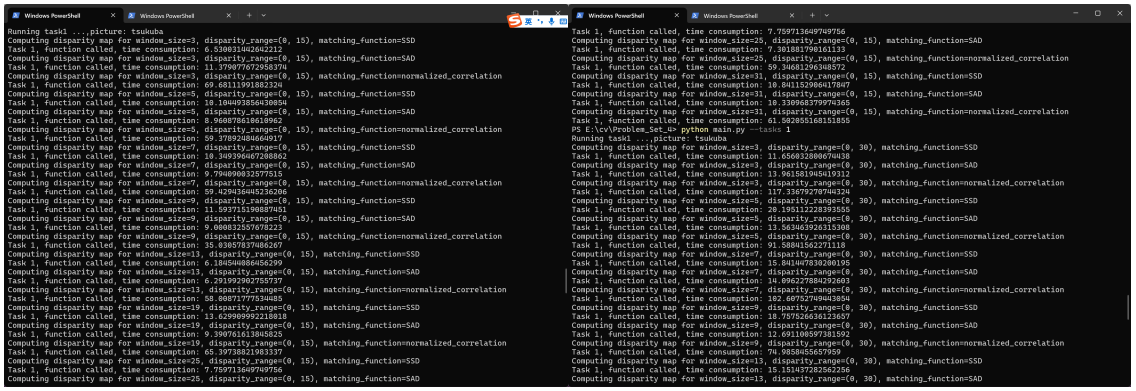
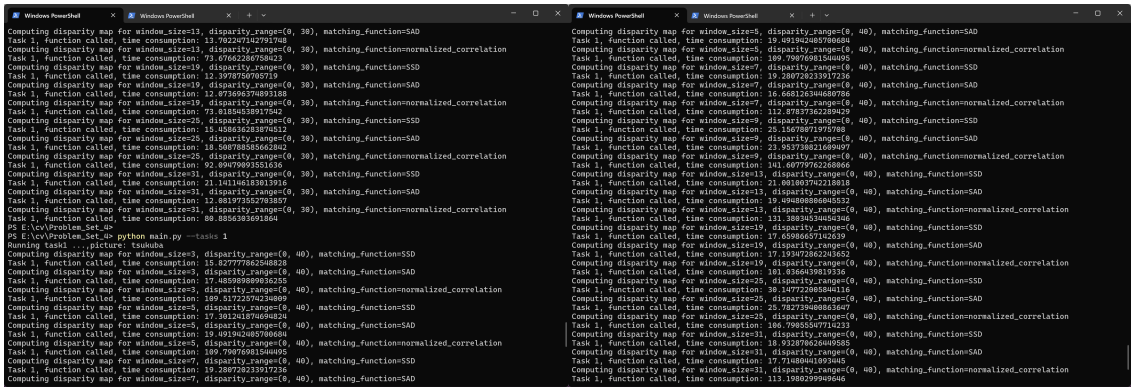
# HW4 Report

## 1 Basic Stereo Matching Algorithm

在output文件夹中，保存了window_size分别为3/5/7/9/13/19/25/31，disparity range在0~10、0~15、0~20、0~30、0~40、0~50的不同情况，记录了运行时的终端（主要输出了运行时间）如下：

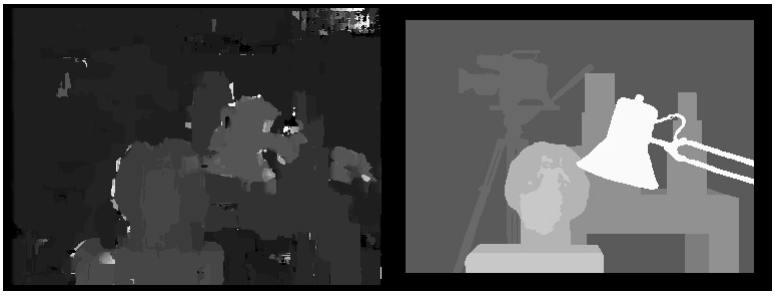




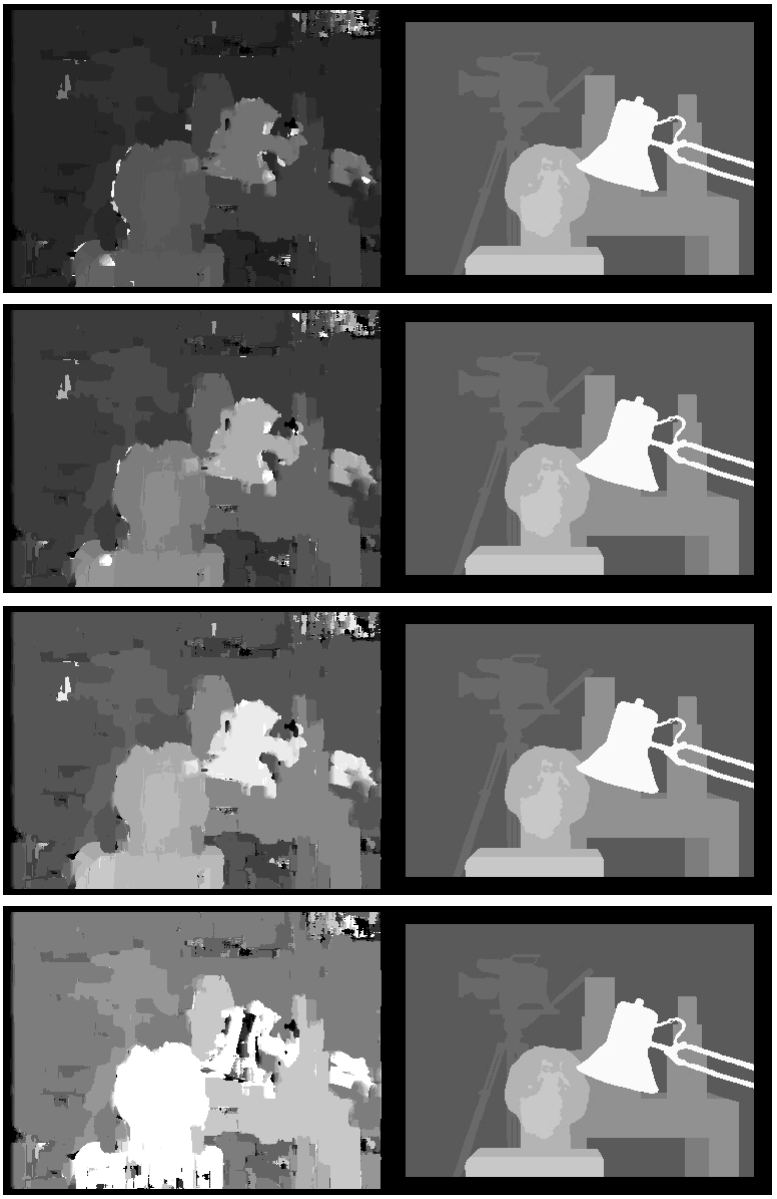
对于要求中的以下问题：

> • How does the **running time** depend on window size, disparity range, and matching function?
>
> • Which **window size works the best** for different matching functions?
>
> • What is the **maximum disparity range** that makes sense for the given stereo pair?
>
> • Which **matching function** may work better for the given stereo pair?

1. 由上述终端的输出，可以看出在运行时间上：总体上disparity range越大耗时越长，两者正相关；
   同样条件下window size越大耗时相对较长，但区别很微小并不明显，在matching function 为normalized_correlation下变化相对较大一点；
   函数选择上，SAD和SSD的运行时间差别不大，并都明显短于normalized_correlation的耗时。
2. 根据output文件夹中的显示，以及我在其他参数下的运行结果，窗口较小大小（如3、5时）噪点过多，过大时（如25、31）过于平滑模糊了很多细节，最终选择**窗口大小为13**的中间值。
3. 从公式上看，disparity = param/depth，所以depth最小即离相机最近的物体（Tsukuba图中的台灯）的视差值就是最大的合理视差；
   而根据测试，视差范围为 0~30 或更高时，台灯的深度值并不明显小于其他物体，甚至大于后景中的许多噪点，并不是理想的合理情况，比如这张图片`task1_tsukuba_13_0_40_normalized_correlation.png`：



可以明显看见，图中台灯后的其他物体上有很多白色的噪点（深度值远小于台灯，并不合理）将视差范围逐步调整（下图依次为 0~30, 0~20, 0~15, 0~10），可以看见效果改善：



同时可以看见，视差0~10时效果也不佳；固定其它参数，在窗口大小和误差函数取其他参数的情况下，发现效果依然类似。因此，根据不同视差范围的效果图，最终选定视差范围**为0~15**，效果较佳。

4. 总体上根据output文件夹中的结果，SAD matching function的表现不佳，SSD 和 normalized_correlation 大体可以正确实现轮廓与深度值的呈现；而具体比对后两者时，发现 normalized_correlation 比 SSD 噪点更明显，且二者的细节呈现差距不是很大，因此最后选择SSD matching function。

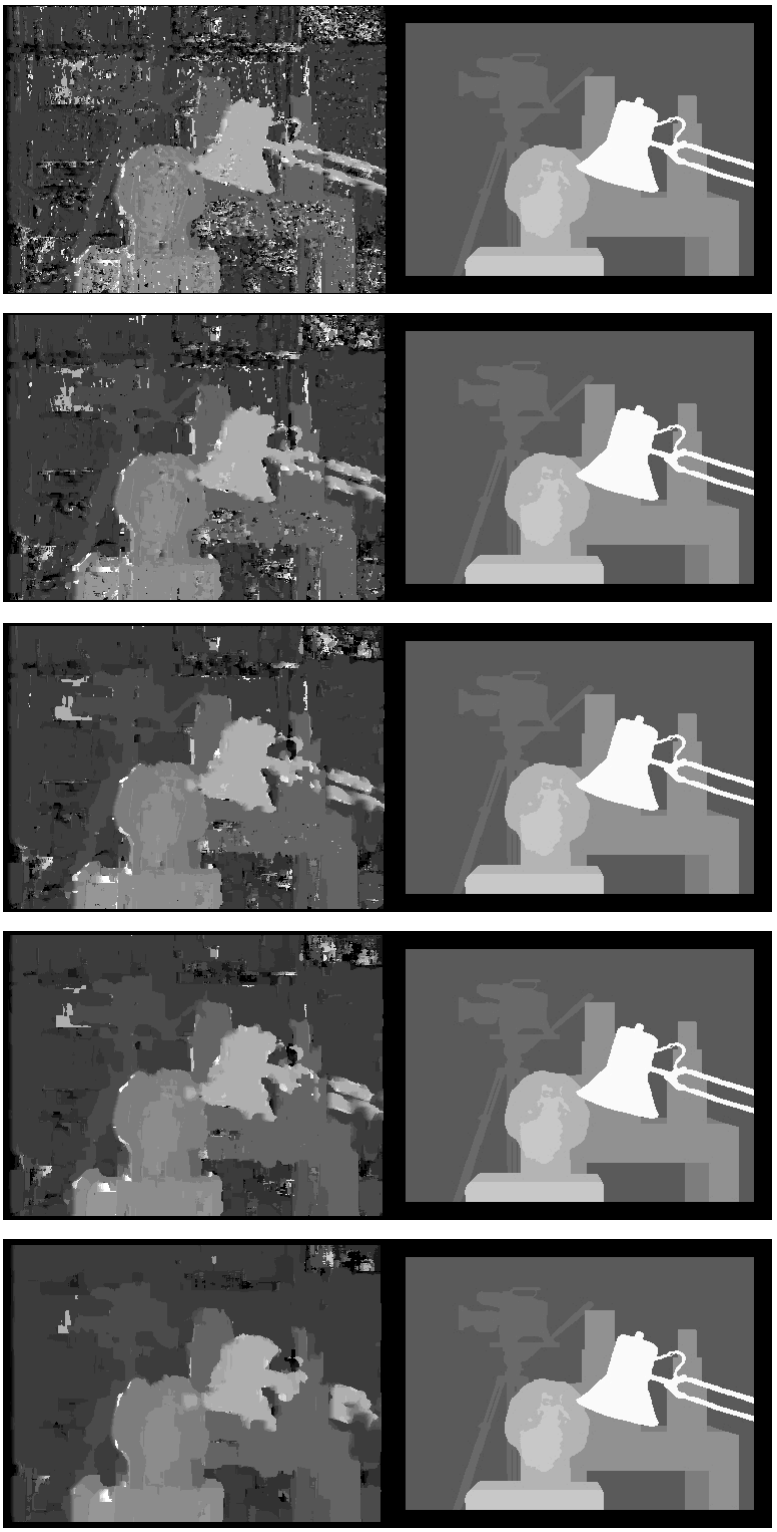
更多讨论：

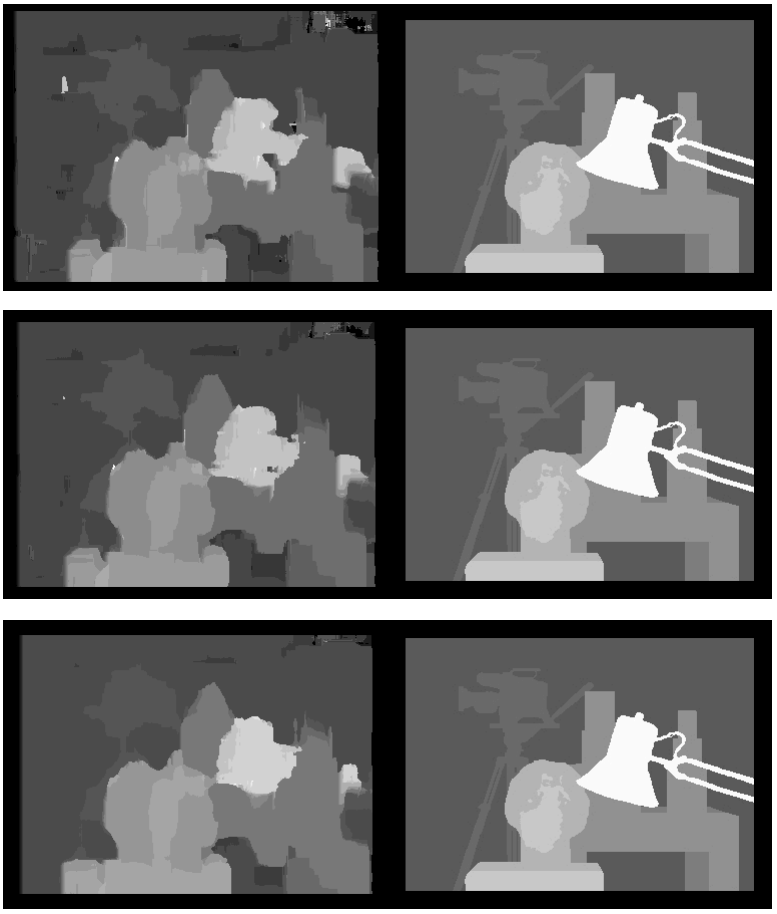
> • Discuss the trade-offs between different hyperparameters on **quality and time.**
>
> • Choose the **best hyperparameters** and show the corresponding disparity map.

> • Compare the best disparity map with the ground truth map, discuss the **differences and limitations** of basic stereo matching

关于trade-offs, 总结多次运行结果和上文的讨论，可概括为：

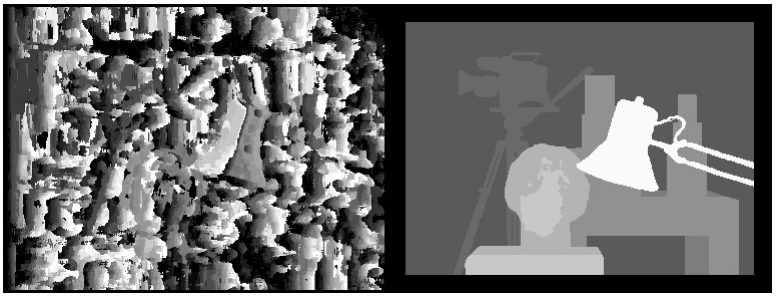
1. 对于窗口大小的选择，具体表现为呈现质量的不同：小窗口保留细节，但噪声大；大窗口较为平滑，但细节丢失
   比如以下几张效果图，依次为在disparity range0~20, SSD function 下 window_size = 3,5,7,9,13,19,25,31的效果：





2. 对于视差范围，范围大时运行时间更长；
3. 不同的函数选择上，SAD和SSD的运行时间差不多并都明显短于normalized_correlation，而效果上SSD的噪声会相对较少一些。

此外，在最开始时SAD的效果极其差，如下task1_tsukuba_7_0_20_SAD.png：



发现这是由于OpenCV 读入图片的数据类型是 uint8如果不做类型转换为浮点数，在计算SAD的时候会发生溢出，导致效果极其糟糕，因此需要在`task1_compute_disparity_map_simple`函数的一开始多一步转换：

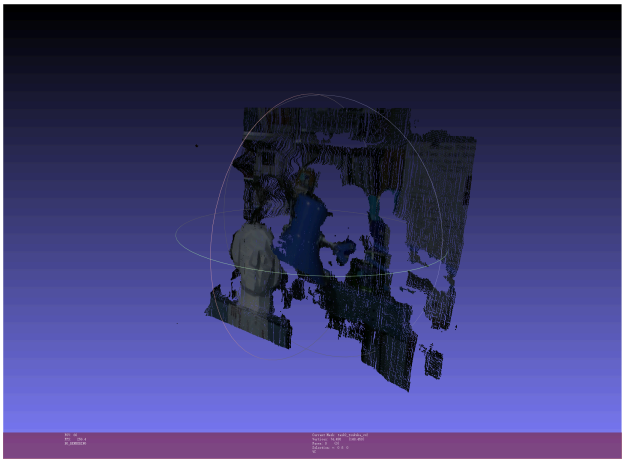
```
ref_img = ref_img.astype(np.float32)
sec_img = sec_img.astype(np.float32)
```

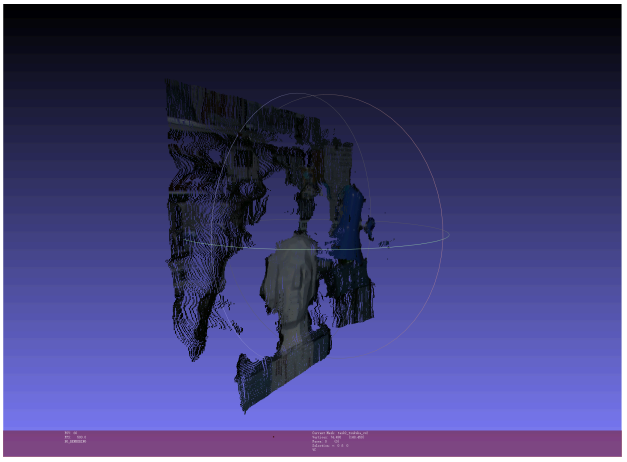
最终，综合细节、噪声等因素，虽然window_size较小时可以保留较多细节但是噪点实在太多，我选择 ***window_size=13, diaparity range 0~15, matching function SSD*** 的参数组合最为最佳，即`task1_tsukuba_13_0_15_SSD`，运行时间**大约在15s左右**：

根据上述图片展示，可以看出尽管选择调参结果中最好的组合，深度计算的结果较标准图偏大一些，整体上依然有更细微处细节不全、鲁棒性低 图像不平滑噪声多的不足 (由于缺乏全局约束，容易在遮挡、低纹理、光照变化下失败)；且基础立体匹配对参数的选择较为敏感，如匹配函数、窗口大小(小窗口保留细节但噪声大、大窗口平滑但细节丢失)、视差范围都会较为明显地影响整体效果

# 2 Depth from Disparity

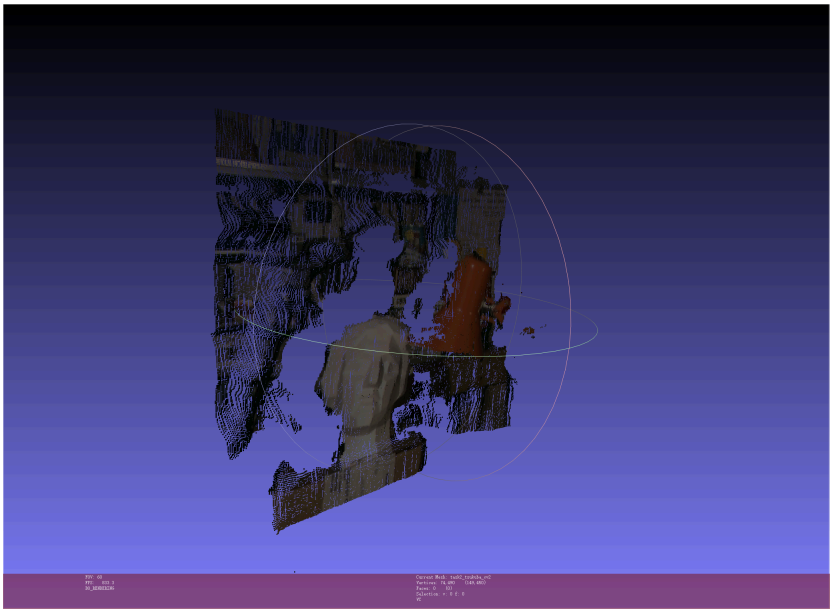
example的效果如图：

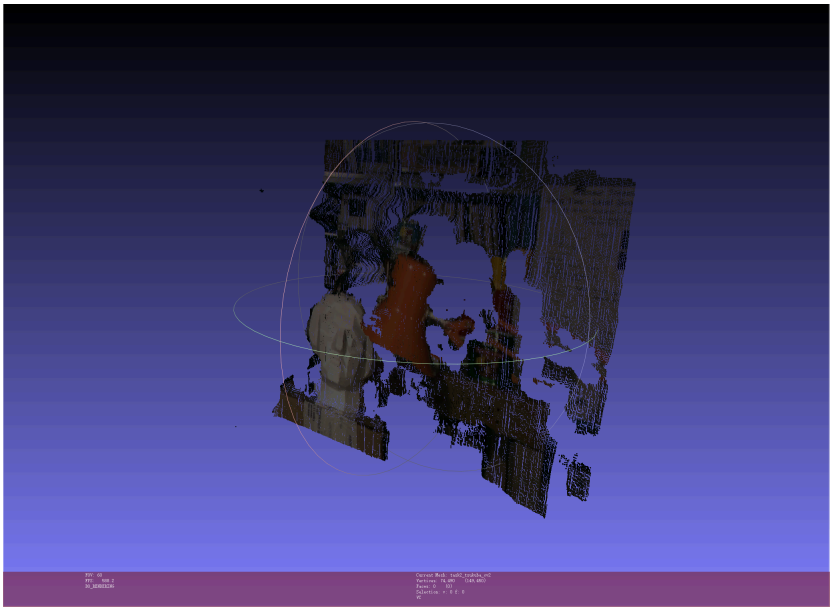




## disparity map computed using cv2.StereoBM:

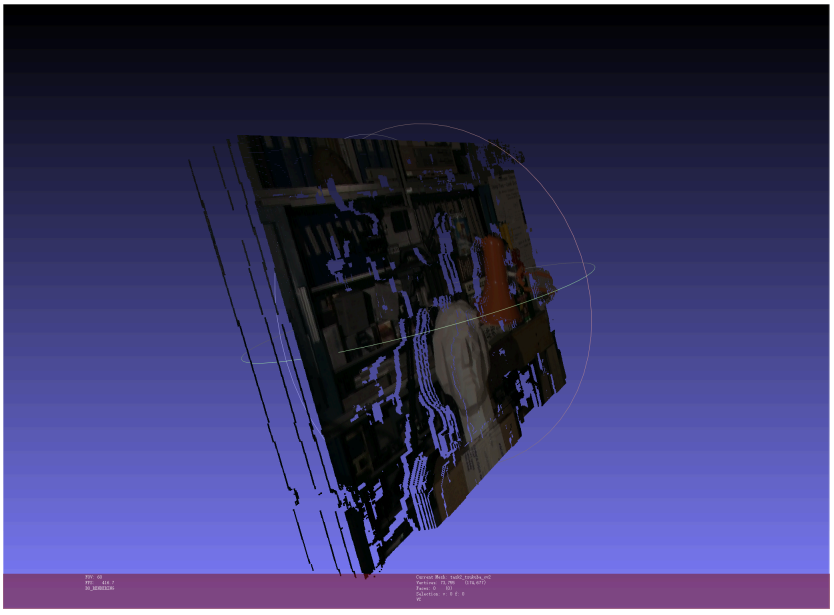
进行参数调整，取baseline = 100，focal_length = 100时最贴近 pointcloud.example的效果。

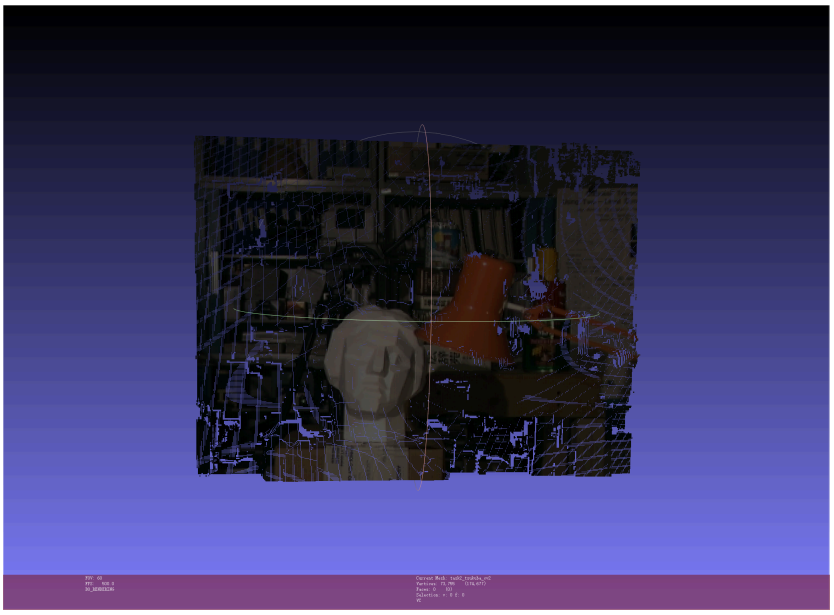


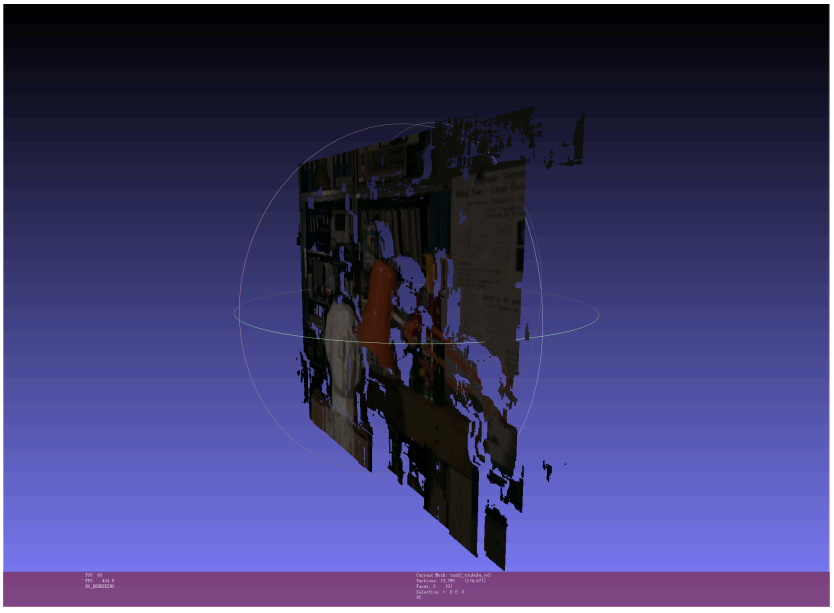


**disparity map computed in task 1:**

进行参数调整，在baseline = 10，focal_length = 15时：

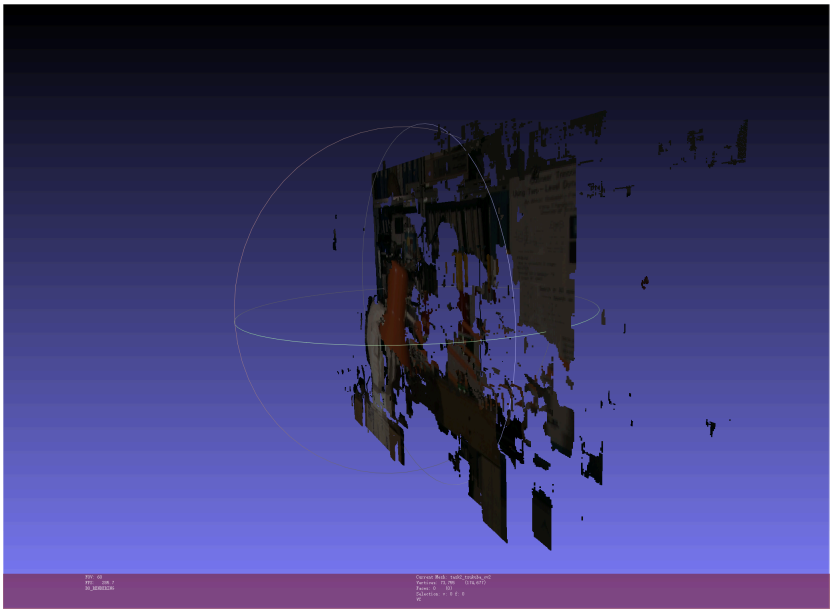


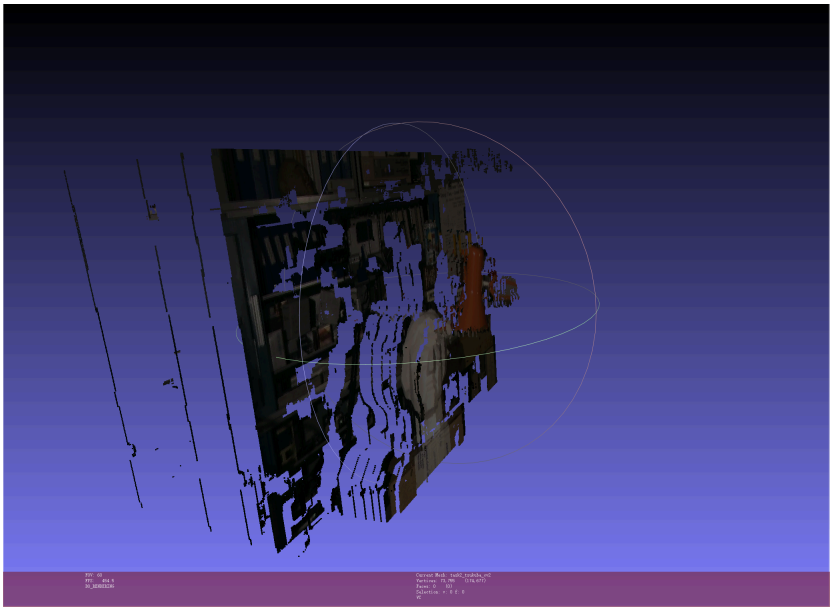




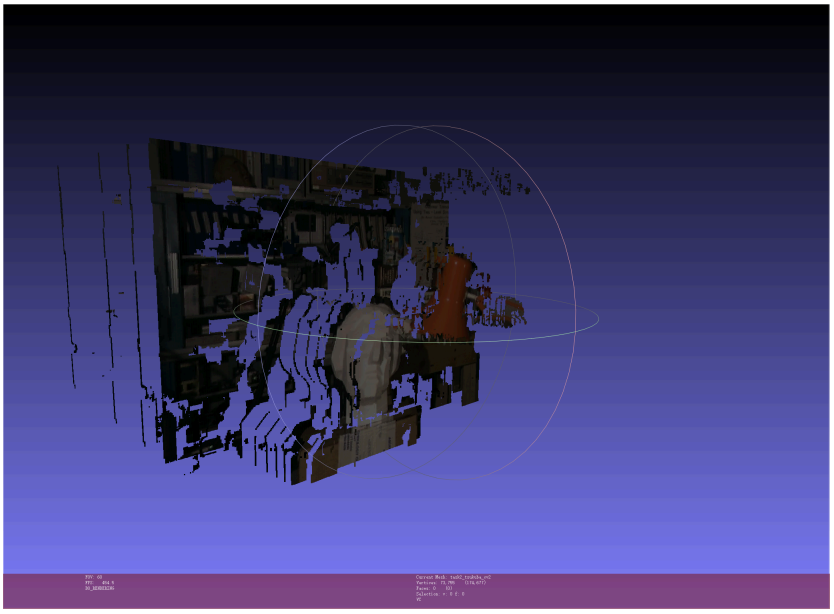
随着baseline * focal_length的增加，由于depth = f*B/disparity，disparity 的微小变化会导致 depth 的变化幅度增大，点云的“分层”拉得更开，呈现出离散的多分层效果：

如在baseline = 10，focal_length = 50时：





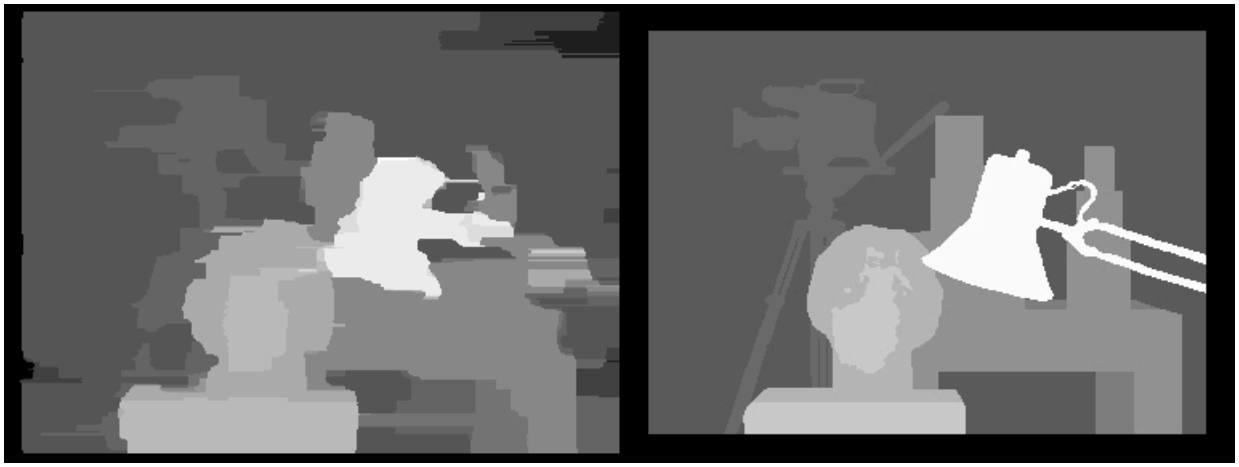
在baseline = 10，focal_length = 120时：



与cv2.StereoBM的情况相比点云图中点的个数较多细节展示较为丰富，且z方向分层较多；而在边缘等局部会出现噪声，且整体上虽然分层明显但不如stereoBM的结果平滑。

# 3 Stereo Matching with Dynamic Programming

按照ppt讲义中的做法，得到效果图:



可以看出，相比于第一问的局部匹配，dp算法能够减少很多噪点，让整张图看起来更加连续，但是由于按行处理，并且过程中设置较大的occlusion惩罚再通过左侧像素值复原，呈现出“横向”一致的感觉，会有较为明显的横行条纹。

运行时间大约在60~100s左右，在baseline = 10，focal_length = 120时得到的点云效果图如下：