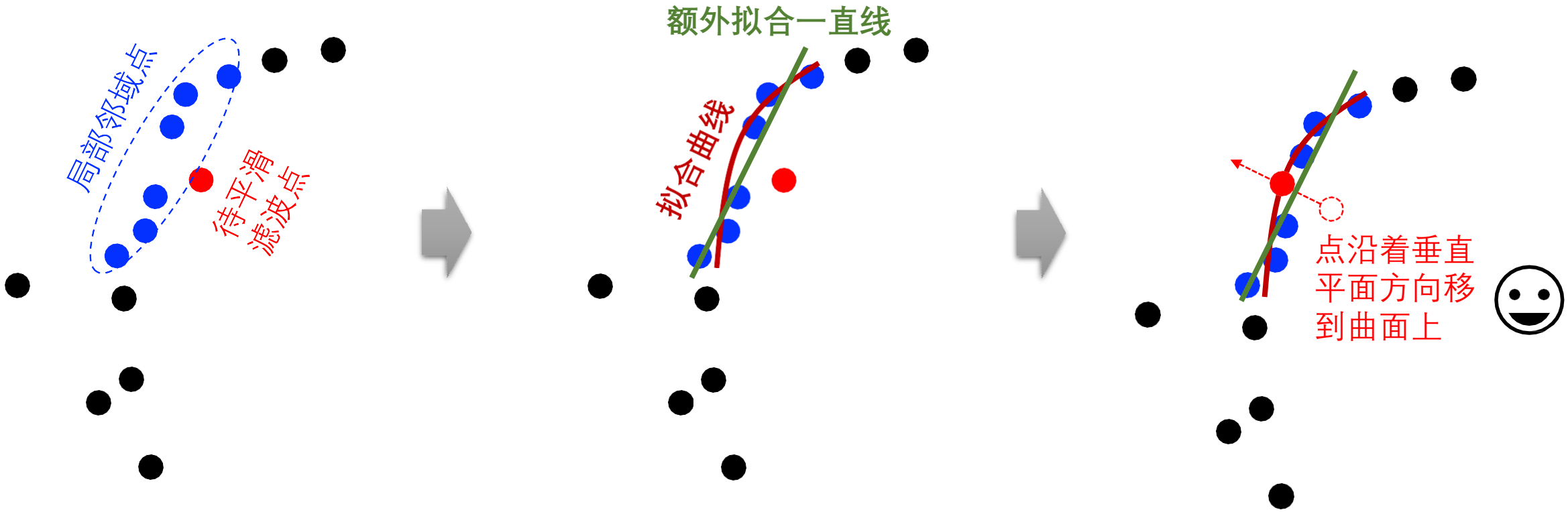
3D数据滤波- 移动最小二乘滤波

被滤波点位置移动会遇到的问题



3D数据滤波- 移动最小二乘滤波

通过额外再拟合一个平面来确定被滤波点移动方向



对3D空间，这个过程是先拟合平面，再拟合曲面，然后将待平滑的点沿平面法向量投影到曲面上去

3D数据滤波- 移动最小二乘滤波

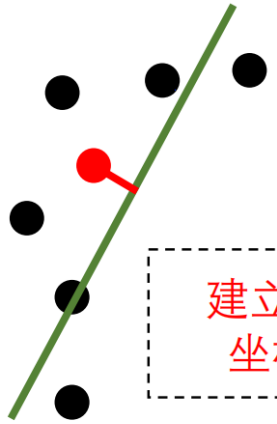
对待滤波点找到邻域点

对邻域点拟合平面

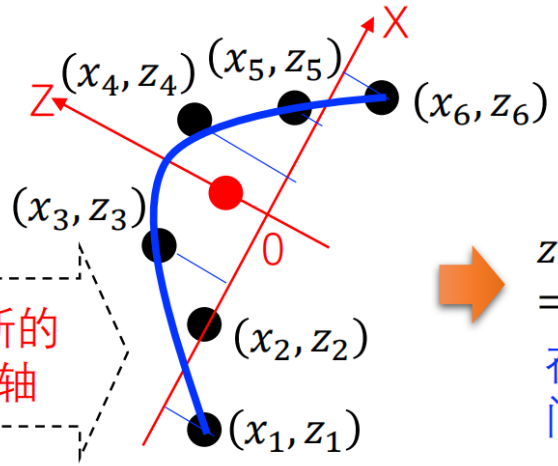
将邻域点投影到平面，并计算他们到平面距离

曲面方程拟合（通常使用多项式）

将待滤波点沿垂直平面方向投影到曲面

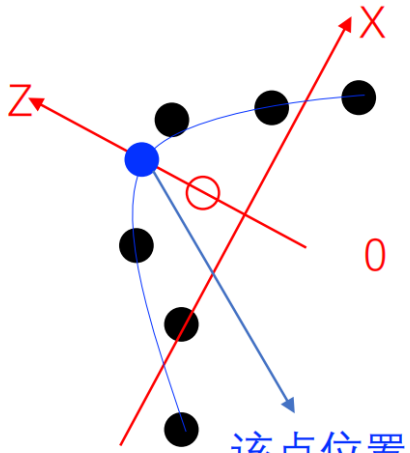


建立新的坐标轴



$z = f(x)$
$= ax^2 + bx + c$

在新建坐标空间拟合多项式



该点位置：
$z_0 = f(0)$

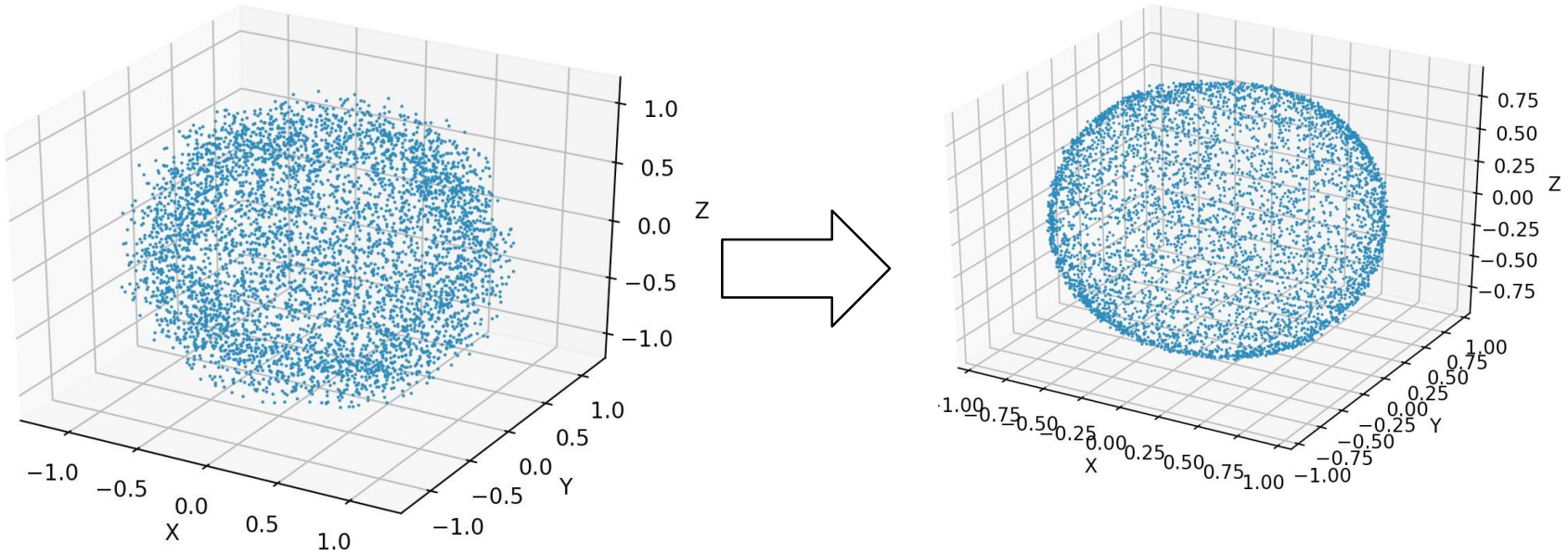
考虑新的坐标系，使待滤波点在平面上的投影为原点，平面法向量和Z轴对齐

在新的坐标系下进行曲面方程拟合（通常使用多项式）

计算曲面在原点的Z值，找到待滤波点的移动位置

3D数据滤波- 移动最小二乘滤波

滤波运行效果



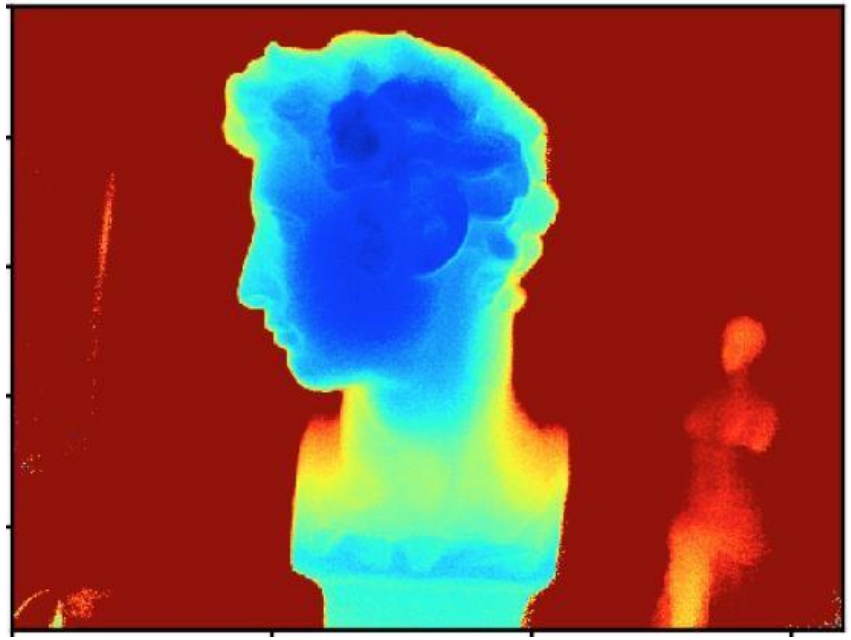
目录

CONTENTS

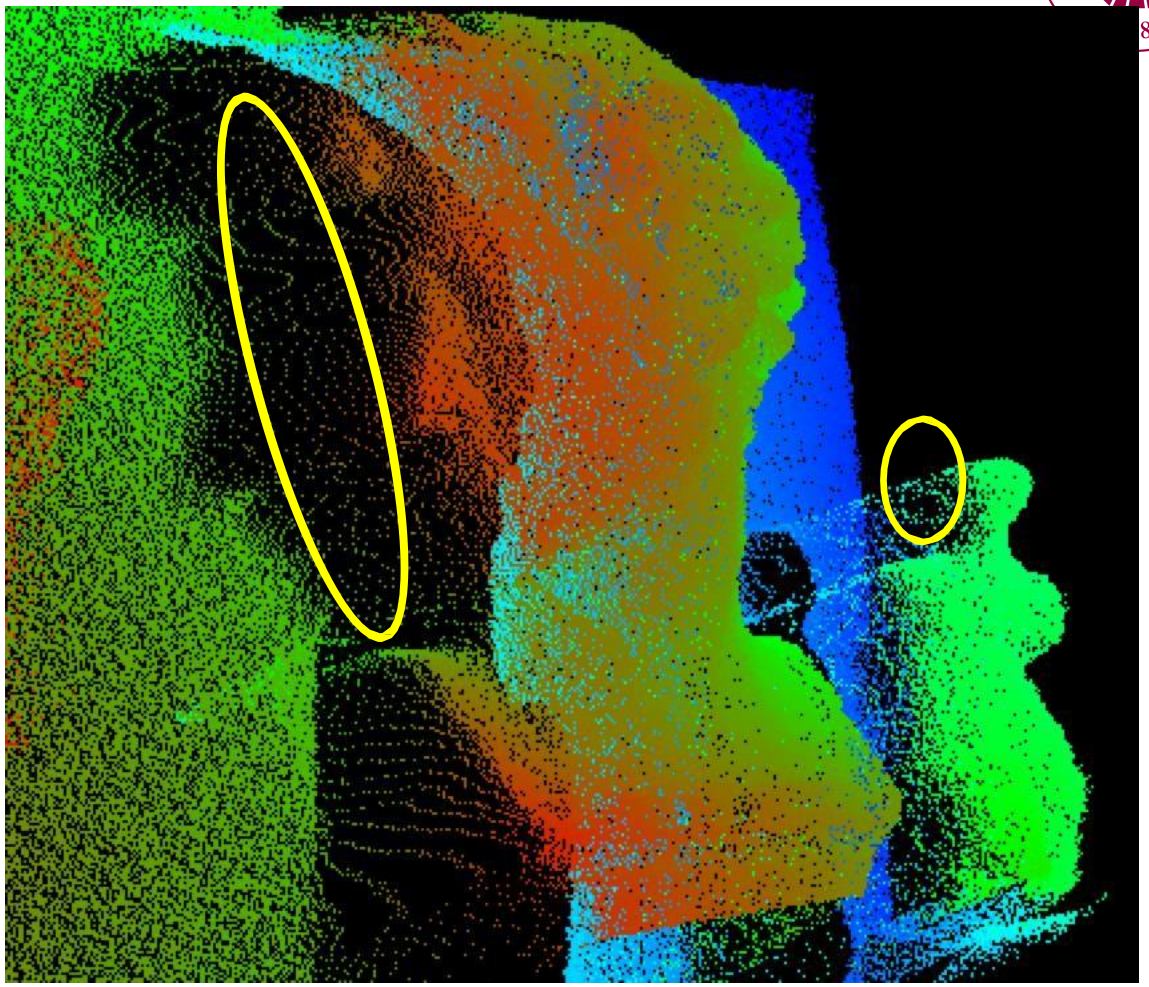
02 3D数据过滤

3D数据过滤

- 前后景边界看到“飞散点”
- 飞散点的使得点云中出现虚假的 “墙面”或者“射线”



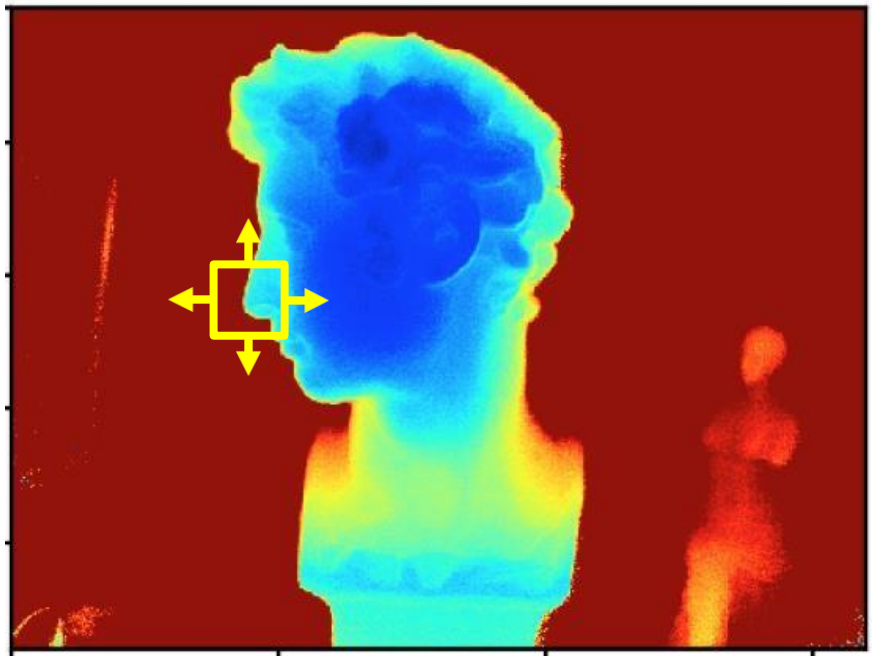
转成点云后旋转查看



边界像素点可能同时包括了前景和后景的叠加，出现距离“混合”

3D数据过滤

- 基于深度图邻域像素值方差的过滤器
- 邻域像素值方差如何计算?



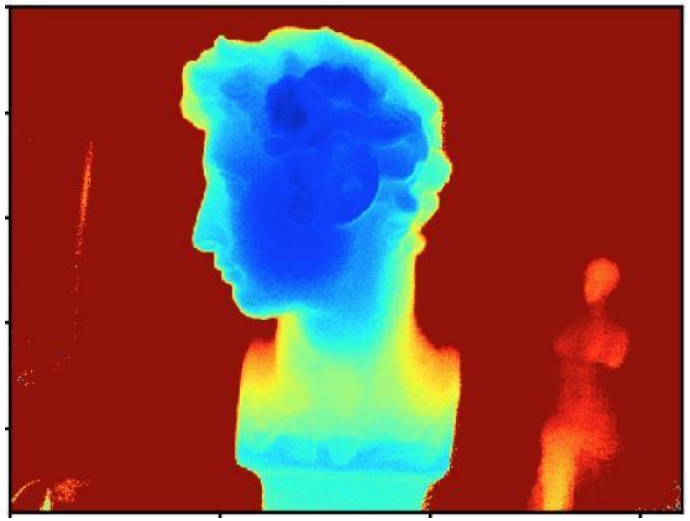
计算滑动窗口
内点的方差

3D数据过滤

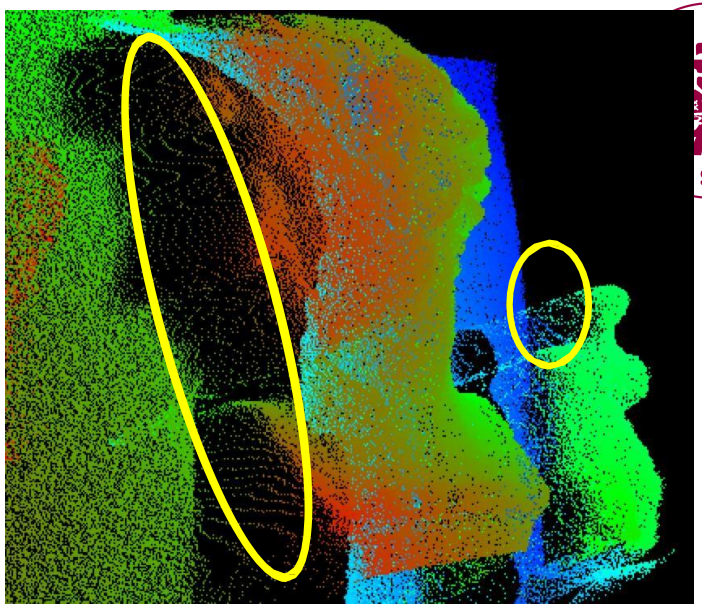
——基于深度图的噪声点过滤

基于深度图邻域像素值方差的过滤器

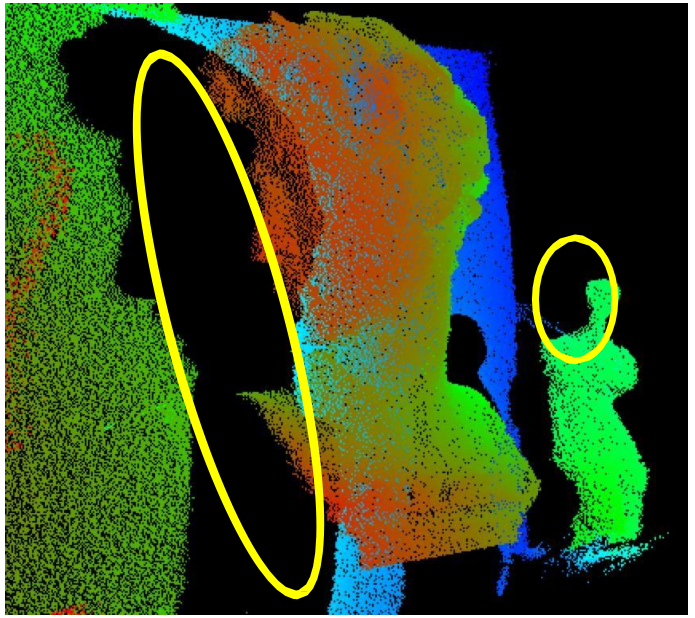
```
 8    img_var0=cv2.blur(img,(win,win))**2
 9    img_var1=cv2.blur(img**2,(win,win))
10    img_var=img_var1-img_var0
11    mask=img_var.ravel()<th
```



局部方差计算



未处理前的点云



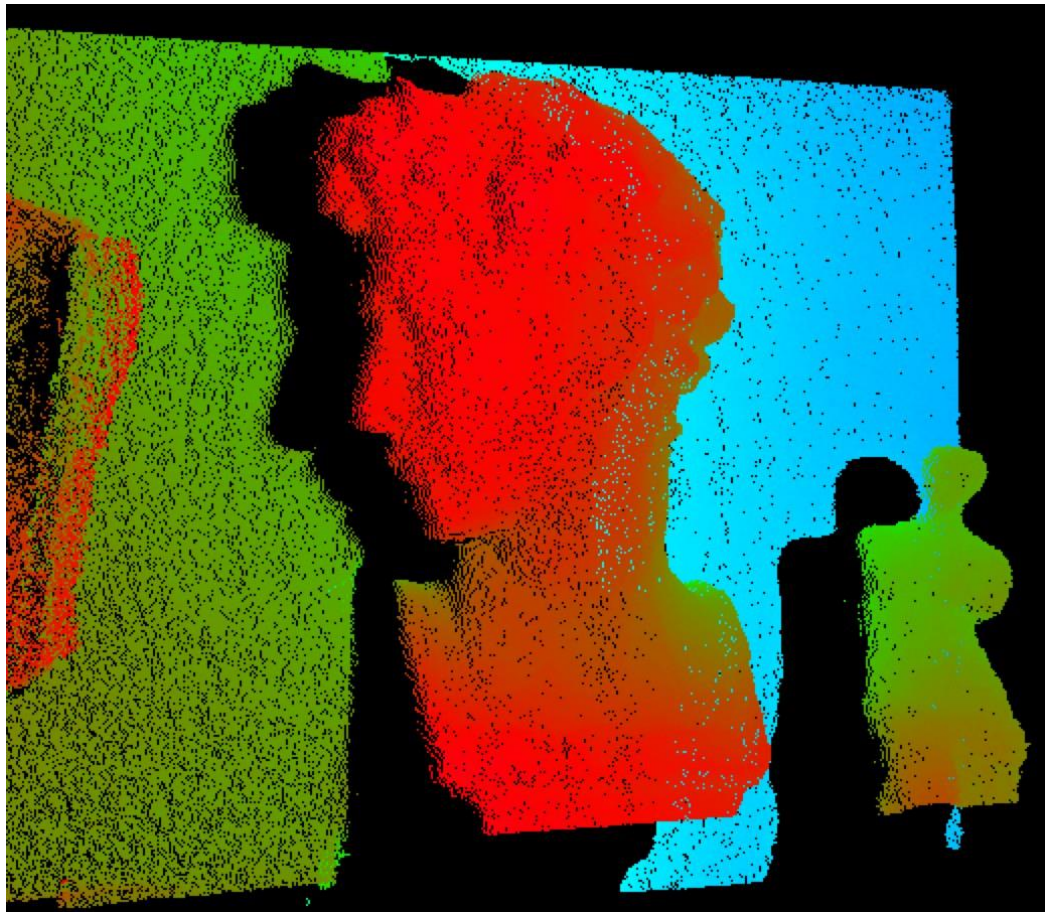
切除深度图中局部方差大于门限的像素后生成的点云

3D数据过滤

—基于点云的噪声点过滤

“半径过滤”——基于邻域点云距离的过滤器

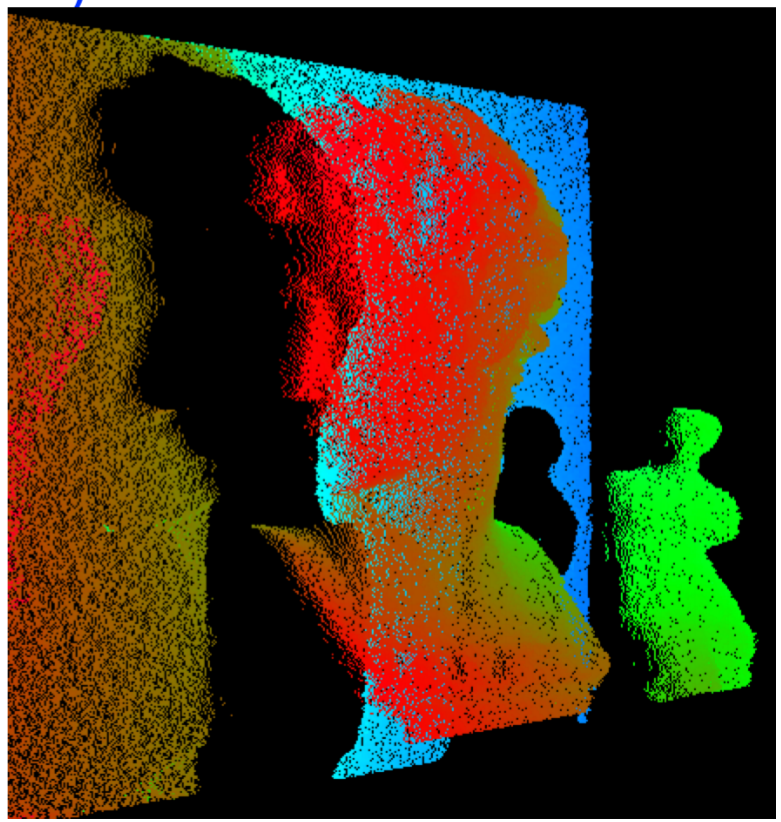
- 对每个点搜寻其K近邻
- 当K各近邻中离他最远的超过给定门限时，表明 当前点是噪声（“离群点”）



3D数据过滤

- 对每个点$\mathbf{p}_i\ (i = 1,2,\dots N)$，计算其K近邻，并得到这K近邻的平均距离$d_i\ (i = 1,2,\dots N)$
- 对所有d_i统计其均值$\bar{d} = \frac{1}{N}\Sigma_n d_i$和方差$\sigma^2 = \frac{1}{N}\Sigma_n\left(d_i - \bar{d}\right)^2$
- 建立门限$\theta = \bar{d} + \alpha\sigma$
- 对每个点$\mathbf{p}_i$，若其K近邻平均距离$d_i > \theta$则删除该点

```
28  import pcl
29
30  # 构建PCL点云对象
31  cloud = pcl.PointCloud()
32  cloud.from_array(pc.astype(np.float32))
33
34  # 利用PCL构建过滤器引擎
35  fil = cloud.make_statistical_outlier_filter()
36  fil.set_mean_k(k)                  # 用于统计距离的近邻点数量(k)
37  fil.set_std_dev_mul_thresh(th)   # 用于识别噪声点的门限(th)
38
39  # 计算过滤结果
40  cloud_r=fil.filter()
```



目录

CONTENTS

PCL滤波参考：

https://pcl.readthedocs.io/projects/tutorials/en/latest/#filtering

Filtering

- Filtering a PointCloud using a PassThrough filter

  Title: **Filtering a PointCloud using a PassThrough filter**
  Author: *Radu B. Rusu*
  Compatibility: > PCL 1.0
  In this tutorial, we will learn how to remove points whose values fall inside/outside

- Downsampling a PointCloud using a VoxelGrid filter

  Title: **Downsampling a PointCloud using a VoxelGrid filter**
  Author: *Radu B. Rusu*
  Compatibility: > PCL 1.0
  In this tutorial, we will learn how to downsample (i.e., reduce the number of points)

- Removing outliers using a StatisticalOutlierRemoval filter

  Title: **Removing sparse outliers using StatisticalOutlierRemoval**
  Author: *Radu B. Rusu*
  Compatibility: > PCL 1.0
  In this tutorial, we will learn how to remove sparse outliers from noisy data, using S

- Projecting points using a parametric model

  Title: **Projecting points using a parametric model**
  Author: *Radu B. Rusu*
  Compatibility: > PCL 1.0
  In this tutorial, we will learn how to project points to a parametric model (i.e., plane

- Extracting indices from a PointCloud

  Title: **Extracting indices from a PointCloud**
  Author: *Radu B. Rusu*
  Compatibility: > PCL 1.0
  In this tutorial, we will learn how to extract a set of indices given by a segmentatio

- Removing outliers using a Conditional or RadiusOutlier removal

  Title: **Removing outliers using a Conditional or RadiusOutlier removal**
  Author: *Gabe O'Leary*
  Compatibility: > PCL 1.0
  In this tutorial, we will learn how to remove outliers from noisy data, using Conditi

作业：点云数据降采样与过滤

- 要求：
  - **点云截取：**获取X值范围在0-20m，Y值范围在(-6)-(-16)m，Z值范围在1-4m内的点云数据，保存PCD文件，命名为“passthrough.pcd”，可视化并截图保存为“passthrough.png”
  - **点云降采样：**将截取后的点云数据降采样到5cm精度，保存为PCD文件，命名为“downsample.pcd” ，可视化并截图保存为“downsample.png”
  - **点云过滤：**将对降采样后的点云数据进行噪声去除，保存为PCD文件，命名为“filtering.pcd”，可视化并截图保存为“filtering.png”
- 作业提交格式
  - 压缩文件命名：姓名+学号+点云数据预处理.zip
  - 文件夹中需要包含：1、原始代码文件夹，2、点云截取文件夹，3、点云降采样文件夹，4、点云过滤文件夹

谢谢