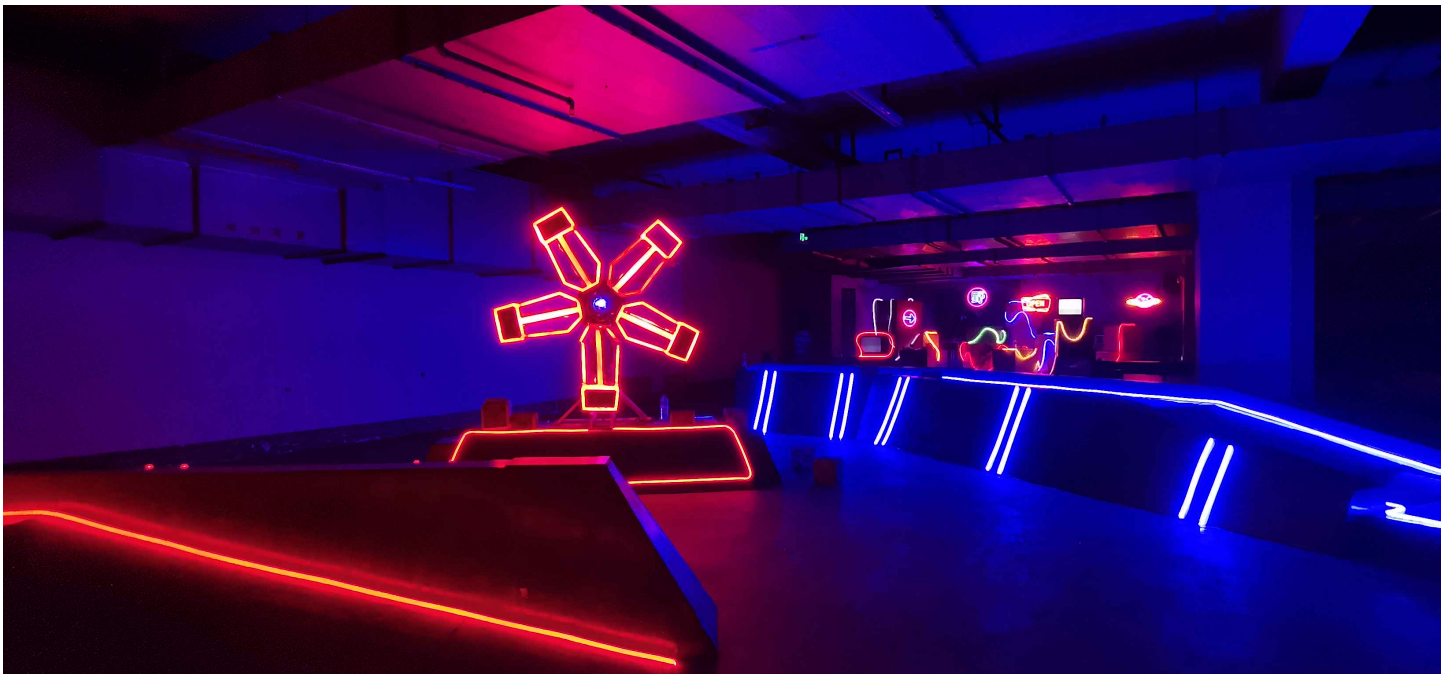


华南虎工程视觉开源文档

相机相关

选型

工业相机：迈德威视MV-SUA133GC

镜头：CS镜头 1/2.5” 2.8mm

相机标定

由于相机位置安装靠前，工作距离大概为500mm。为了满足所有五级兑换的视觉识别需求，需要用到2.8mm甚至更短焦距的镜头以获得更大的视野。但问题随之而来，镜头的鱼眼效应特别严重（畸变），但是OpenCV提供的基于棋盘格检测的相机标定方法并不是十分有效，而一个不准确的畸变矩阵不但会影响SolvePNP解算结果，甚至对轮廓识别造成影响，因此一个精确的畸变矩阵和内参矩阵是十分重要的。

首先用焦距除以相机的像原尺寸估算一个近似的f/dx和f/dy值，然后u0，v0为1/2画面宽度、高度。这个只做为一个粗调，后面还会修改。

既然畸变矩阵只有5个值，那不如直接在画面框中创建5个滑动条，手调畸变矩阵。滑动条的回调函数修改畸变矩阵后，初始化两个矩阵，map1，map2，然后在while循环中使用initUndistortRectifyMap(相机矩阵，畸变矩阵，Mat()，新相机矩阵，画面尺寸，CV_16SC2，map1，map2)。其中的Mat()直接这么输入就行，由于画面的矫正会改变图像的尺寸，新相机矩阵理应用getOptimalNewCameraMatrix()函数计算得出，但实测使用原相机矩阵也可以达到较高精度的姿态解算结果（工作距离500mm下平均误差2mm）。initUndistortRectifyMap函数会对map1，map2

赋值，运用remap（原图像，新图像，map1，map2，INTER_LINEAR）进行调整。设定好曝光、增益等相机参数后，在相机前放置一块标定板就可以目视调整了。由于投影原理，画面会存在近大远小的情况，这个不用管它，只需要调整到标定板任意地方的直线在图像上也都是直线就可以了。

畸变矩阵的调整就完成了，然后是相机矩阵的精调。

运行写好的自描或者兑换槽解算代码，打印出SolvePNP函数的tvec值，手动测量相机cmos平面到识别物的距离，调整f/dx，f/dy的值，直到tvec中的z准确即可。

至此，相机标定的全部工作结束。

算法相关

兑换槽姿态解算（传统视觉，识别正面四角点）

1. 预处理

画面畸变矫正，通道分离，可以提取出红蓝两个通道进行叠加再进行二值化，这样得到的二值图是同时包含红色和蓝色的，也就是红蓝兑换槽均可识别，由于工程兑换视觉的特殊性，并不会工作时红蓝均出现的情况，同时避免类似自描红蓝方切换可能会出bug的隐患，也使调试更便利，便于检测算法的鲁棒性。

2. 角点轮廓筛除

在第一步初筛中我用了以下几个判断条件：

a. 轮廓宽高比和高宽比小于4.5 (height / width < 4.5 && width / height < 4.5)

b. 轮廓面积大于400小于12000pixel (area > 400 && area < 12000)

c. 多边形拟合边数大于5小于9 (edge > 5 && edge < 9)

将符合以上条件的轮廓放入vector中进行二次筛选，筛选条件为：

a. vector元素数量等于4

b. 最大轮廓面积小于10倍最小轮廓面积 (max.size() < 10 * min.size())

3. 角点顶点定位

a. 对vector中轮廓做三角形拟合和最小外接圆。用四个圆的圆心确定兑换槽前表面中点。

注意：由于近大远小的图像特性，三角形拟合的最大角不一定是轮廓顶角。同样，离中点最远的三角形顶点也不一定是轮廓顶点。通过观察，错误的最大角角度一般不会超过100度，顶角只有在大于120度以上才会出现到中点的距离小于其他两角的情况，根据以上结果，做出以下判断条件。

b. 计算四个三角形最大角的角度，若此角度大于130度，则认为此角顶点为角点顶点。

c. 如果三角形中没有大于130度的角，计算各顶点到中点的距离，取距离最远者为轮廓顶点。

至此，四个顶点已经都识别到了。

4. 角点排序

设兑换槽正对时，右上小角点为0号，逆时针顺序排序，左上、左下、右下为1、2、3号角点。通过观察发现，vector中的角点顺序永远为逆时针顺序（可能是opencv中findContour函数的特性导致的），也就是说vector中角点有0123、1230、2301、3012四种顺序情况。我们可以通过找到0号小角点来确定vector中的顺序。还是由于近大远小的图像特性，0号角点对应的轮廓的面积不一定是最小的。我通过找0号角点边上的两个小方块来确定0号角点的位置。通过以下条件筛选：

a. 面积小于200大于50 （ area < 200 && area > 50)

b. 矩形拟合的矩形面积小于1.2倍轮廓面积 (rectangle.shape.area() < 1.2 * contour.area())

c. 宽高比和高宽比均小于2 (height / width < 2 && width / height < 2)

现在找到了两个小方块，也有可能只能识别到一个，就找到小方块的中点，取顶点离这个中点最近的点为0号角点。但是也有可能一个小方块都识别不到，那此时0号角点肯定是远离相机的，也就是说此时的0号角点对应的轮廓就是面积最小的。做一个面积判断就能很容易找到0号角点了。找到0号以后重新排序一下角点就可以了。

5. 识别点滤波

由于pnp解算出的旋转向量和最后与电控对接的四元数各个数据间都有一定的依赖关系，设计这样的滤波器会较为复杂，所以干脆就对识别出来的Point2f点进行滤波，也就是在SolvePnp前滤波。原理也很简单，令第一帧的识别数据为(0,0)，当前帧数据为上一帧数据乘0.9再加上本帧数据。

X(0) = (0,0)

X(t) = X(t-1) * 0.9 +X(t) * 0.1

6. pnp

选用SOLVEPNP_IPPE_SQUARE方法，将角点按正方形解算的顺序要求排序好进行运算。实测四个解算点的SOLVEPNP_IPPE_SQUARE的精度、速度远优于12个解算点的常用的IPPE。甚至识别有轻微抖动、误差，SOLVEPNP_IPPE_SQUARE仍能正确解算（误差小）。

7. 后处理（识别正确性判断）

场上兑换站环境并未出现误识别情况，但是在家测试由于兑换站漏光，有几率会误识别，所以加上了一些判断条件。识别出的四边形由于滤波拟合不完全或误识别等情况，会造成四边形出现凹四边形，或四边形面积某瞬间小于正确的矩形框，所以需要判断四边形的形状和面积。此外，由于兑换槽是仰角向上的，解算出的四元数x应为负值。而且由于滤波原理，本帧数据若存在，不应该会与上一帧数据一样，而且角点轮廓边缘光强较弱，会导致识别点有相邻像素间的跳变，也会导致数据在很小的某范围内抖动（0.1度，0.5mm)，所以相邻帧数据不同应是正确情况。综合以上，对数据给出一定的裕度，会有如下判断条件。

a. 识别出的四边形任意内角不小于40度，不大于130度 （ angle < 130 && angle > 40)

b. 识别框面积大于30000pixel （ rectangle.shape.area > 30000)

c. 解算出的四元数x值不大于0.05 （ quaternion.x < 0.05)

d. 本帧数据解算值不与上一帧相同 （ quaternionNow != quaternionLater)

以上算法由于大量使用华南虎视觉库内部函数，单独看代码可读性较差，故代码不做开源，场上的视觉相机实拍低曝光视频将放在网盘分享。

金矿石6D位姿估计（神经网络，单张RGB图像姿态估计）

Linemod格式数据集制作

6D-of位姿估计的数据集有很多种，类似于Linemod、Linemod-Occlusion、Ycbvideo等等，由于某些格式的官方数据集过大，如Ycbvideo官方数据集800多G，以及数据集制作过于复杂，需要rgb和深度数据融合等，最终选择了最经典的Linemod传统格式数据集，Linemod全称为Linemod_preprocessed数据集，以下简写为lm数据集。lm数据集包含以下数据：RGB图像、深度图像、掩模图、标注文件、相机参数、物体的点云模型、点云模型原点与尺寸数据、实例语义分割掩模图（这个一般用于模型评估，非必要）。

实拍数据点云重建自动语义分割方法

一、下载ObjectDatasetTools源码并配置环境

下载源码

```
🥖 git clone https://gitee.com/yu-chenghuan/Linemod-Real.git
```

创建并打开一个新conda环境python=3.6

```
🎉 conda create -n ODT python=3.6
   conda activate ODT
```

安装依赖

```
🎁 sudo apt install build-essential cmake git pkg-config libssl-dev libgl1-mesa-glx
```

安装meshlab点云处理软件

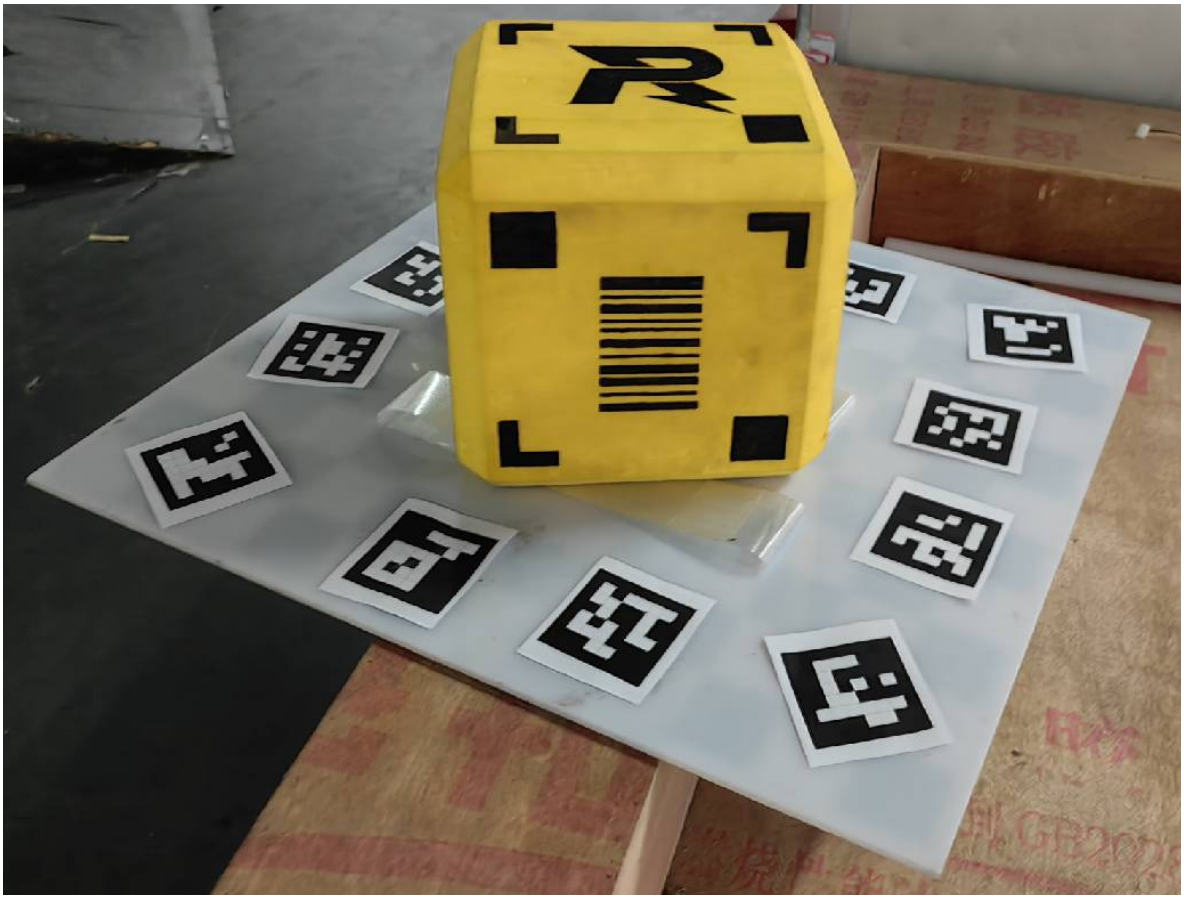
```
🏆 sudo apt-get install meshlab
```

安装python包（注意：只支持opencv-python4.6版本！！）

🌟 pip install numpy pypng scipy scikit-learn open3d scikit-image tqdm pykdtree opencv-python==4.6.0.66 opencv-contrib-python==4.6.0.66 trimesh pyrealsense2 matplotlib pyyaml plyfile

二、数据集制作

把arucomarker文件夹里的二维码都打印出来，围着要拍摄的物体贴一圈，尽量是都贴上，要贴平整



在文件夹下运行拍摄脚本(为避免后续的麻烦，文件夹名必须为LINEMOD，后面的Gold替换为自己的物体名)，可以把脚本里的拍摄时间改成50秒，这样会拍出来1100张左右照片，效果更好，不要多于70秒，后续运算的时候你的内存吃不消。拍摄的时候最好是相机不动，物体和二维码转动，匀速转一圈。

✍️ python record2.py LINEMOD/Gold

计算每一帧图片相对于第一帧图片的位移和旋转

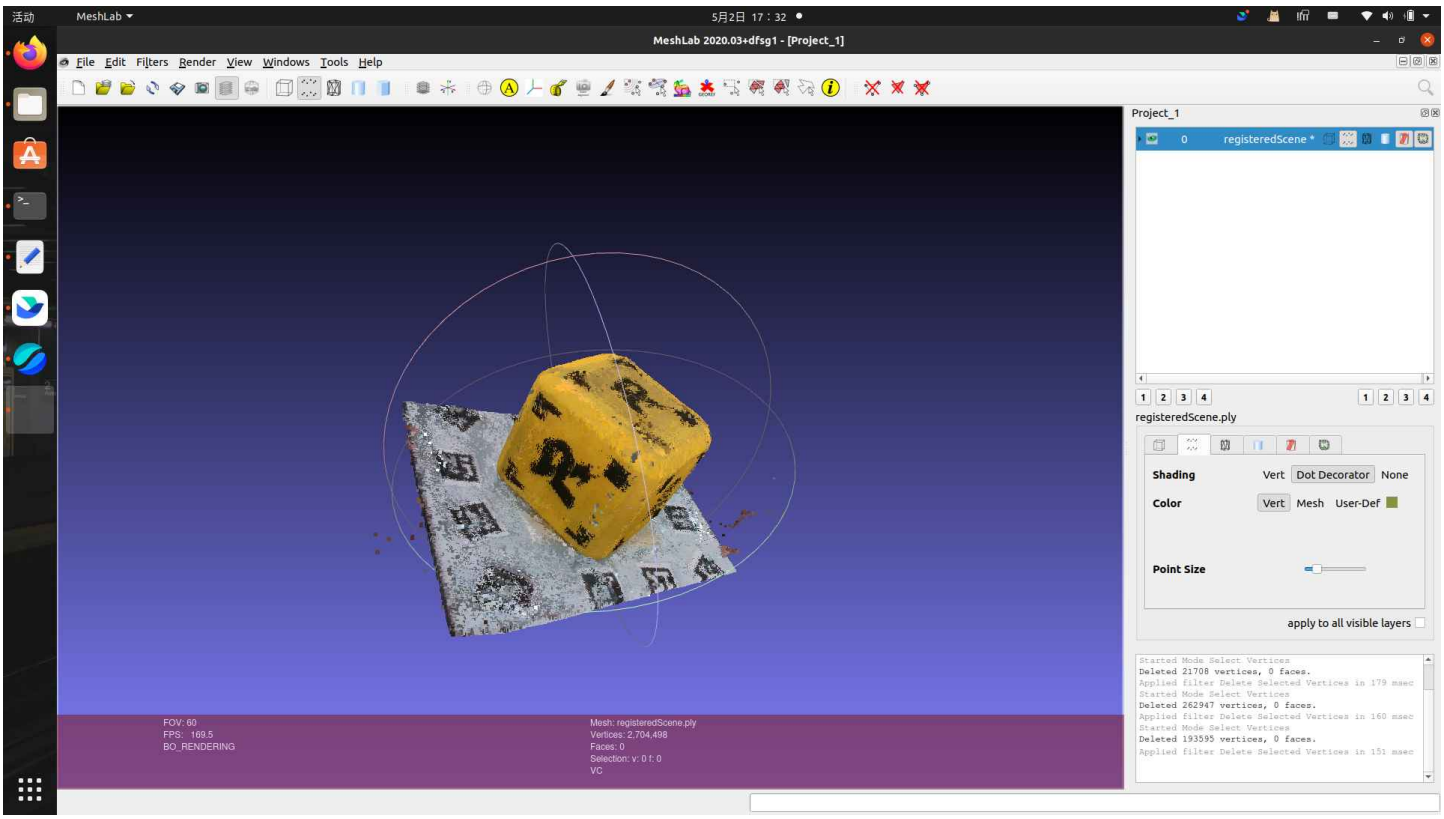
🌰 python compute_gt_poses.py LINEMOD/Gold

然后用register_scene.py进行自动点云整合（这种方法需要用meshlab手动处理点云，也可以用register_segmented.py脚本进行自动曲面重建和底面补全，但是效果一般都不好）

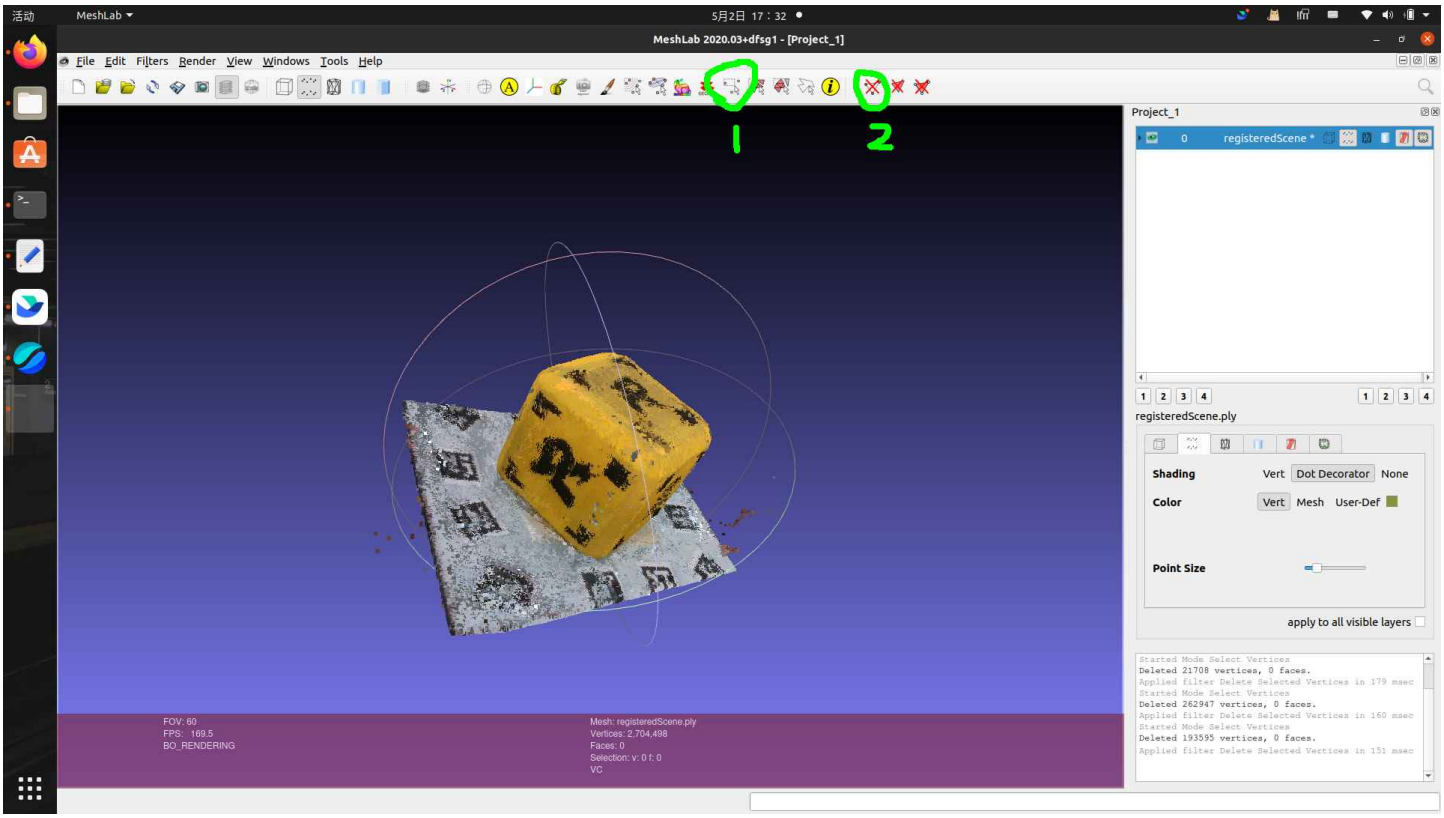
python register_scene.py LINEMOD/Gold

或 python register_segmented.py LINEMOD/Gold （效果一般）

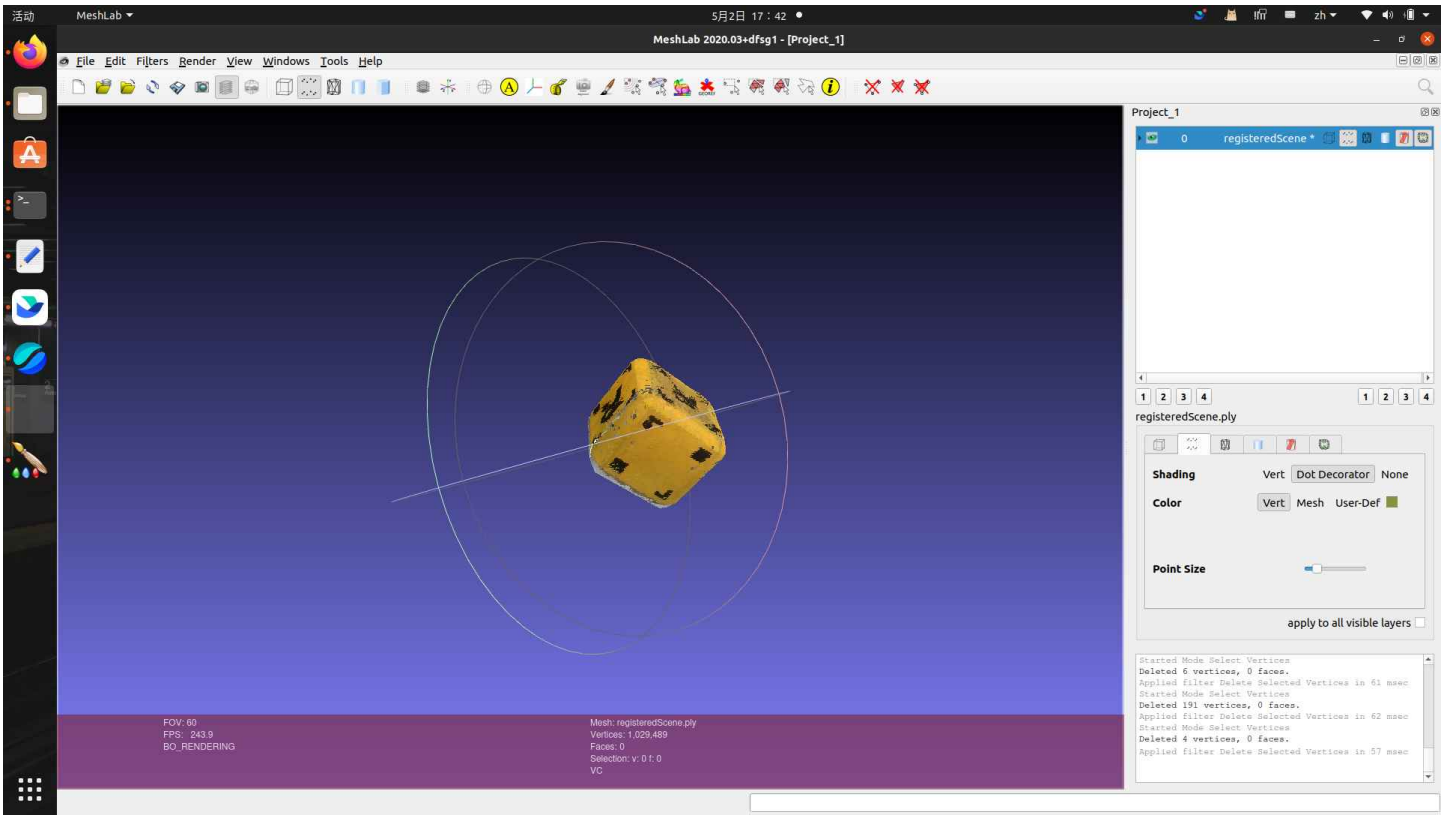
打开LINEMOD/obj_name文件夹找到registeredScene.ply文件，用meshlab打开你会看到这样一个点云



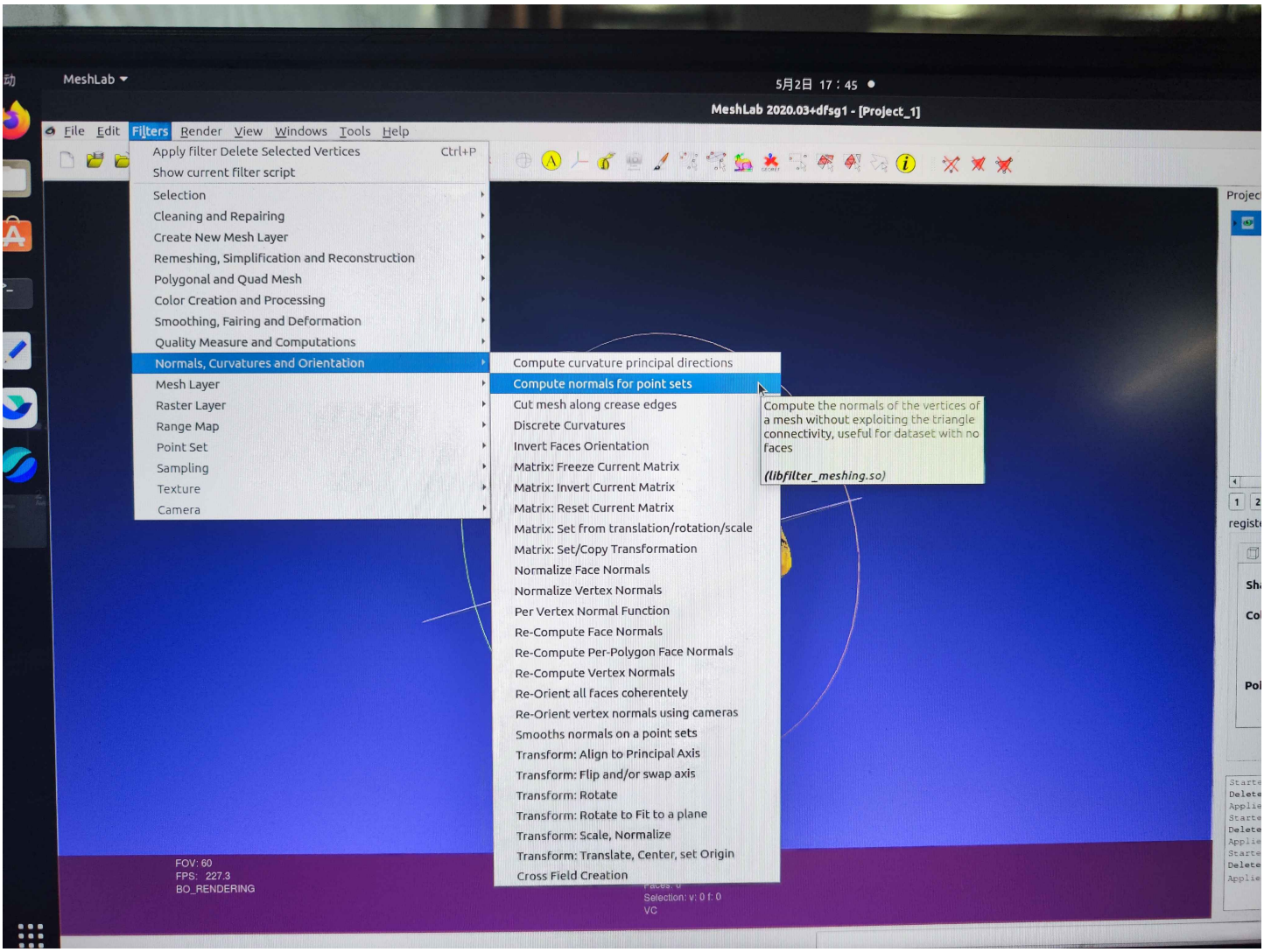
用这两个按键选中、剪裁点云，只保留物体的点云



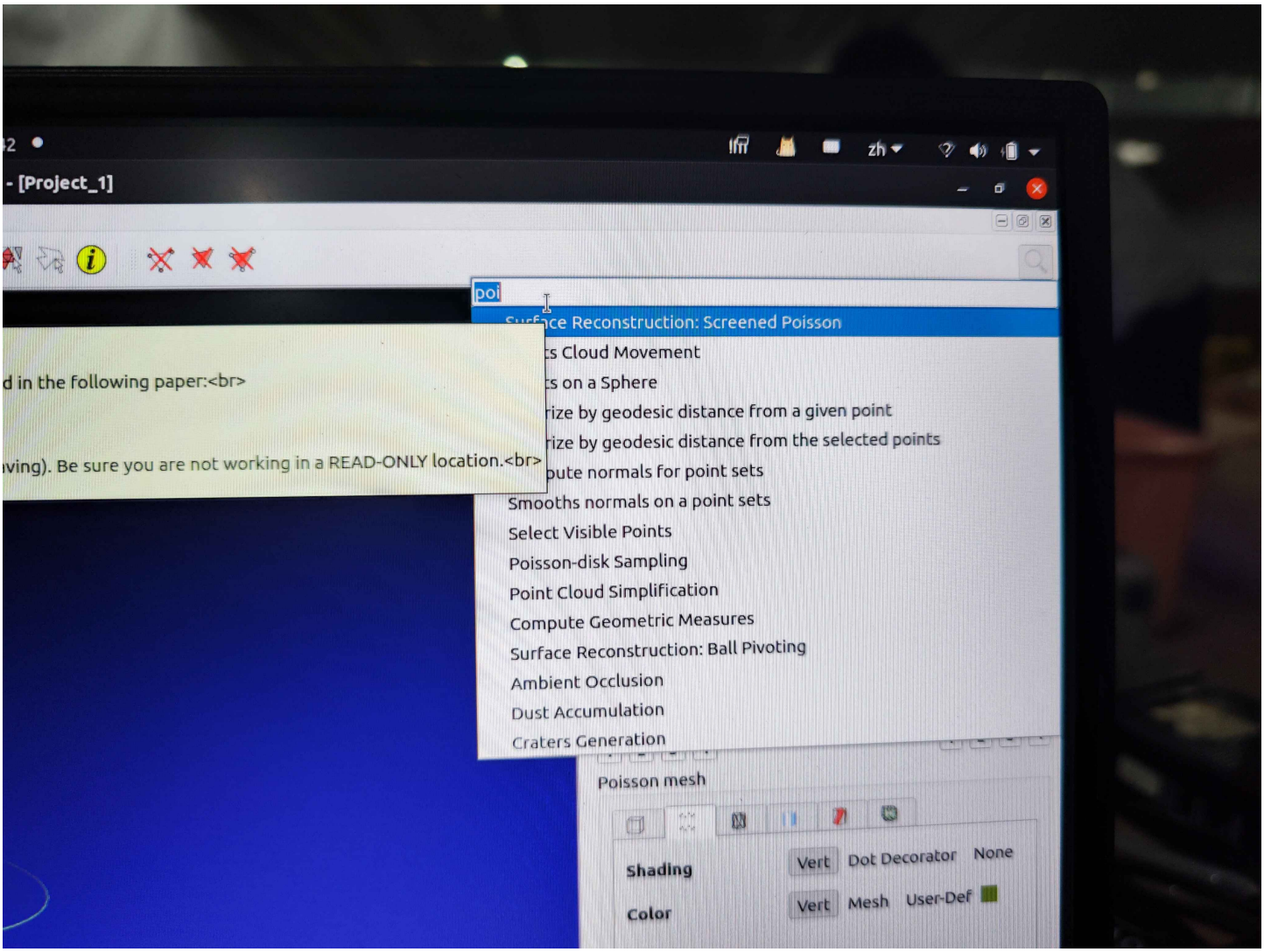
效果如图（底下是空的没有关系，后面会进行重建）



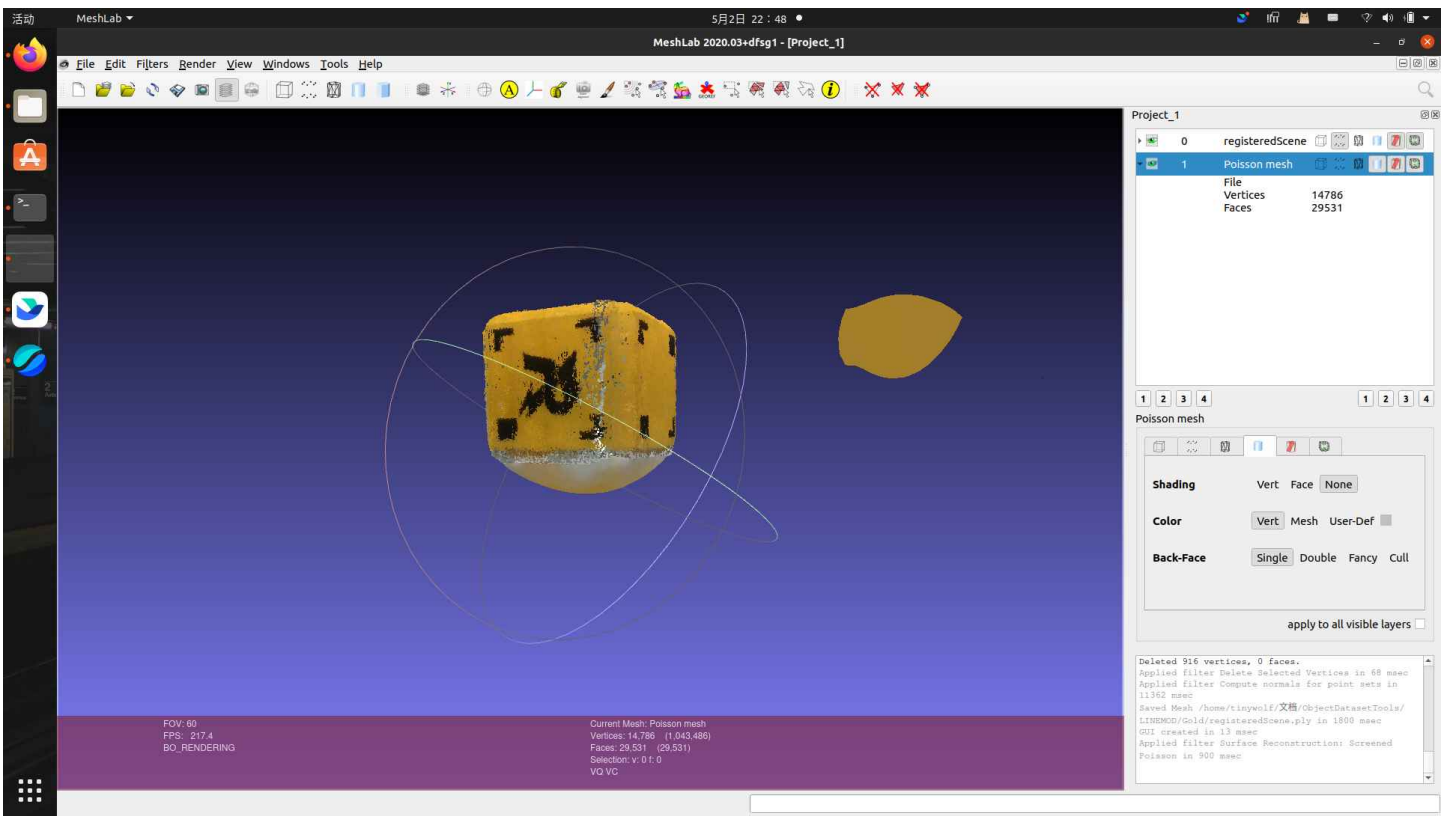
计算点的法向，跳出的对话框里直接点apply就行，默认参数



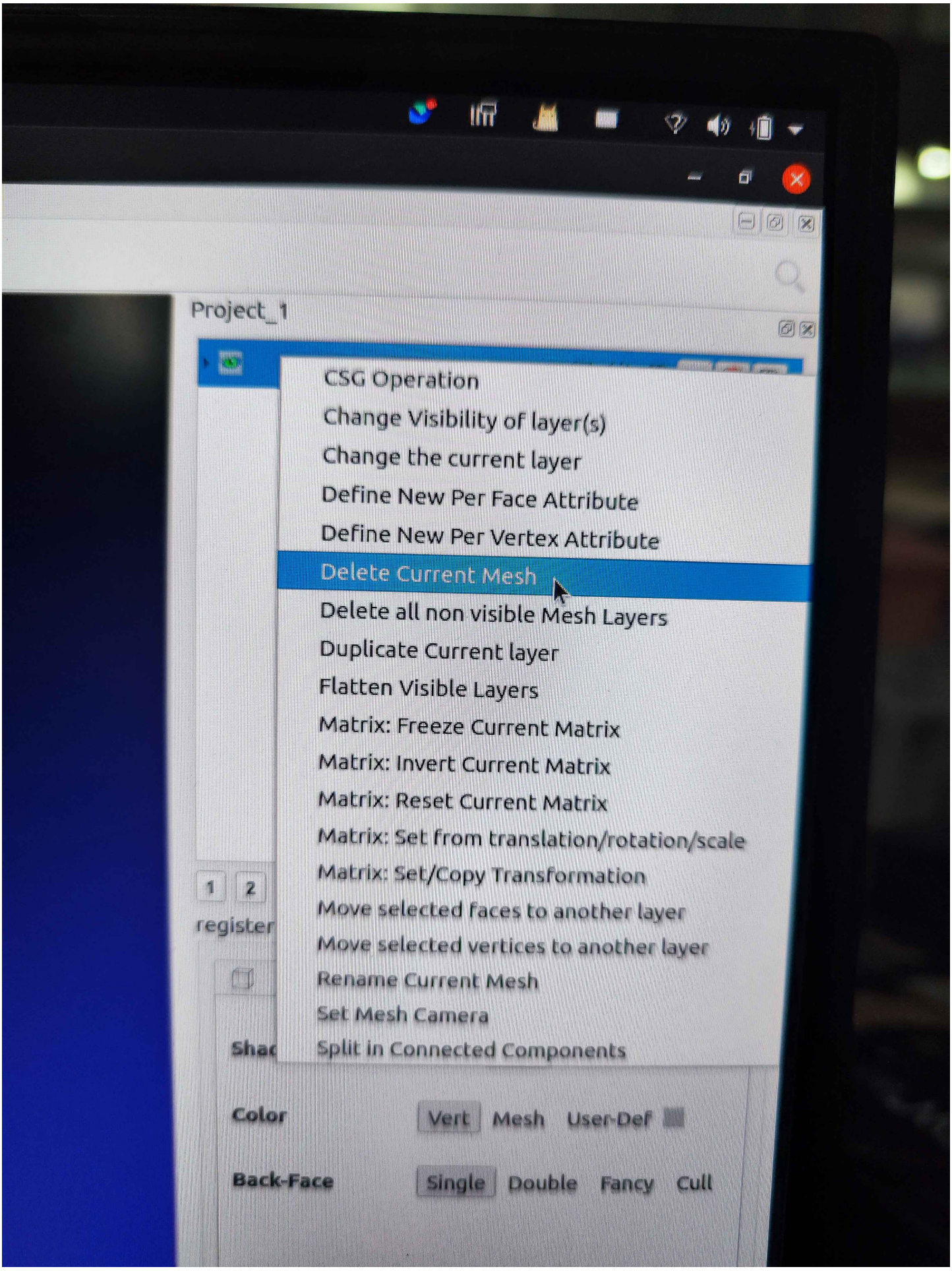
运行完后在右上角搜索poisson，选择第一个表面重建，用默认参数，apply



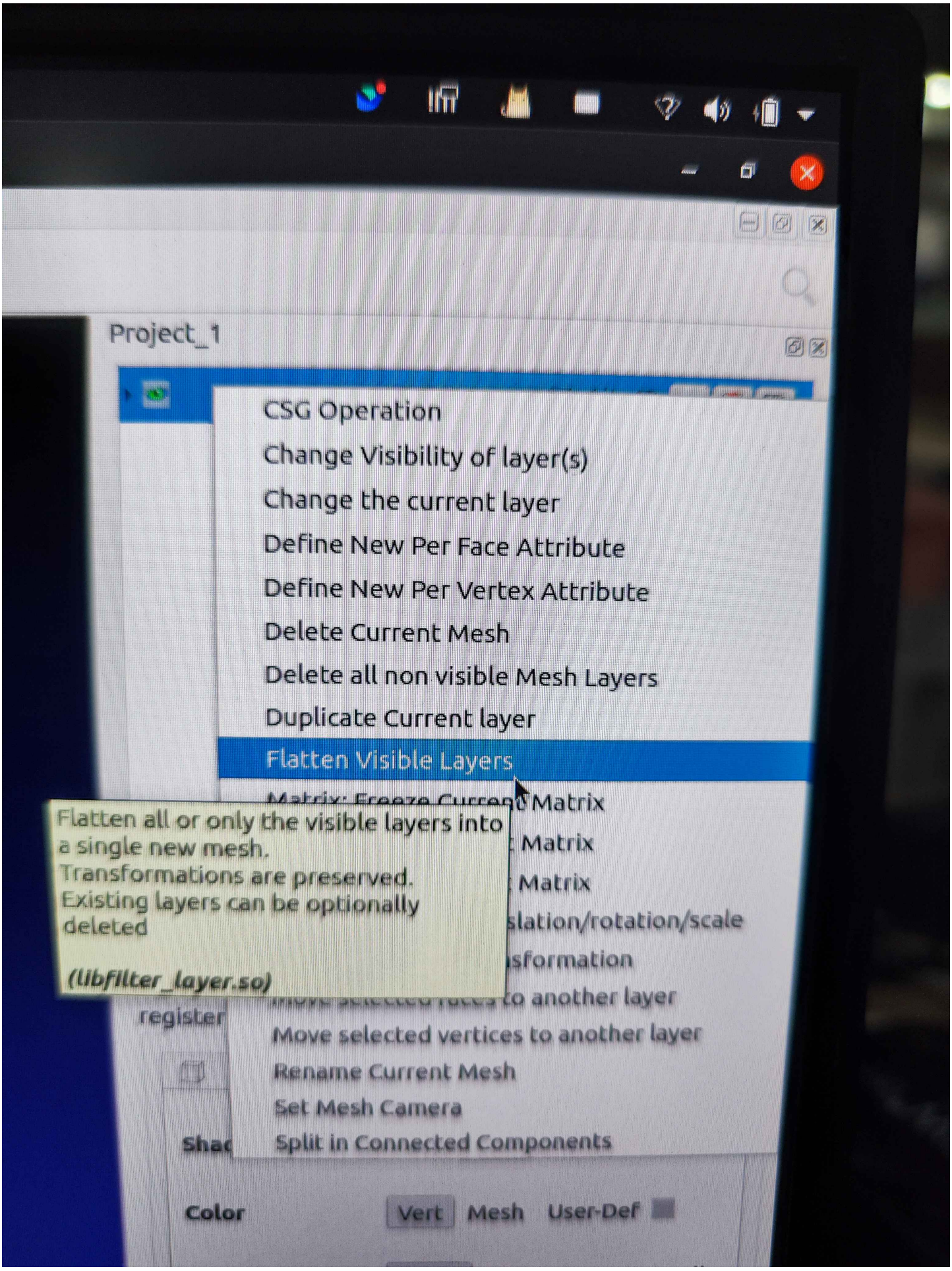
会生成一些奇奇怪怪的东西，不要惊慌



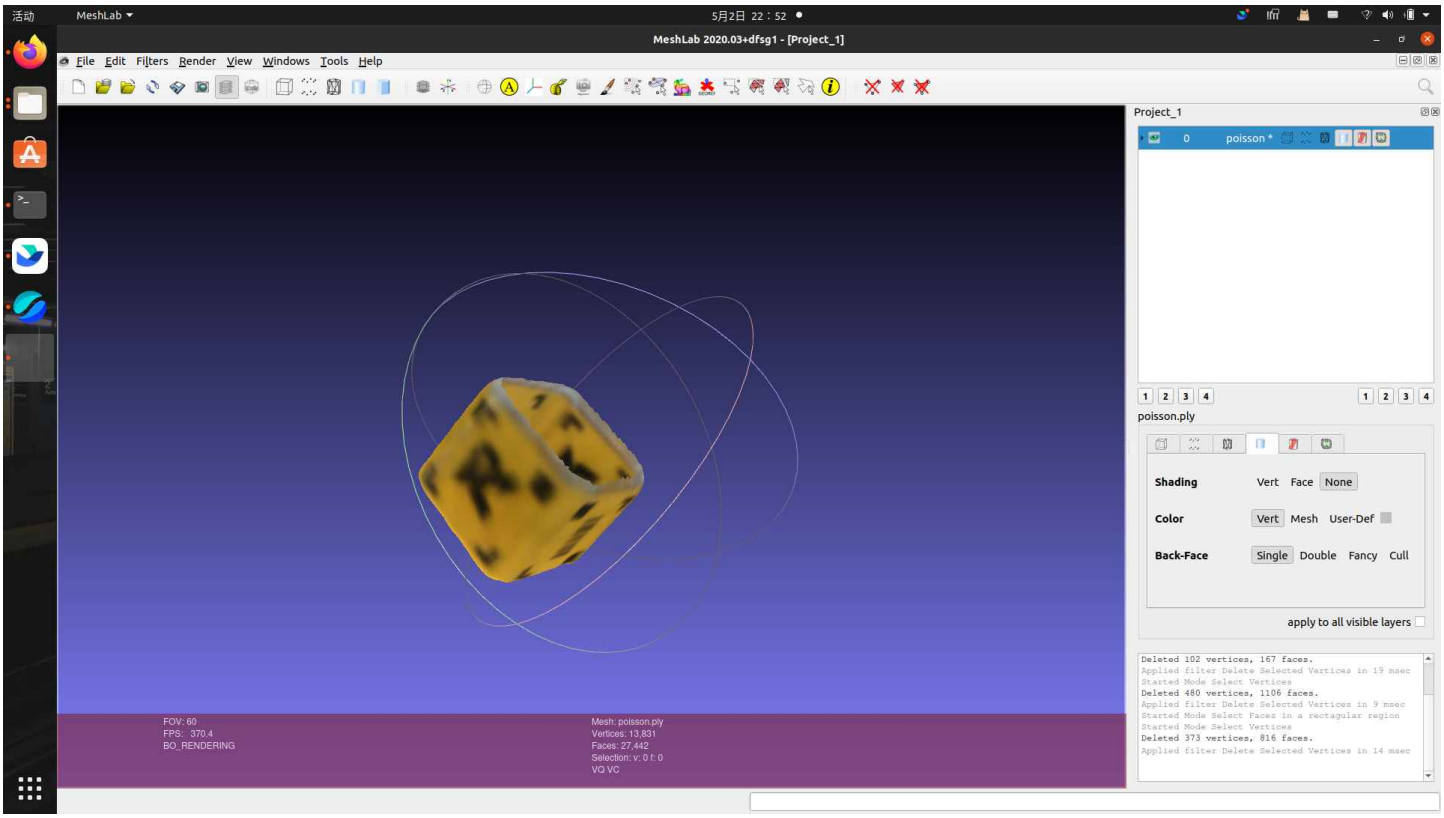
选中原来的那个点云，删掉它，只保留新生成的那个



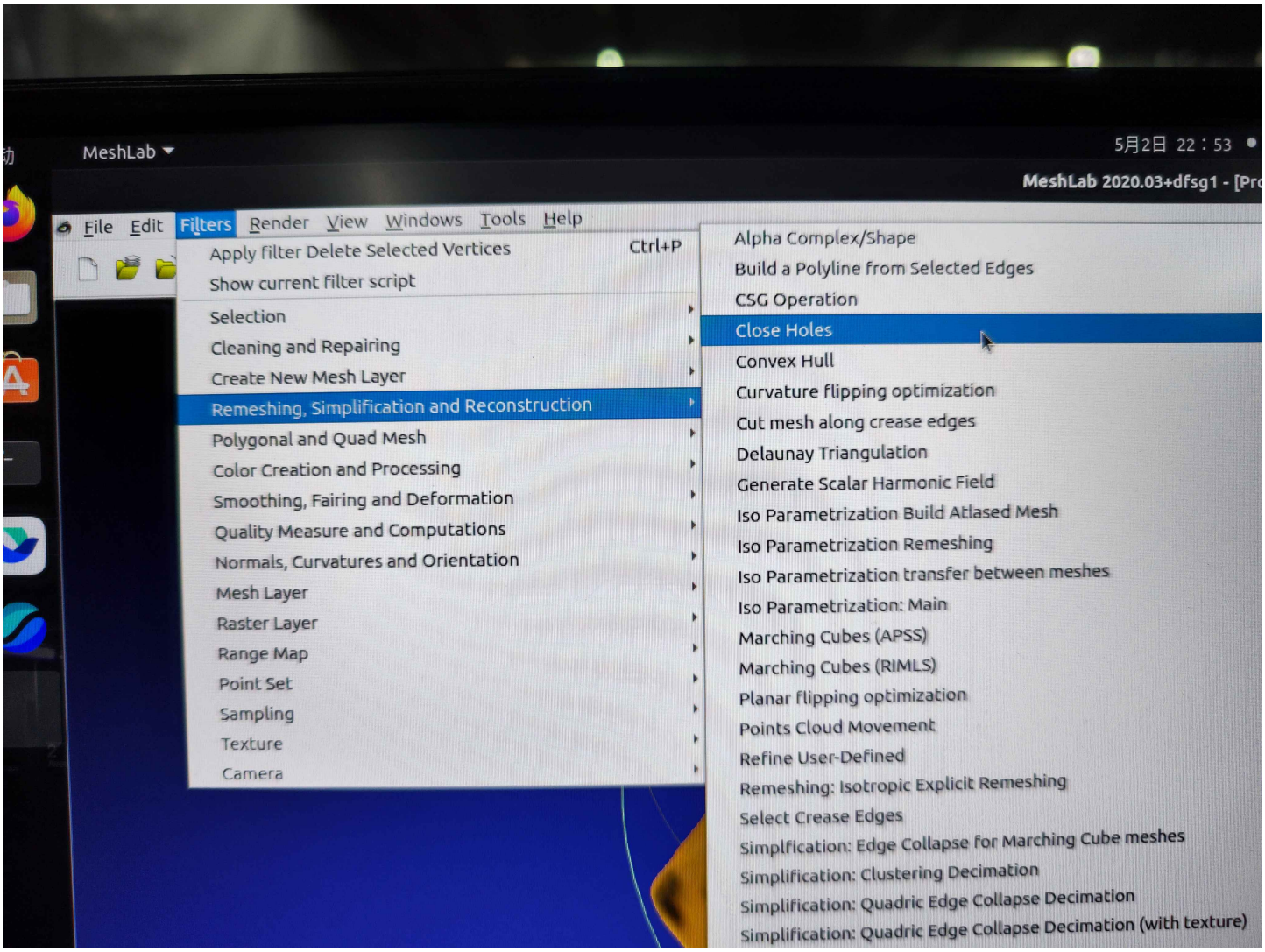
然后对新生成的mesh运行平坦化

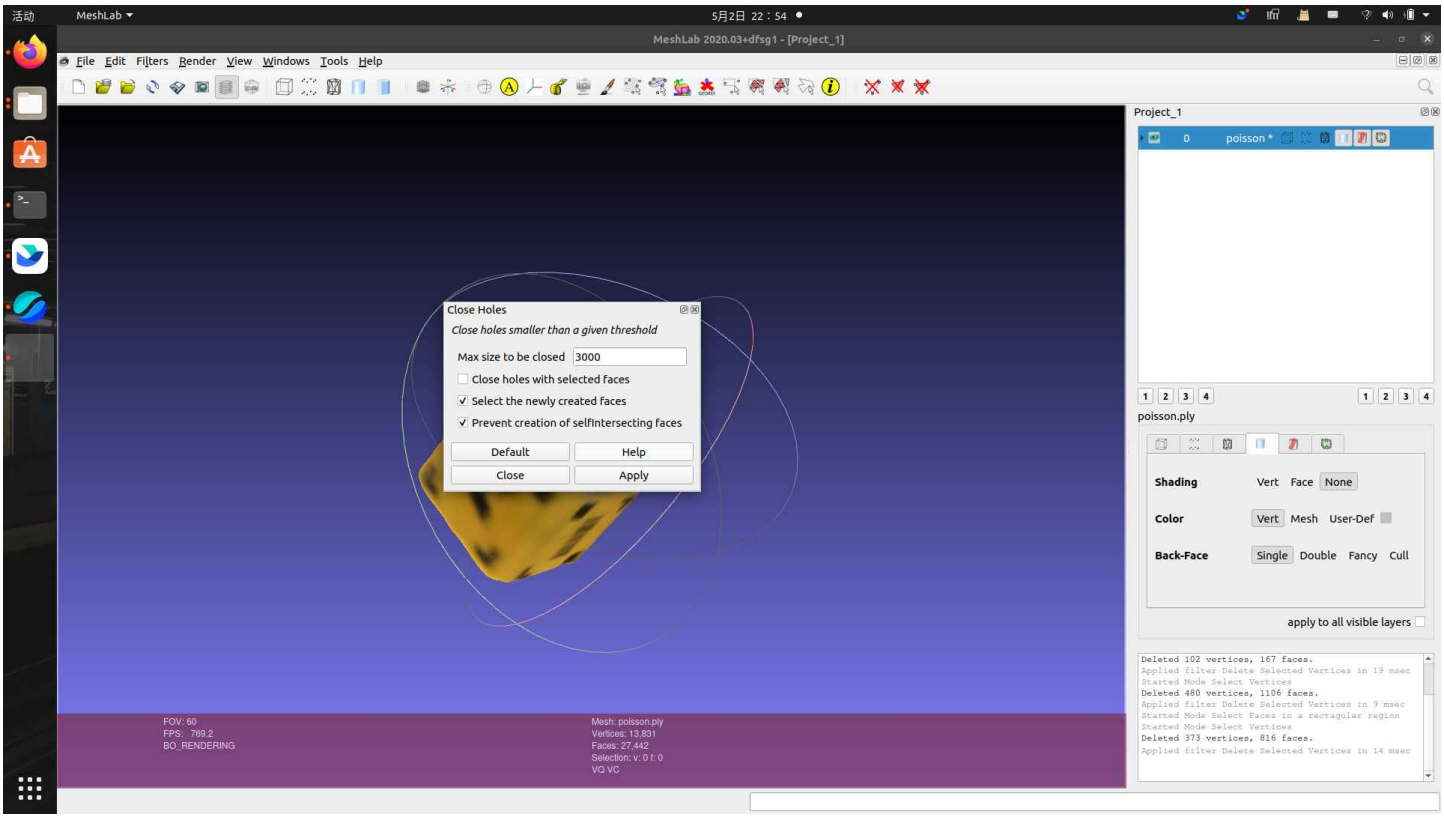


把那些奇奇怪怪的东西切掉

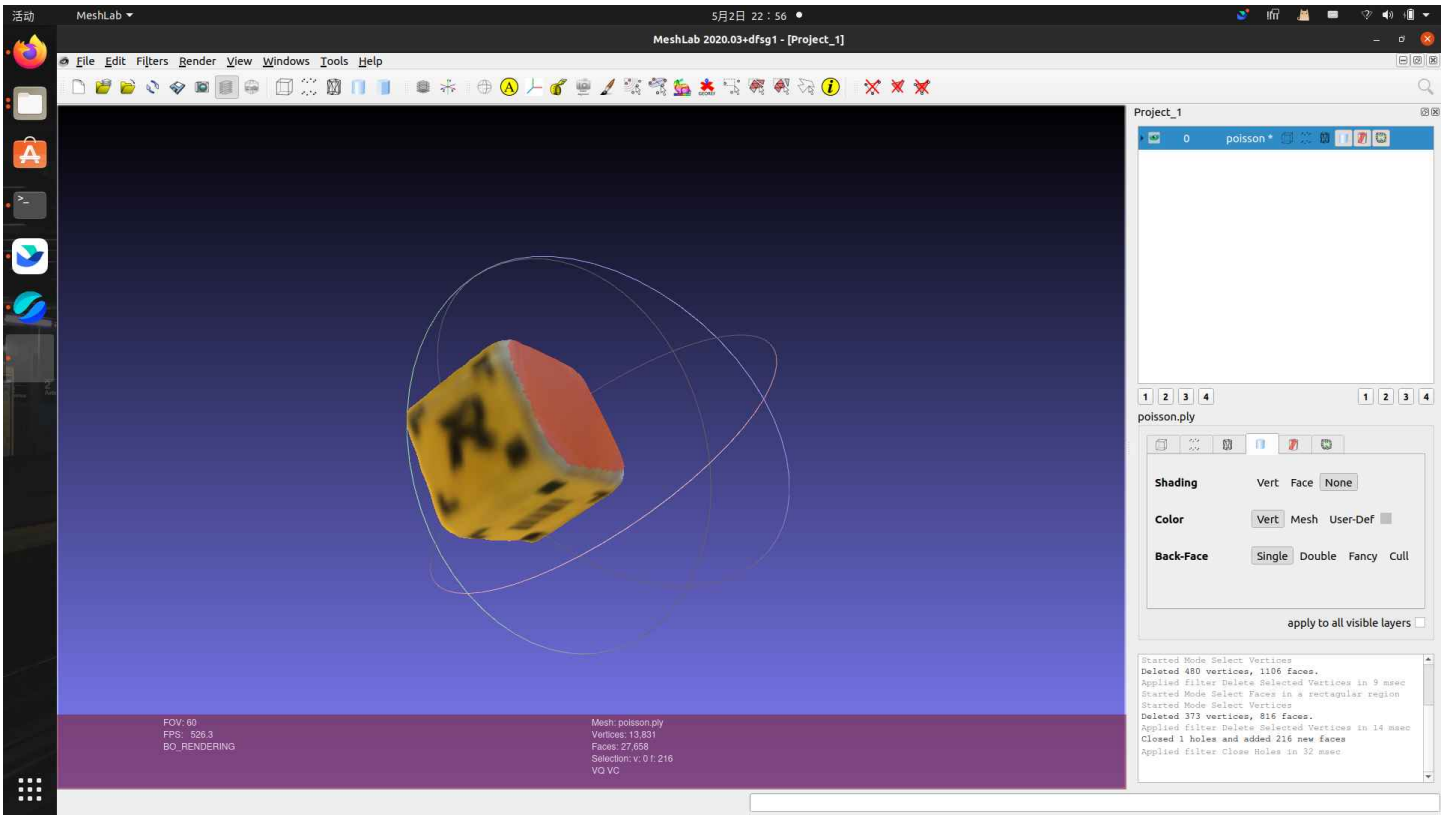


运行补洞的功能，在设置中把面积调大一些

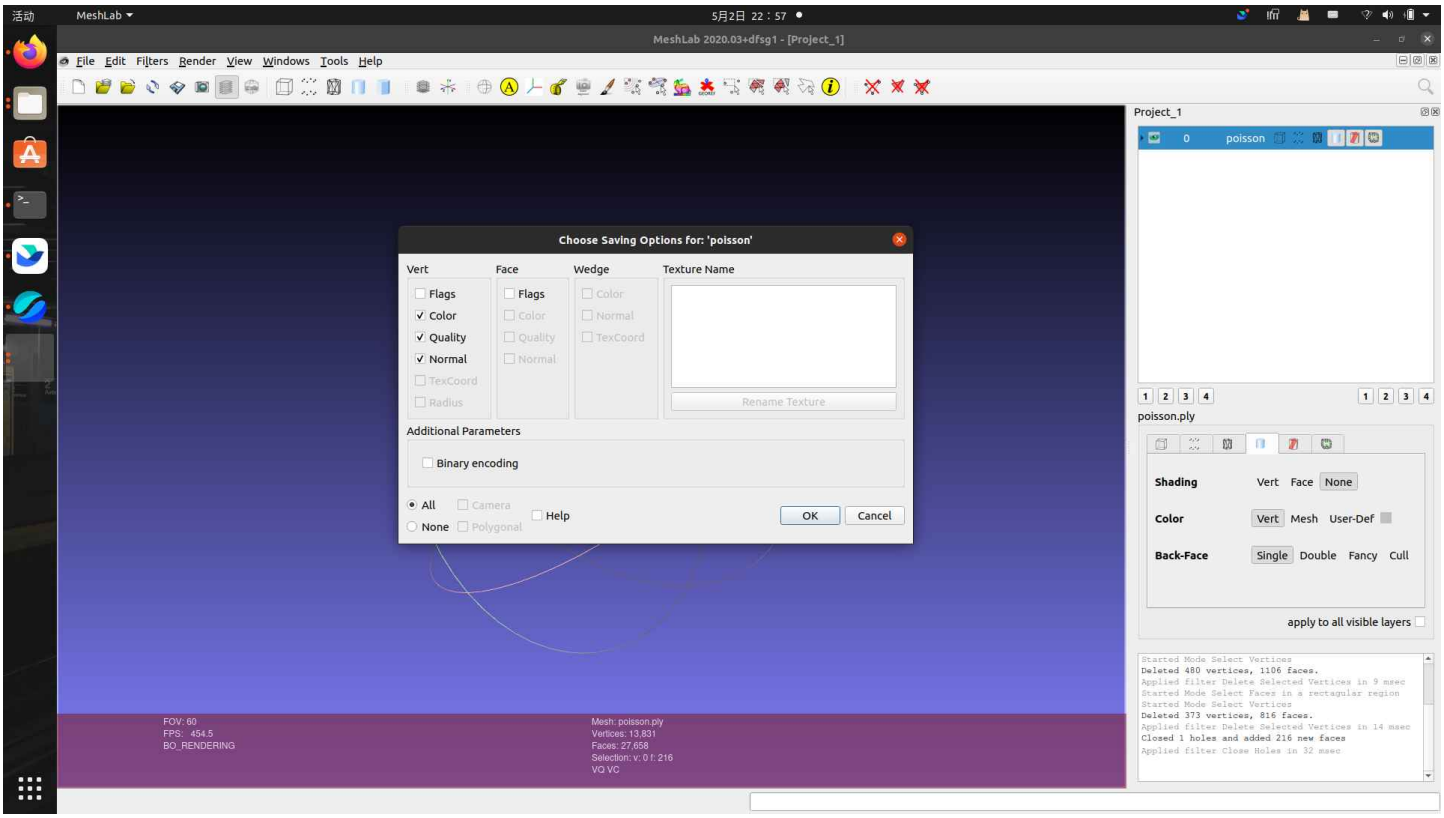




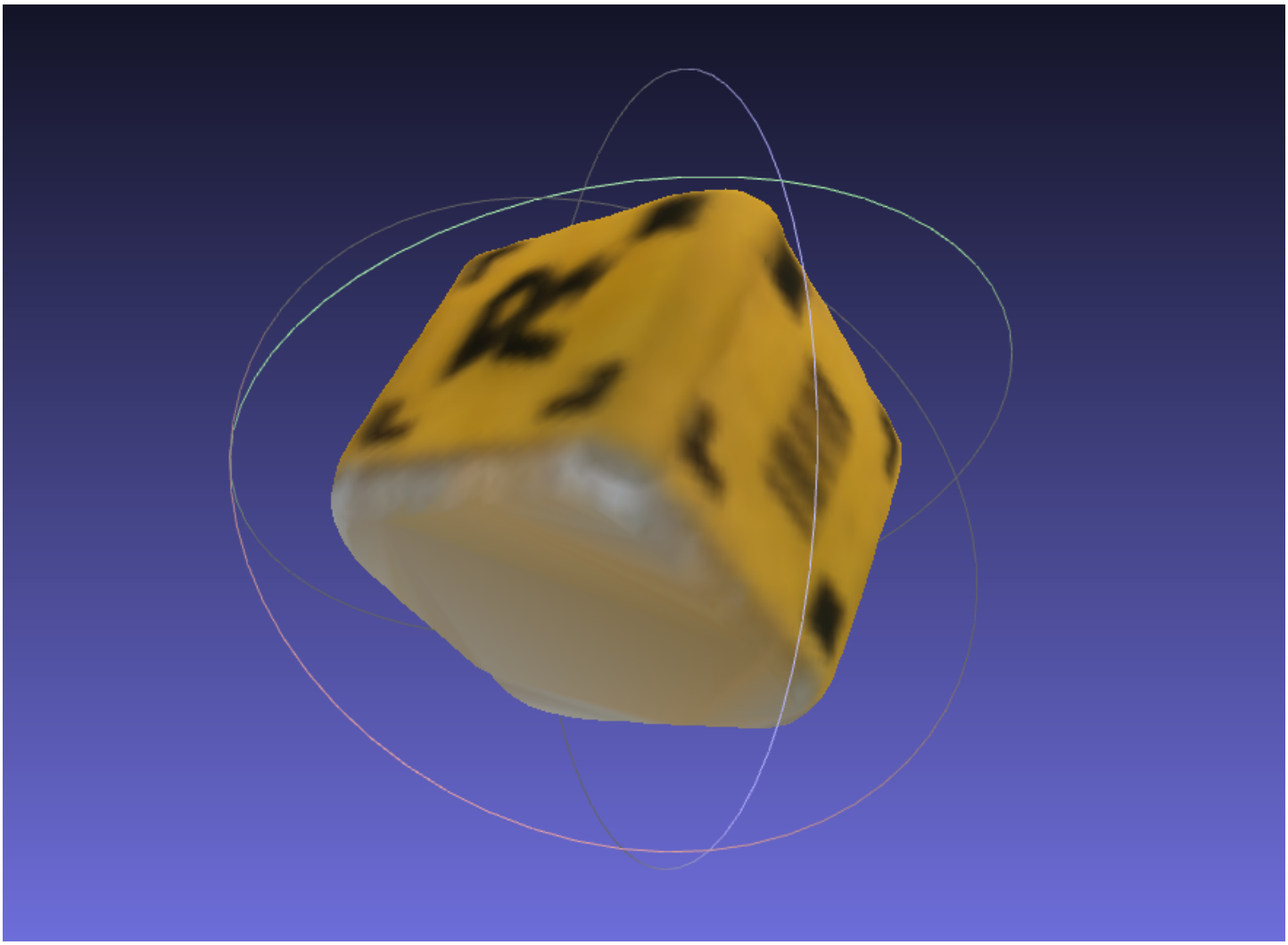
然后就能看到一个补好底面的金矿石了

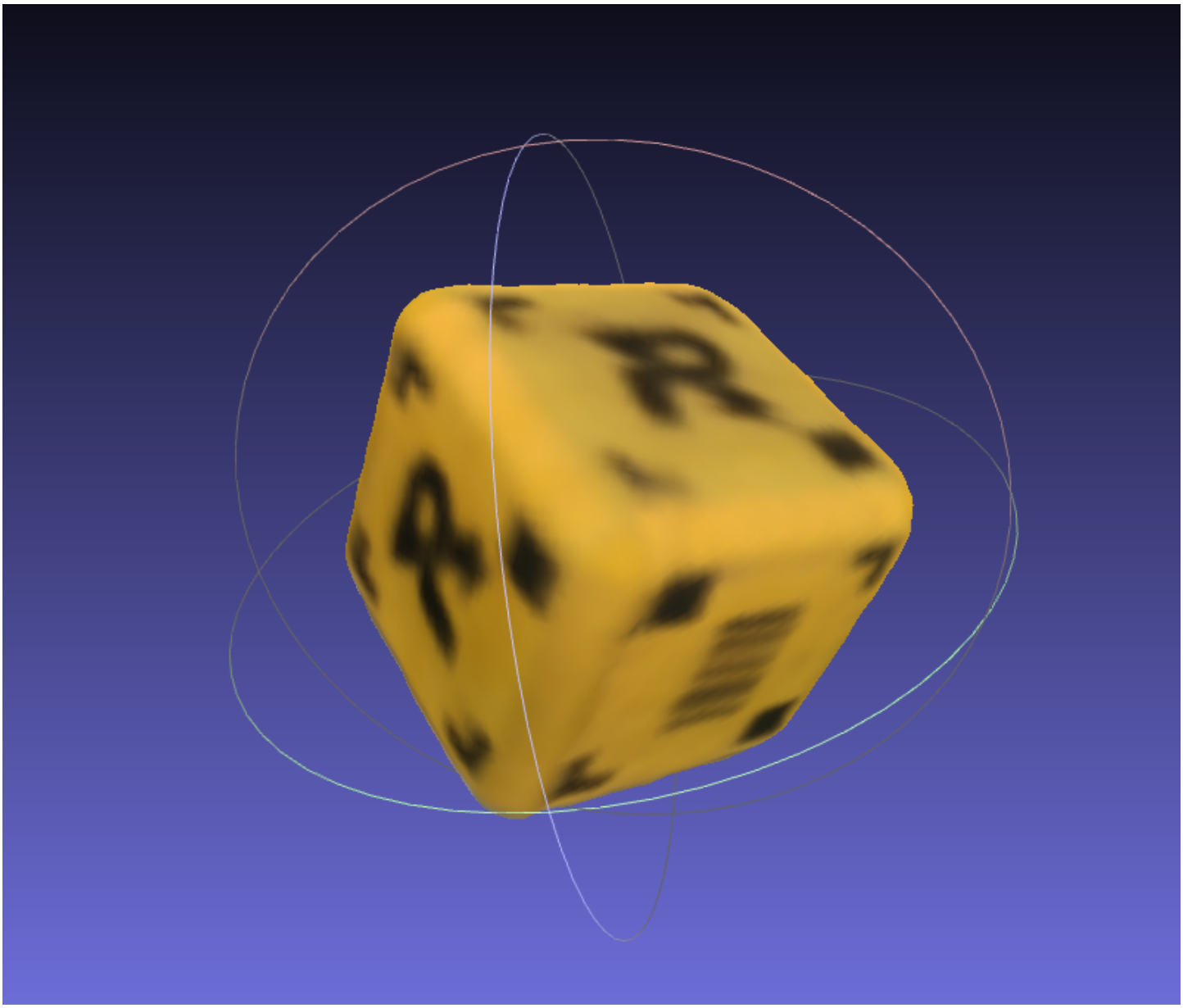


保存，记得把二进制的选项取消掉



重新打开，就可以得到一个漂亮的金矿石三维重建模型了。



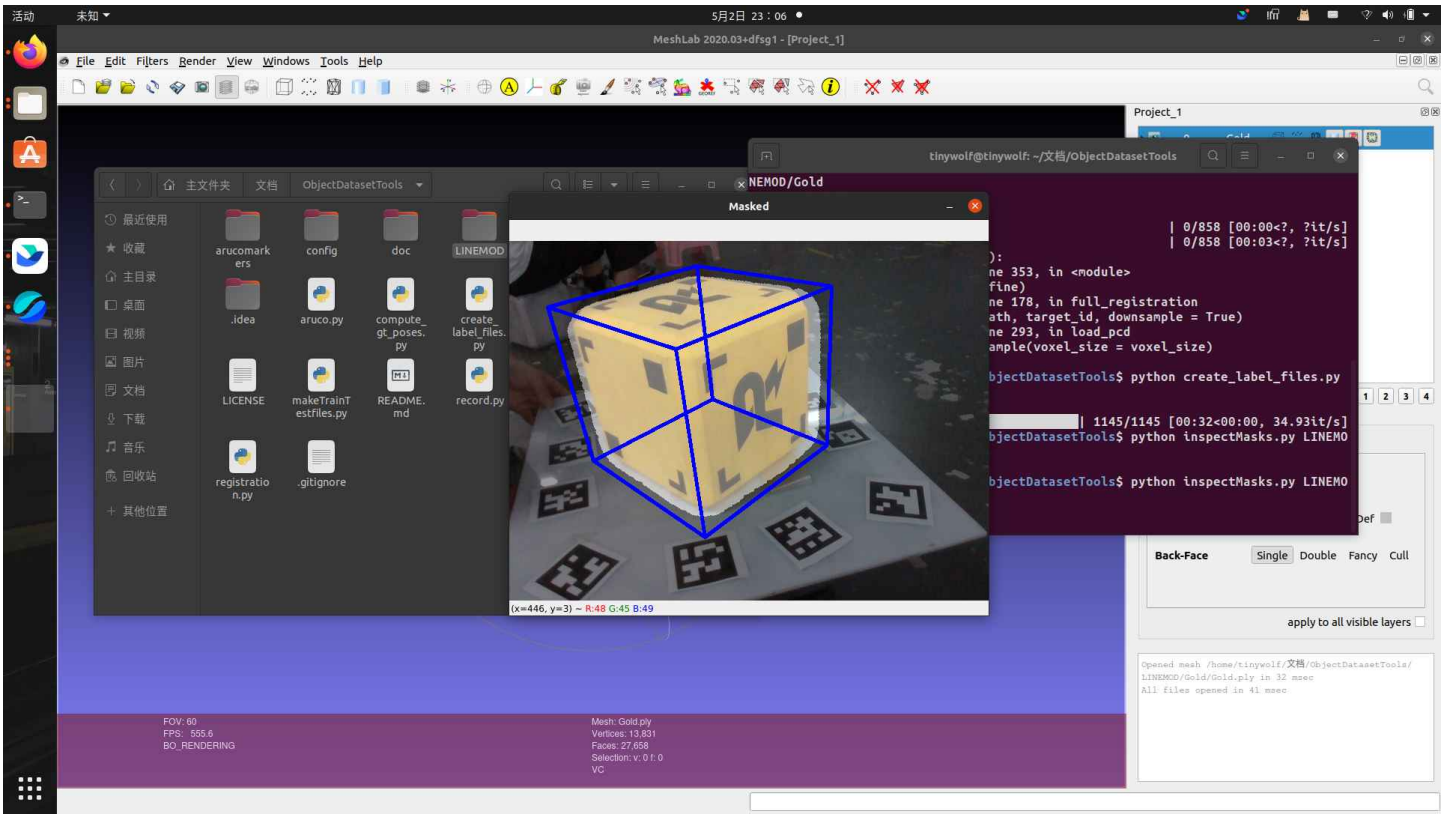


继续做数据集，记得把刚才保存的点云模型名字改成registeredScene.ply，回到工作空间

🎁 python create_label_files.py LINEMOD/Gold

会生成Mask等文件，要想查看数据集做的效果怎么样，运行

🏝️ python inspectMasks.py LINEMOD/Gold



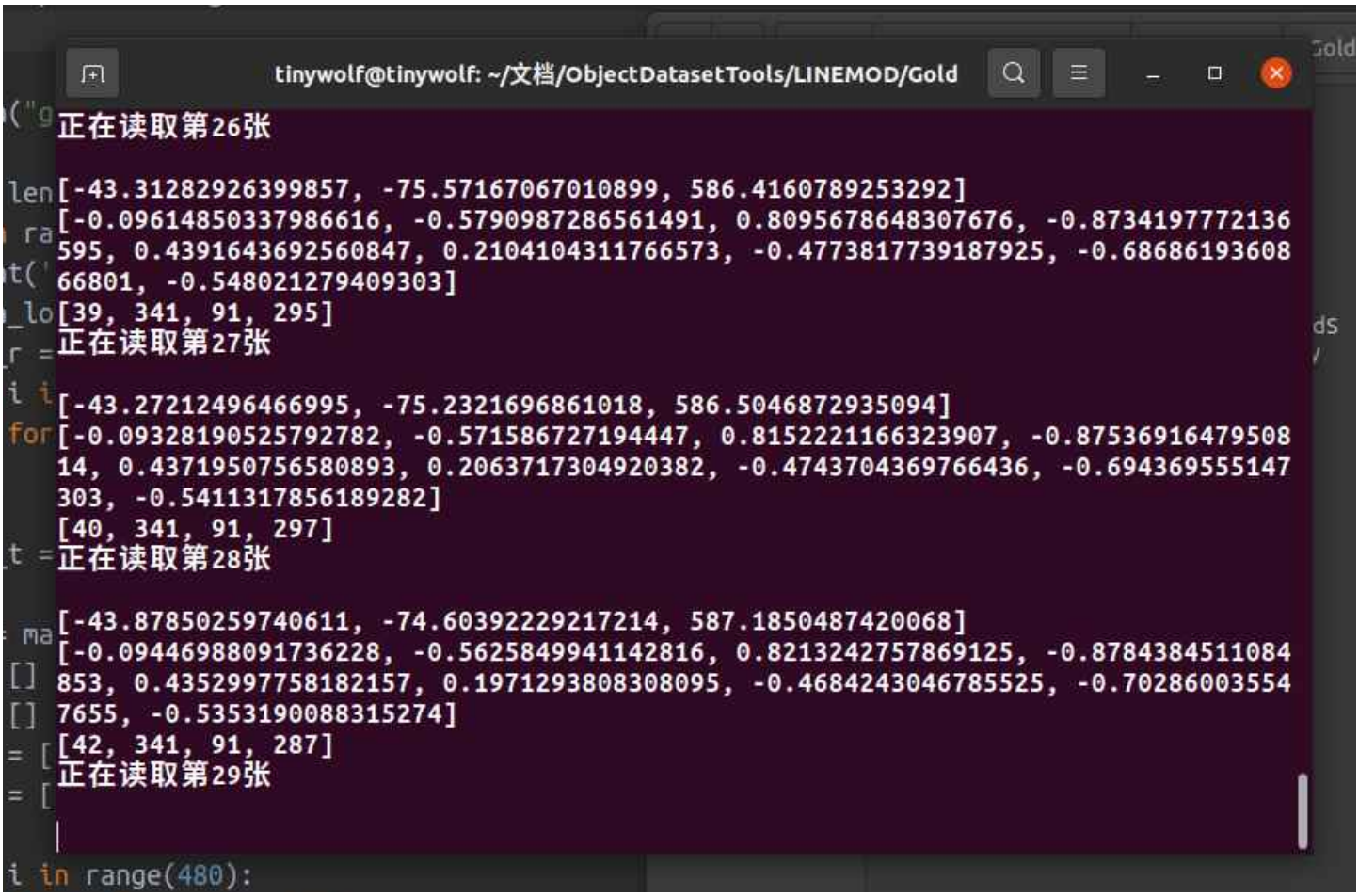
到此为止，数据集主要的工作就进行得差不多了，但是呢，ObjectDatasetTools源码生成的数据集格式有很大的问题，运行以下脚本自动修改数据集格式。

pip install matplotlib numpy pyyaml plyfile

三、数据集的规范化处理

把scripts文件夹里的所有的.py文件复制到LINEMOD/Gold文件夹下，运行脚本生成位移、旋转矩阵文件，读取npy文件挺慢的，慢慢等吧

python gt_file.py



运行改名改格式的脚本，脚本会把原图片的按自然数命名的名字改成0013.png这样的四位数字名称，同时把JPG图片转成png

🦄 python png_rename.py

然后运行info文件生成脚本，用之前记得把info.py里的相机内参改成自己的，可以自己标定realsense也可以直接看intrinsics.json文件

✏️ python info.py

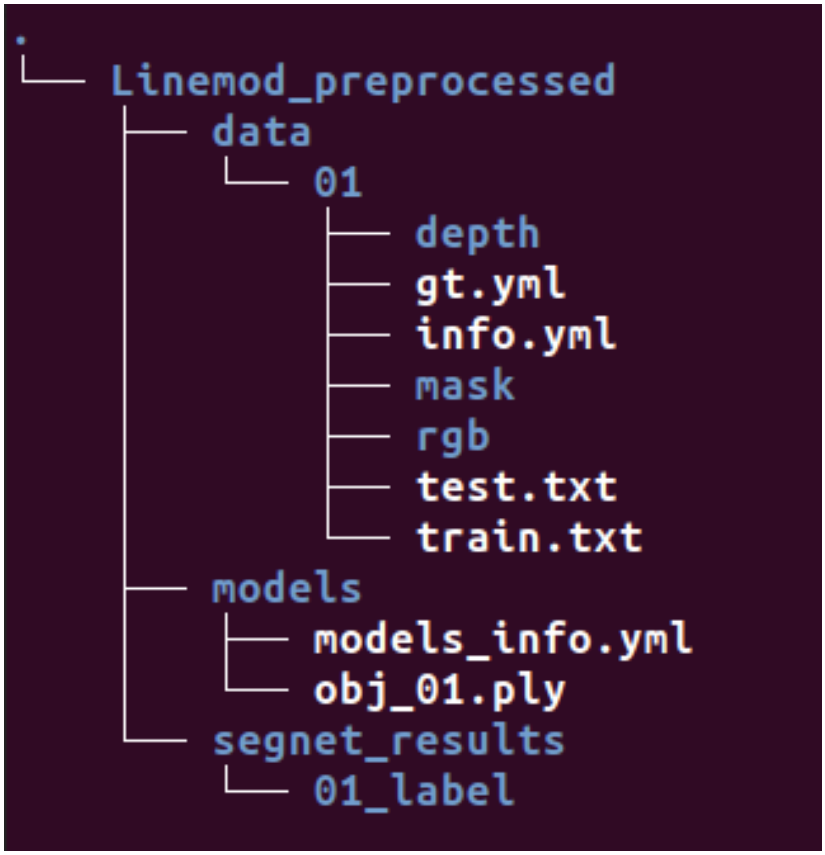
然后运行格式化的脚本，让每个文件夹移动到它应该去的地方,然后你会在LINEMOD/Gold文件夹下得到一个叫做Linemod_preprocessed的文件夹，点开进去，你会看到models、segnet_results、data三个文件夹，这样就成功了

🏝️ python re-format.py

现在就差最后一步啦，运行train_test_txt.py脚本生成训练、测试txt文档

🎁 python train_test_txt.py

最后检查一遍数据集文件夹，是这样一个结构就可以啦



BlenderProc渲染数据集制作方法

有许多游戏制作软件可以渲染物体模型，并模拟摄像机进行拍摄。这里选用Blender作为制作工具，因为Blender以及扩展包BlenderProc提供源码版本的非可视化程序，并可以同步生成深度图以及拍摄同时记录物体位姿，相比于虚幻引擎，具有更强的二次开发性。

一、下载Blenderproc源码并配置环境

下载源码

> 🌟 git clone https://gitee.com/yu-chenghuan/Linemod-Render.git

注意：以下步骤一定要先编译安装bop_toolkit，没有先安装bop_toolkit或是安装失败，后面的安装编译是能够继续的，但是运行代码会报错，想重来要重构库，很麻烦！！

注意：挂好梯子，不然bop_toolkit某些组件下载可能失败。

创建并打开一个python3.8的环境，进入目录下的bop_toolkit文件夹，运行：

> 🌟 sudo apt-get update
>
> sudo apt-get upgrade

```
pip install  --upgrade pip setuptools
pip install -r requirments.txt -e
```

bop_toolkit安装好后，安装blenderproc，二选一。

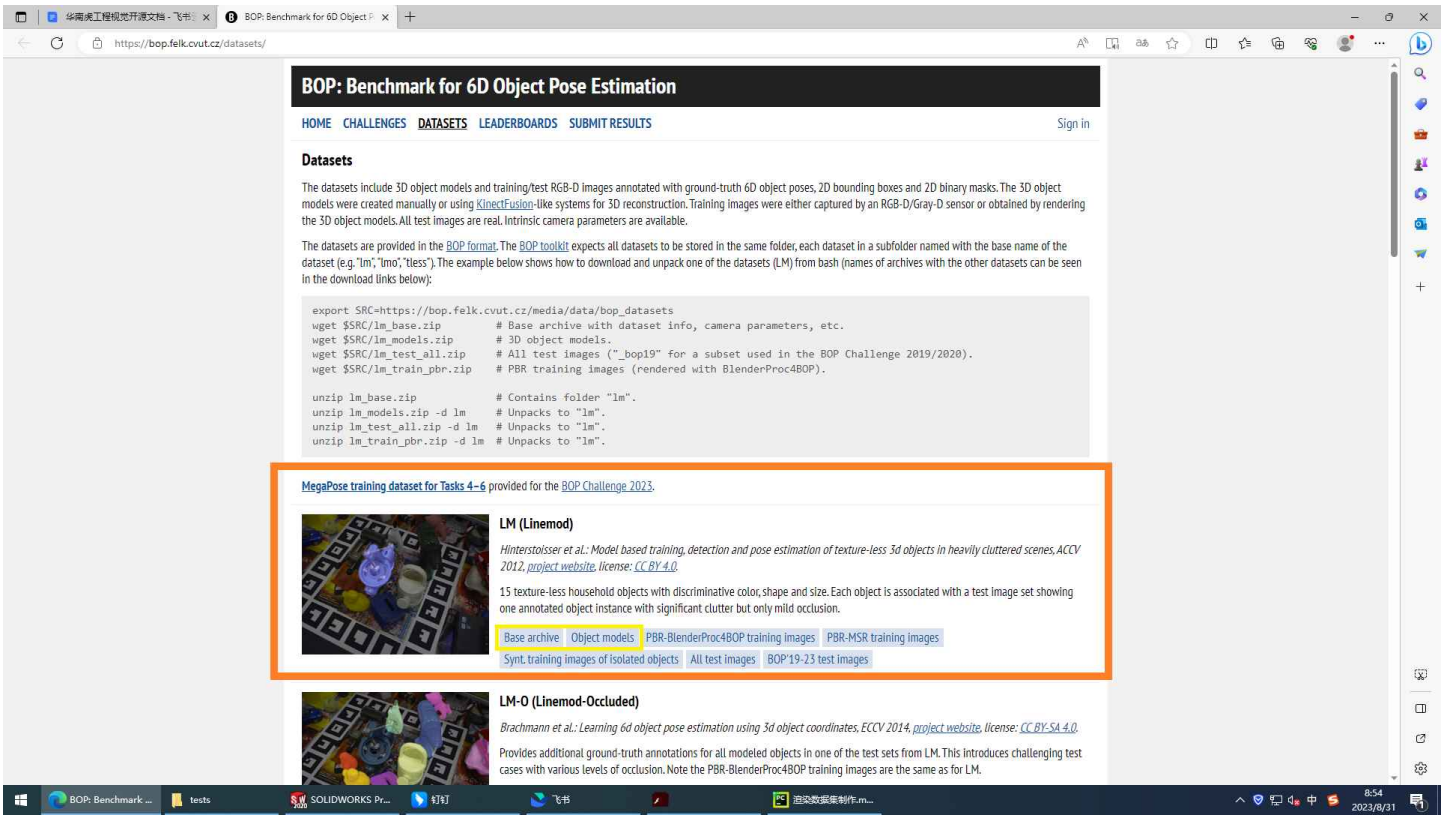
```
🌰 pip install blenderproc
```

或者，进入BlenderProc文件夹

```
🏆 pip install -e .
```

二、数据集制作

进入BOP网站https://bop.felk.cvut.cz/datasets/，下载模型，下载黄色框里的那两个就可以，其他的不需要。



把下载的压缩包解压，把models文件夹放在lm文件夹目录下，models文件夹中的models_info.json和obj_xxx.ply都替换成自己的模型数据。lm文件夹里的camera.json替换成自己的相机矩阵，简单标定一下，差不多准确就行。

新建一个文件夹backgrounds一个outputs，终端运行

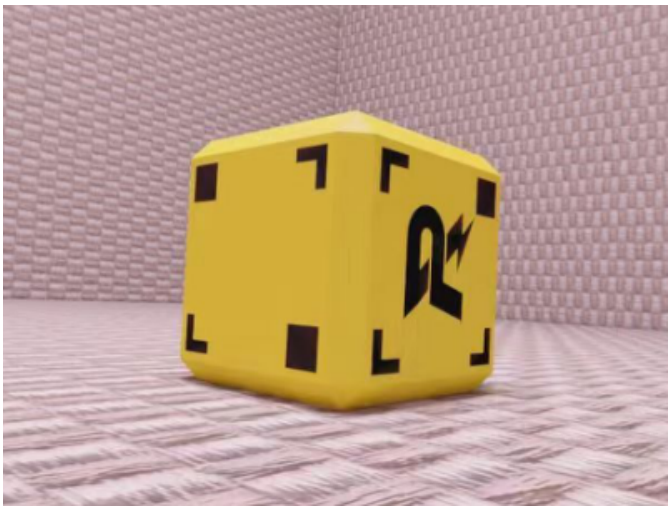
```
blenderproc download cc_textures ./backgrounds
```

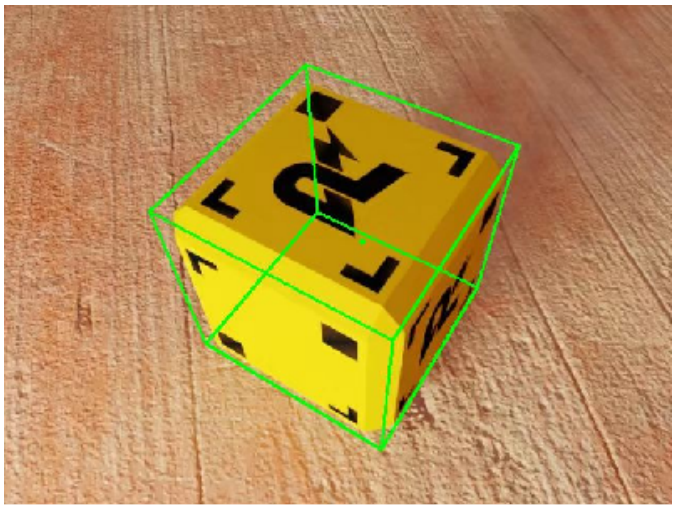
这一步会下载特别多的场景，每一个文件夹是一个场景的贴图，会下几百个，个人觉得50个左右就够用了，感觉差不多就Ctrl + c就行，不然后面渲染出的数据集会特别大。下载完成后，进入BlenderProc文件夹运行

```
blenderproc run examples/datasets/bop_challenge/main_lm_upright.py .. ./backgrounds ./output --num_scenes=20
```

..为刚才处理好的放了models文件夹的lm文件夹的父目录。

然后就是漫长的等待，建议用一个显卡性能强的服务器。如果你觉得数据集够用的话直接结束进程就好，不知道这个代码渲染的上限数量是多少。我修改了num_scenes（每个场景循环次数）的值和修改了代码里for循环的次数，改到很低，30多个小时仍然渲染出了20多万组图片，而且代码并没有结束。





生成出的数据集就是这样的，右图为数据集位姿数据可视化

到这里，需要的数据都已经生成完毕了，接下来是格式处理

三、数据集的规范化处理

进入bop_toolkit文件夹下运行

```
python scripts/calc_gt_masks.py
python scripts/calc_gt_info.py
python scripts/calc_gt_coco.py
```

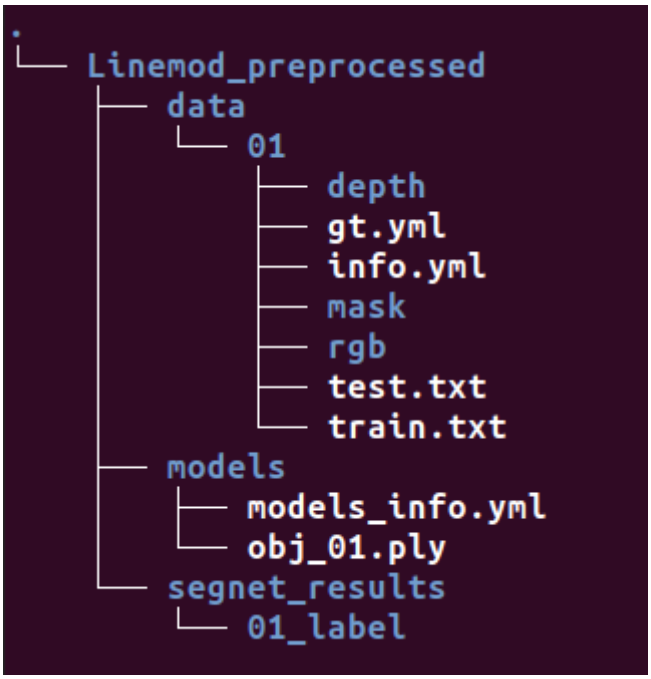
```
上述代码中均更改以下部分，其他的保持不变
p = {
  # See dataset_params.py for options.
  'dataset': 'lm',

  # Dataset split. Options: 'train', 'val', 'test'.
  'dataset_split': 'train',

  # Dataset split type. None = default. See dataset_params.py for options.
  'dataset_split_type': 'pbr',

  # Folder containing the BOP datasets.
  'datasets_path': '<path to your datasets>',(为lm父文件夹的位置)
}
```

三个程序全运行完后，运行：

📍 python gt.py

python info.py

以上两个代码需要在最后的main函数里修改指定文件的路径，注释已经标明。



mask_visib就是segnet_results的掩模图，最后仍是按这个格式梳理数据集，lm数据集中的图片全是png格式，格式修改以及文件名的修改一些细枝末节就交给你啦。

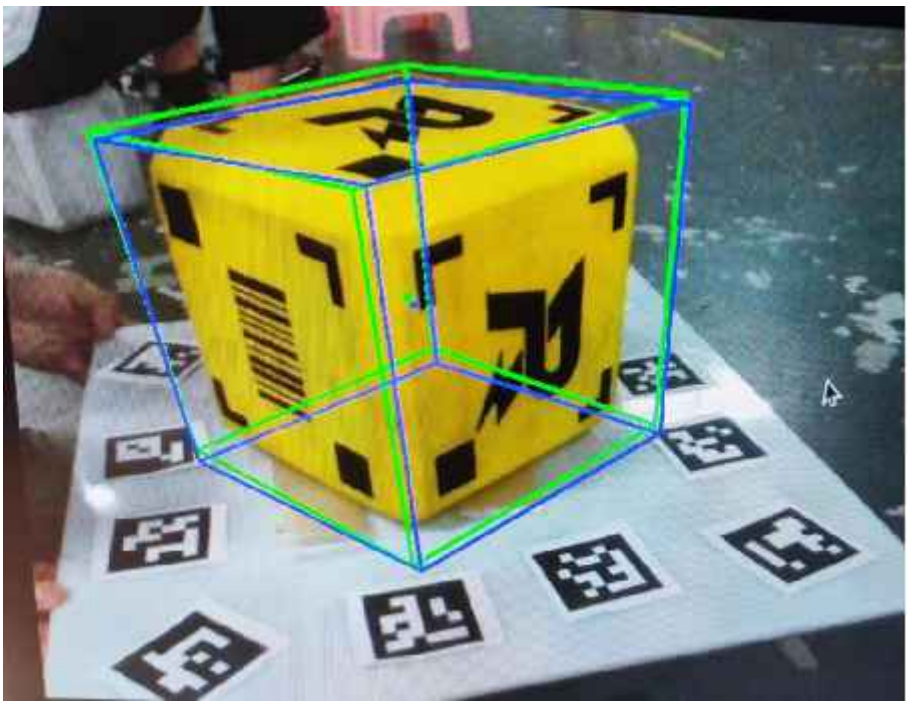
Linemod数据集制作总结

以上介绍了两种lm数据集制作的方法，分别是真实数据点云重建自动语义分割方法，和BlenderProc渲染数据生成方法。每一个方法在网络训练、数据集制作等方面均有明显的短板。

比如真实数据很难一次制作足够大的数量，而且背景、光照等环境因素不能做到多样化，所需的时间成本很高。即使应用了数据增强，网络仍会出现严重的过拟合，也就是训练集、测试集表现都很好，但是用相机部署实测，姿态估计的结果完全错误。

而渲染数据集由于场景变化多样，物体位姿跳变大，训练时损失值很难收敛。

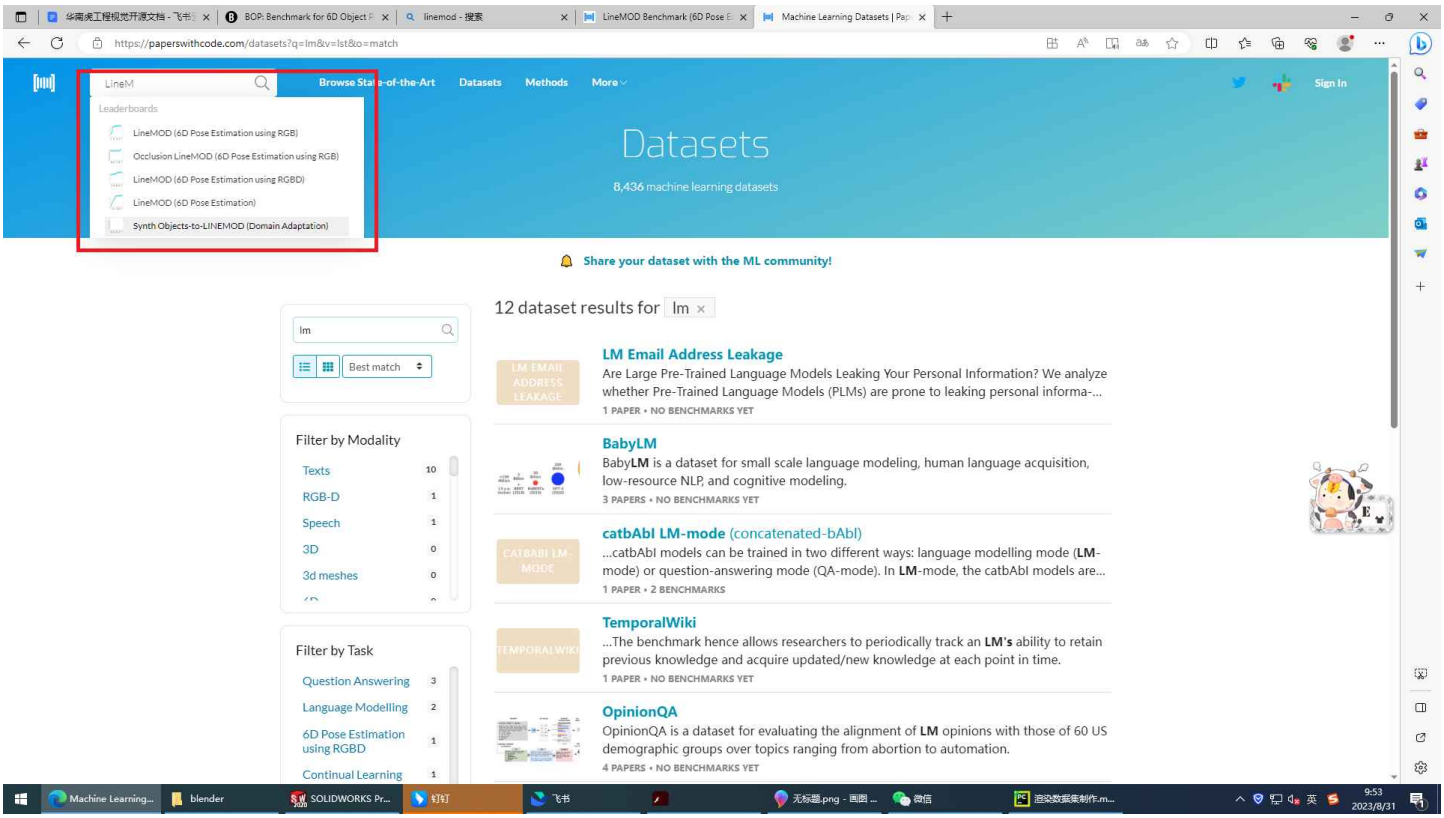
所以，不如将两者结合，取长补短。我用了单一背景的2000组真实数据的数据集给模型进行预训练，也就是模型预热。然后将10个场景下拼接起来的真实数据集共计20000组图片，以及48个场景下BlenderProc渲染数据30000组图片，做成一整个数据集，进行网络训练。在实际测试中取得了不错的效果（算法为EfficientPose，phi=0的复杂度最低模型）。



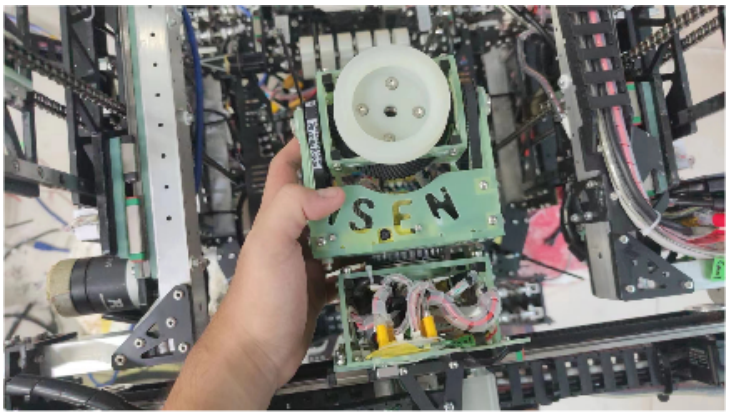
绿色框为数据集数据，蓝色框为网络推理结果

6D-of位姿估计算法选型

进入paperswithcode网站https://paperswithcode.com，选择dataset，输入LineMOD，可以看到很多运用lm数据集不同类型的算法，而且还贴心地根据算法的表现进行排序。有单张rgb进行推理的，也有rgbd推理的，根据自己的需求选型就行。



算法推荐



由于我们的方案是小三轴前端放一个小的模组相机，所以选用的算法是基于RGB的**Efficientpose**。Efficientpose算法论文简洁易懂，Readme文档写得极其详细，代码基于keras架构，部署简单，而且训练时提供非常多的选项，包括模型复杂度，是否数据增强、是否Freeze-Backbone等等。又由于其是2020年的算法，环境配置简单，众多开发者做的复现教程以及改进方案都很丰富。

不过，如果机械结构允许，可以安装深度相机的话，还是更推荐基于RGBD的算法比如**PVNet**等等，多出的深度数据，无疑会让位姿估计的结果更精确。

以上便是2023赛季华南虎工程视觉的全部开源资料，感谢大家的阅读，也欢迎大家添加我的微信交流。

余承寰

微信号：Tiny_ych_wolf