

# 计算机视觉 双目视觉 实验报告

林乐天 2300012154

## Basic Stereo Matching Algorithm

### 实现

首先实现单行的基本匹配算法。

对于每一行，计算 $[-win\_half, win\_half]*[-win\_half, win\_half]$ 的窗口和将其移动 d 在另一张图上的窗口差异，该差异用 SSD,SAD 和 normalized\_cross\_correlation 三种方式计算。对于每个点的 disparity，选择以该点为中心最小的差异对应的 d 值作为 disparity。实现的函数为 compute\_disparity\_for\_row。

为了加速计算，我选择对图像的每一行建立一个线程，并行计算 disparity，实现了比较理想的效果。

### Results

#### 1.参数对时间的影响

先看理论的影响。对于每个点，计算 disparity 需要  $win\_half*win\_half$  次计算，对于每个点，需要计算 max\_disparity 次 disparity，所以总的计算量为  $win\_half*win\_half*max\_disparity*height*width$ ，故理论来说，计算量与窗口大小的平方成正比，与最大 disparity 的线性成正比。

实际测试数据如下

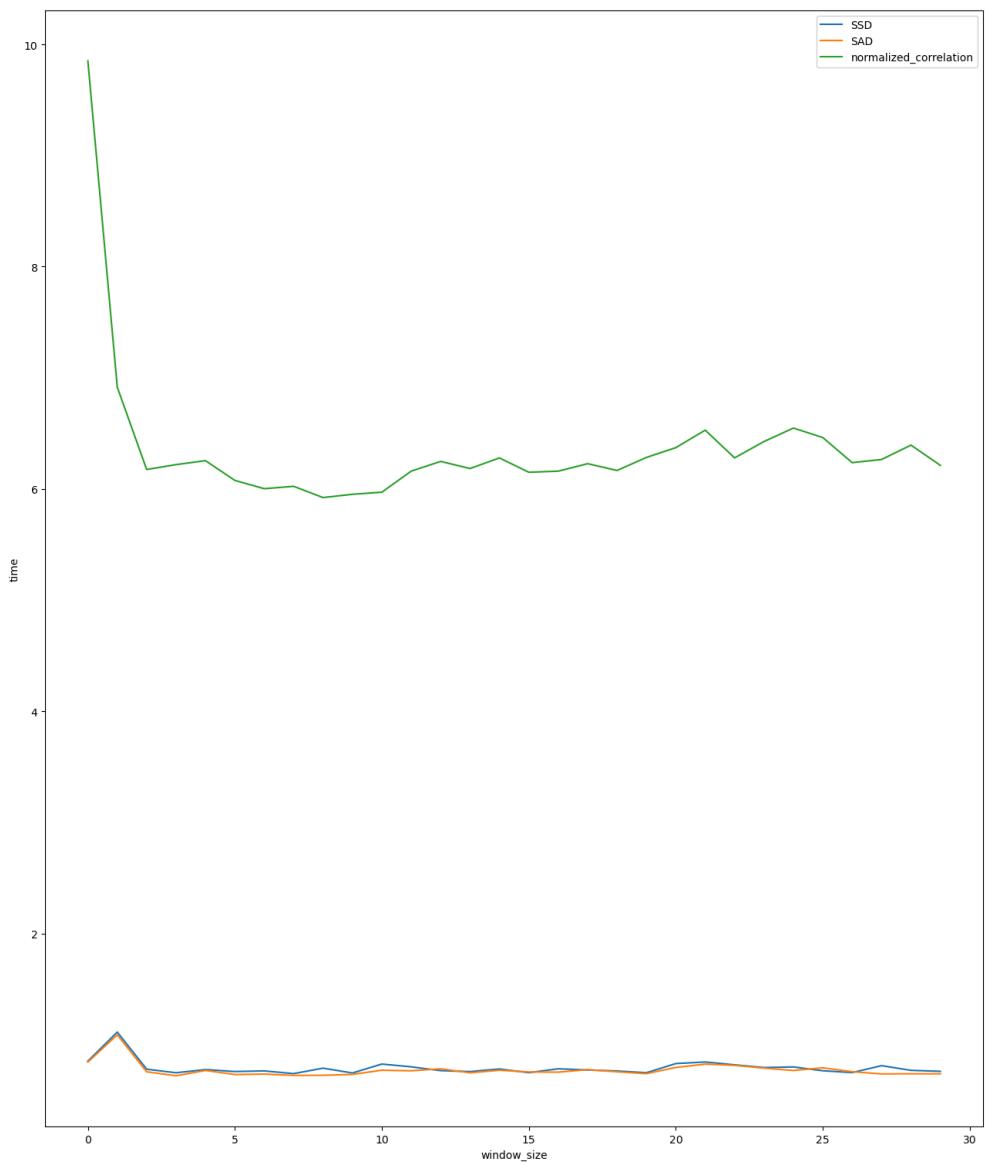


Figure 1: Time vs. window size (disparity=15)

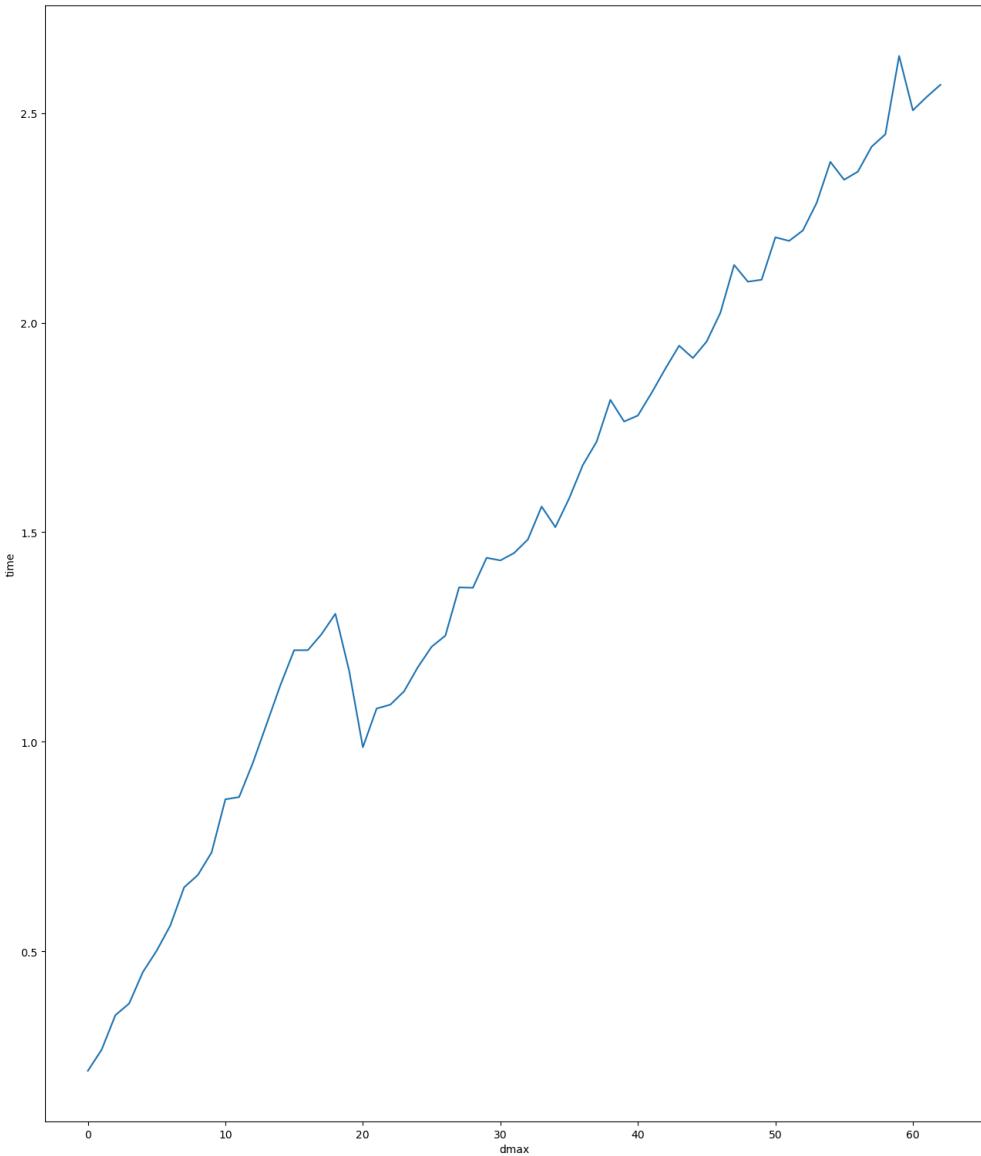


Figure 2: Time vs. max disparity (window size=31)

结果略有出乎意料。对于最大 disparity，时间与 disparity 的关系是近似线性的，而与窗口大小关系不大（很短时间内变大是由于数值不稳定导致出现数据溢出造成的额外时间）。这是因为在实际计算中，有关窗口内部求和的计算是 numpy 的内置函数，而 numpy 的内置函数是高度优化的，故其瓶颈不在窗口大小，而在使用 python 原生实现的 disparity 上。

另一个让人略有惊讶的是，normalized\_cross\_correlation 的时间远远大于其他两种方法，这时因为计算这种代价需要计算均值和方差，而这两个操作是比较耗时的。

## 2.window size 对结果的影响

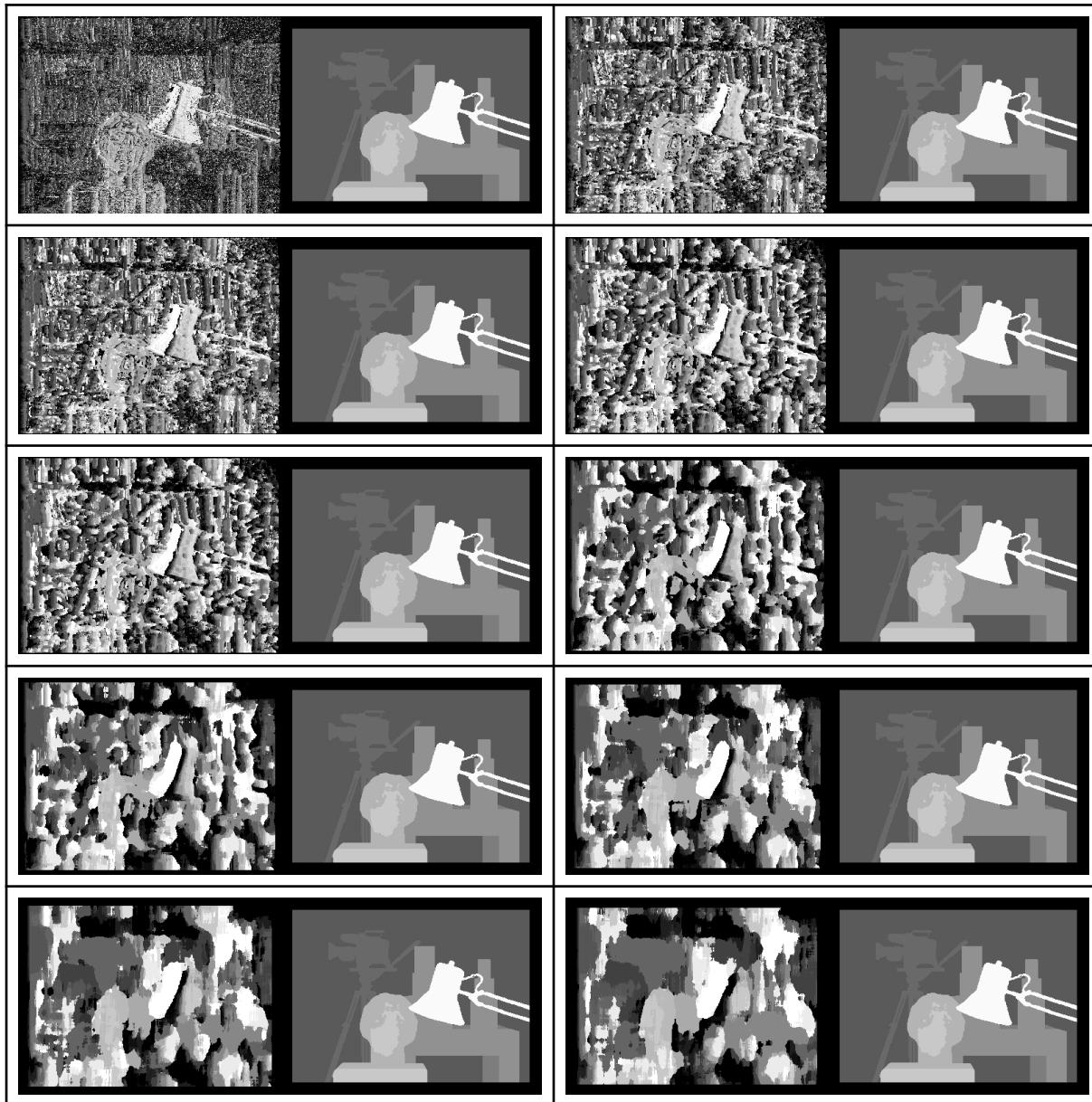


Table 1: Disparity vs. window size,SAD  
winsize=1,2,3,4,5,10,15,20,25,30



Table 2: Disparity vs. window size,SSD

winsize=1,2,3,4,5,10,15,20,25,29

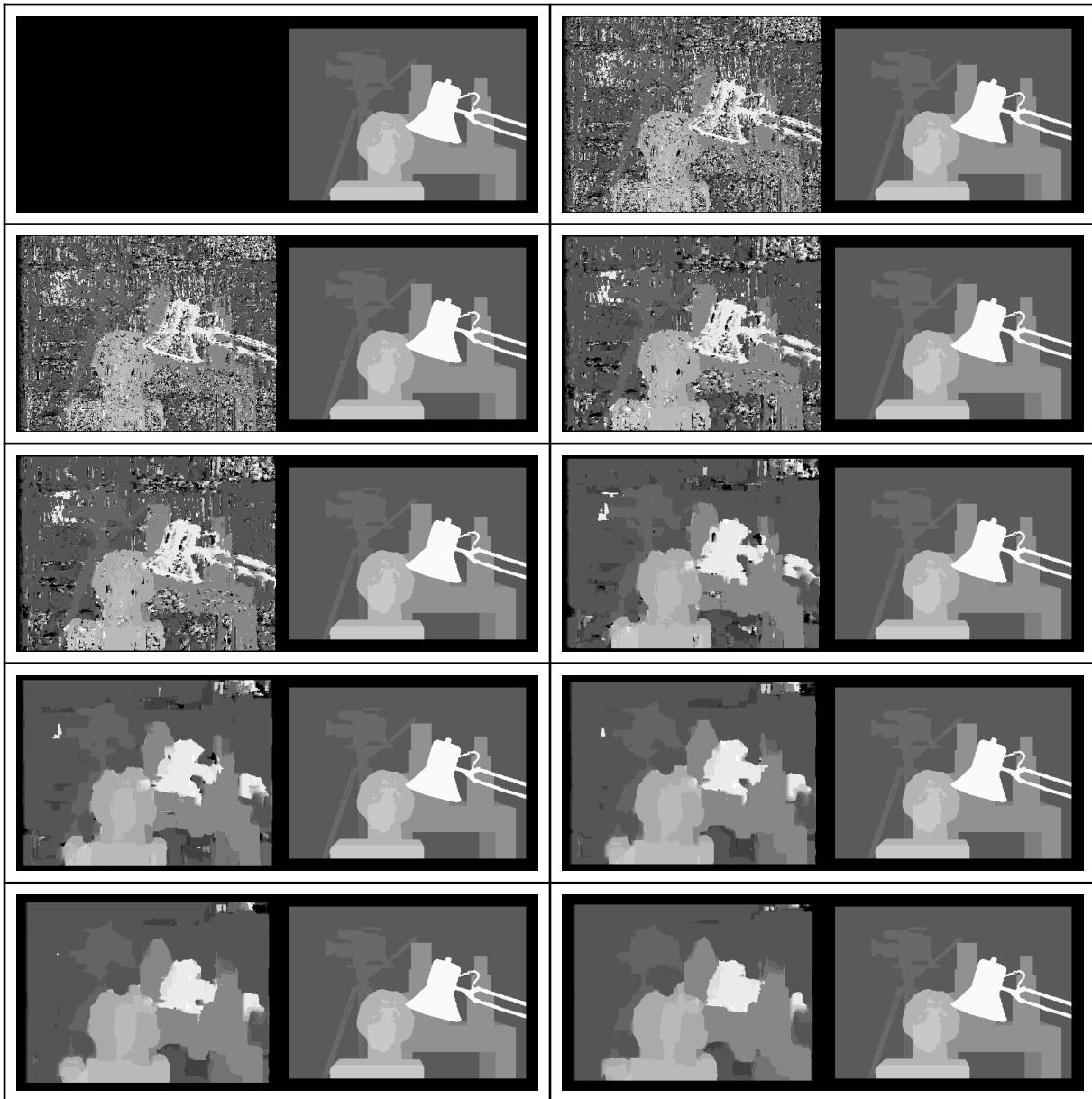


Table 3: Disparity vs. window size, normalized\_correlation  
 winsize=1,2,3,4,5,10,15,20,25,30

结果如上，可以看到，对于三种方法，随着窗口大小的增大，disparity 的结果变得更加平滑，但是，SAD 的图像始终为较为混乱的色块，效果始终不好。同时需要注意的是，其余两个虽然 winsize 增加减少了噪点，但是也减少了细节。在过大的 winsize 下，台灯的边缘和周围环境混淆，这不是一个好的结果。

### 3. max disparity 对结果的影响

为了方便起见，我们选择 winsize=10, SSD,max\_disparity 从 1 到 63 的情况进行测试。

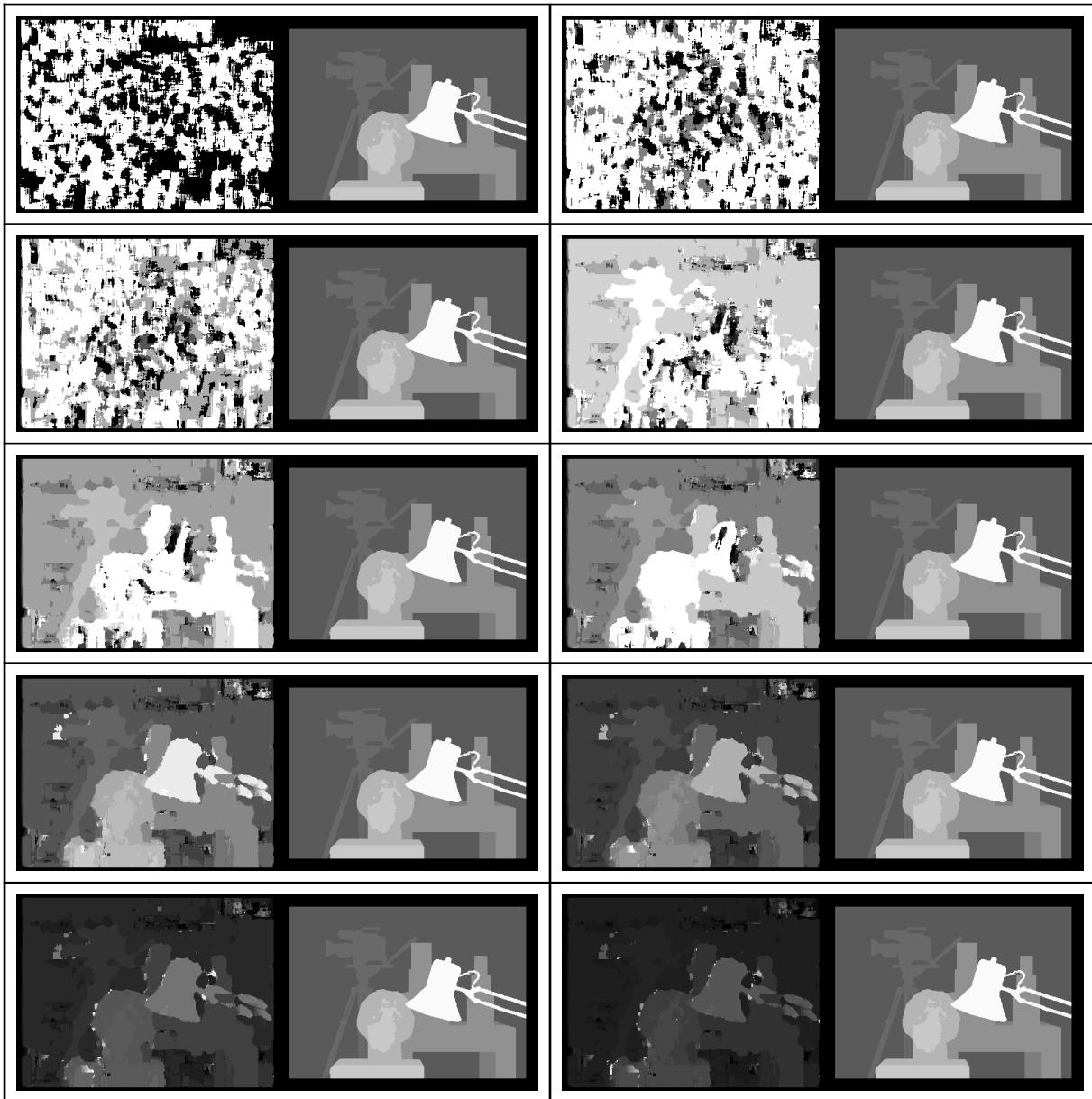


Table 4: Disparity vs. max disparity, SSD  
 $\text{max\_disparity}=1,2,3,6,8,10,15,20,30,40$

后面的图依次变暗是由于一些失配点的 disparity 过高，归一化后导致匹配的点绘制时偏暗。容易看到，当 disparity 增加到一个合适的值之后，其继续增大就没有什么意义了。

#### 4.不同方法的比较

从上面的图片可以很直观的看出，normalized\_correlation 和 SSD 效果差异不大，最差的是 SAD。

#### Disscussion

##### 1. trade-off between hyperparameter and time

对于窗口尺寸来说，过大的窗口会导致细节丢失，过小的窗口会导致噪点增多。但其实际对时间影响不大，这里就选择 15 作为 winsize。

对于 disparity 来说，过大的 disparity 耗时大，且效果提升有限。但是，对于另一张图片，disparity 本来就不小，所以选择 64 作为 max\_disparity。

对于求解方法，选择 SSD 就足够了，速度也快。

## 2. 选取超参计算结果

效果如下：

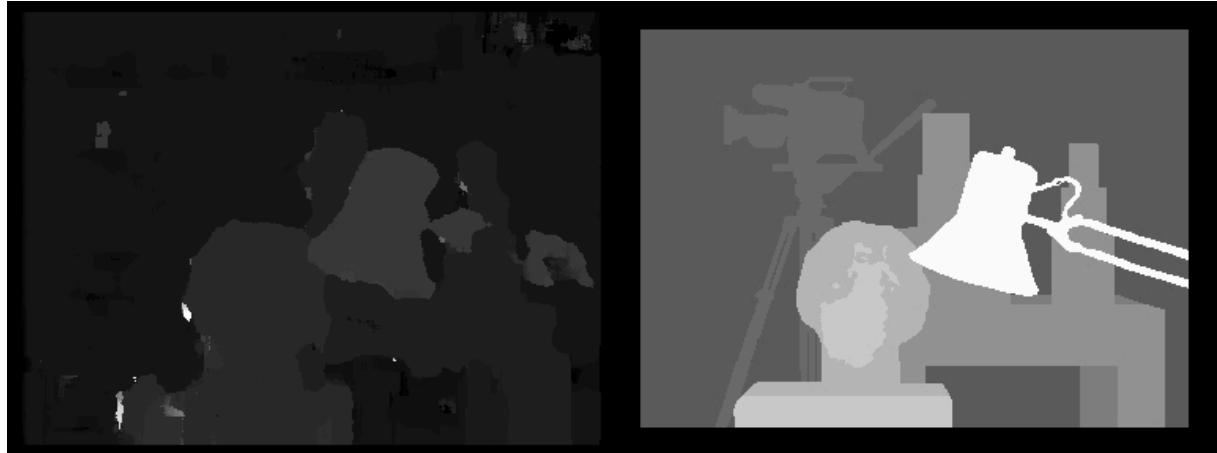
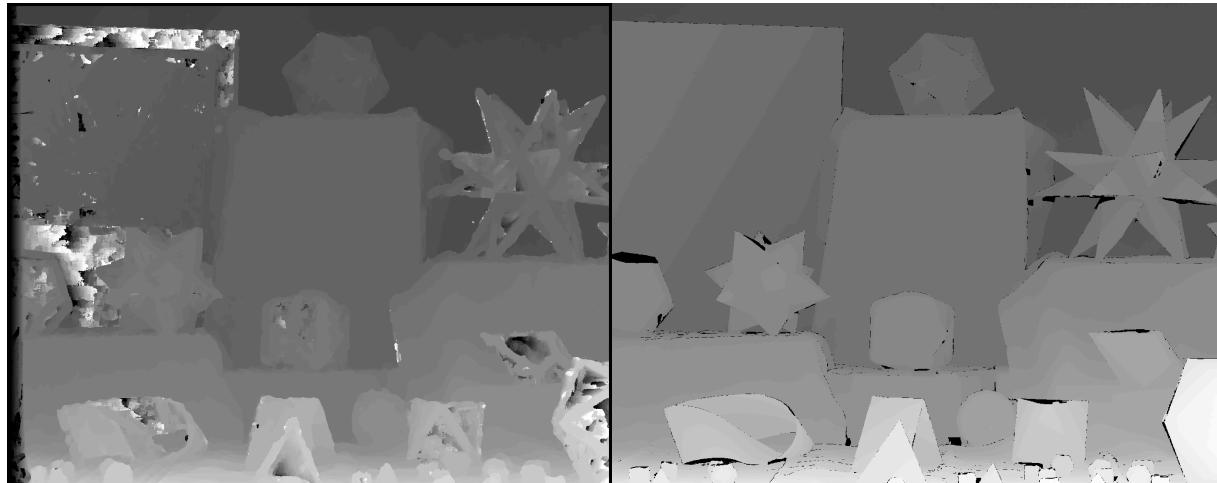


Figure 3: Disparity=64, winsize=15, method=SSD



效果尚可。

## 3. 缺陷

1. 对于失配点效果很差，可能产生了一些奇怪的结果。这些失配点是由于两个视角出现的遮蔽造成的，需要使用合适的方法建模这些遮蔽
2. 有较多的超参数，需要手动调整
3. 很可能出现不光滑的 disparity，需要进一步处理

## Depth from Disparity

实现

首先实现了从 disparity 到 depth 的转换，然后实现了从 depth 到点云的转换。对于 disparity 到 depth 的转换，使用了公式  $\text{depth} = f * b / \text{disparity}$ ，其中  $f$  为焦距， $b$  为基线。如此便得到了点云，将距离小于  $\mu - \sigma$  的点去掉，得到了点云。

## Results

为了进一步优化效果，使用二次插值方法，其公式为（来自 ChatGPT）

$$d = d_0 + \frac{C(d_0 - 1) - C(d_0 + 1)}{2(C(d_0 - 1) - 2C(d_0) + C(d_0 + 1))}$$

其中  $d_0$  为  $\text{argmin } C(d)$

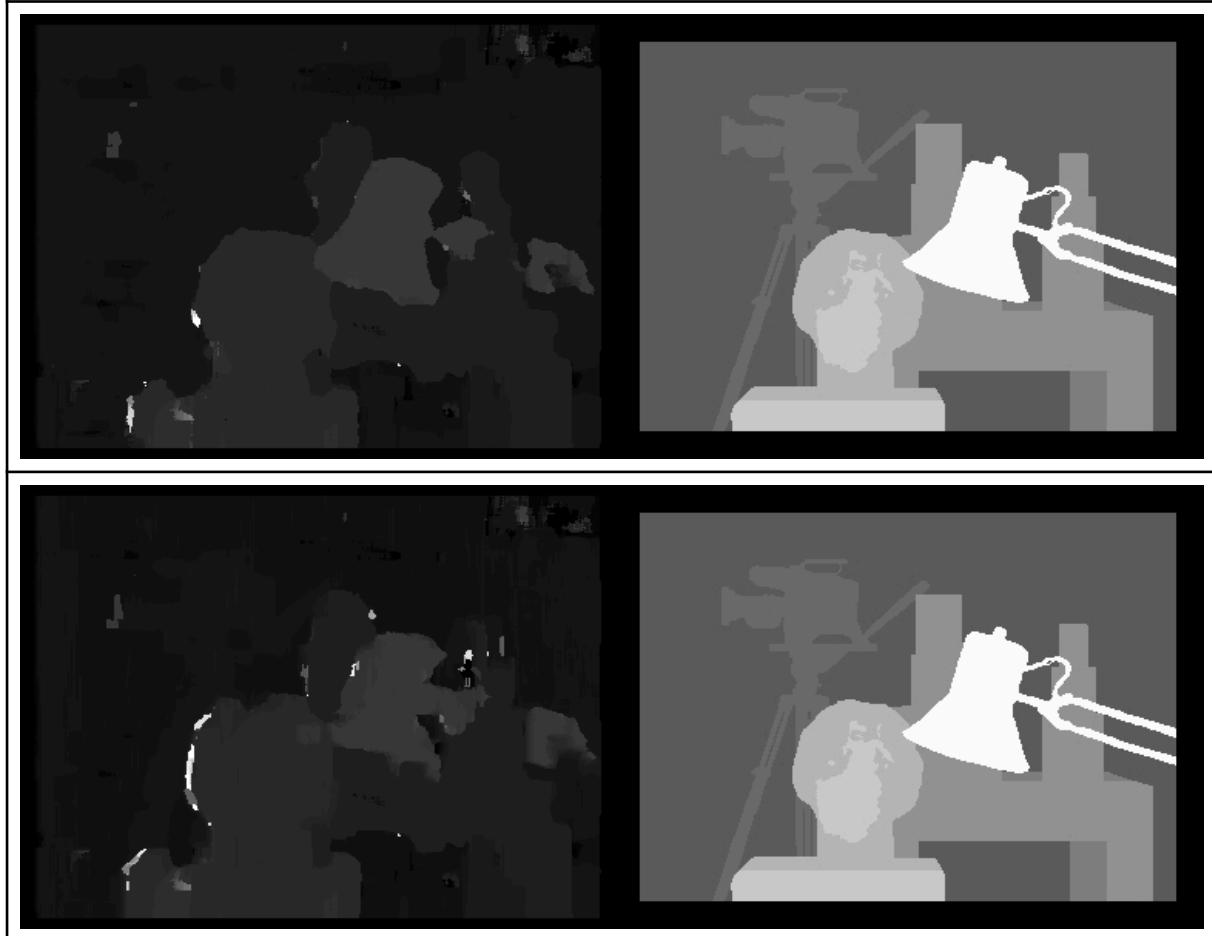


Table 5: 插值前后对比

仔细观察可以看到，边缘点有了更加光滑的边缘，且有了更多的细节。

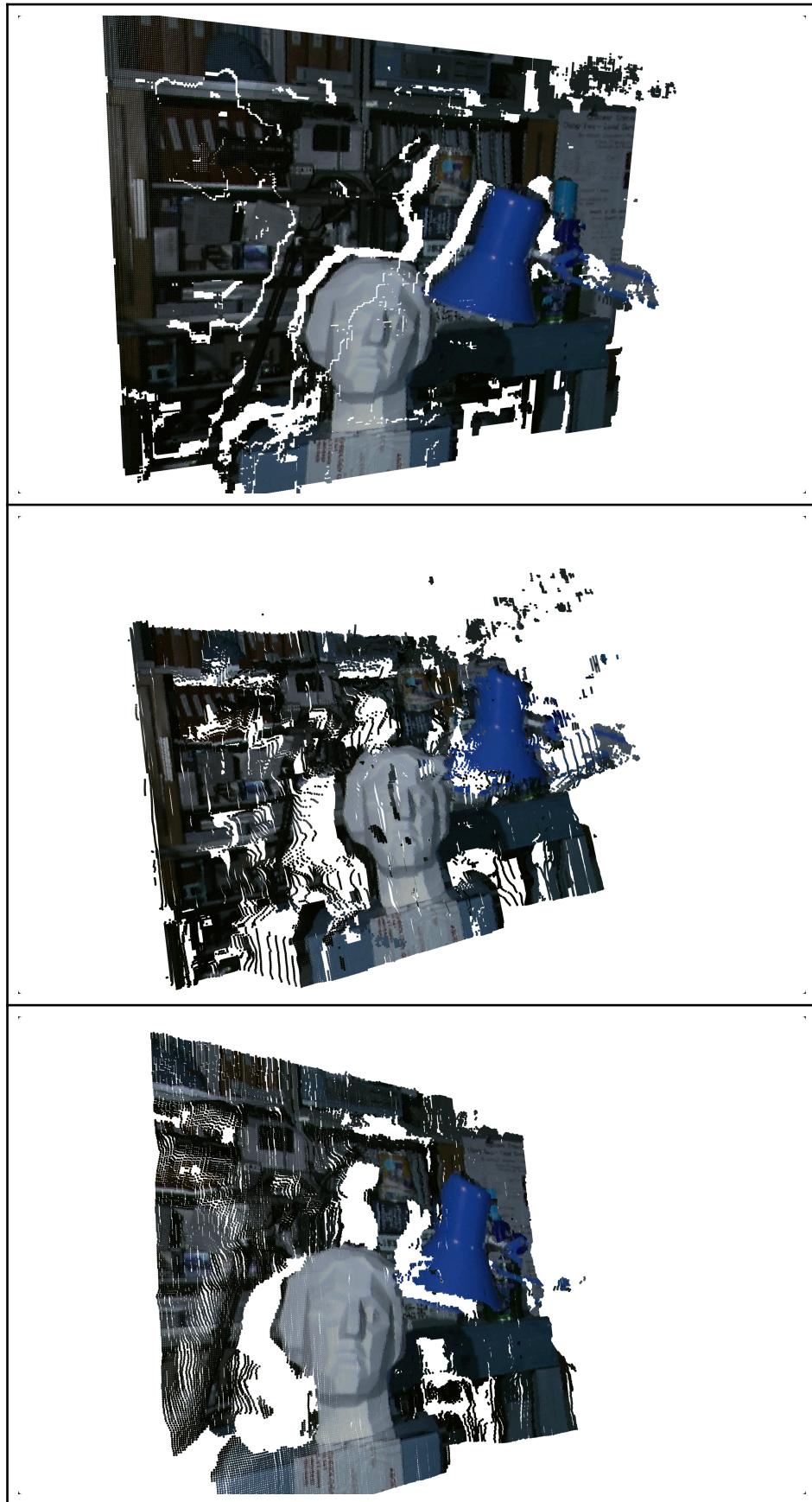


Table 6: 插值前、后、opencv 实现对比

从 3D 图像更加直观。可以看到，插值后的效果更加平滑，细节更加丰富，然而相较于 opencv 的实现，还是有一定的差距，主要体现在平面的一致性上，仍然存在一些噪点。

## Stereo Matching with Dynamic Programming

参考论文:I. J. Cox, S. L. Hingorani, S. B. Rao, and B. M. Maggs, “A maximum likelihood stereo algorithm,” Comput. Vision Image Understanding, vol. 63, no. 3, pp. 542–567, May 1996, doi: 10.1006/cviu.1996.0040.

实现了论文中的算法，首先计算了 cost volume，然后使用了动态规划的方法计算了 disparity。

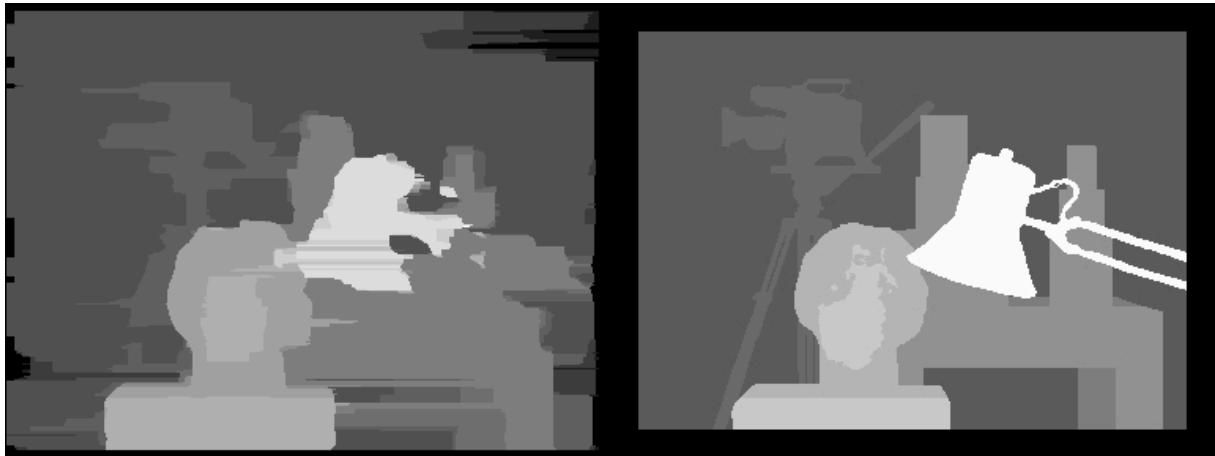


Figure 5: DP 结果

运行时间（通过 tdqm 测得）为 41s,主要是没做并行计算。所以比较慢，也没什么比较的价值。

点云重建结果如下：

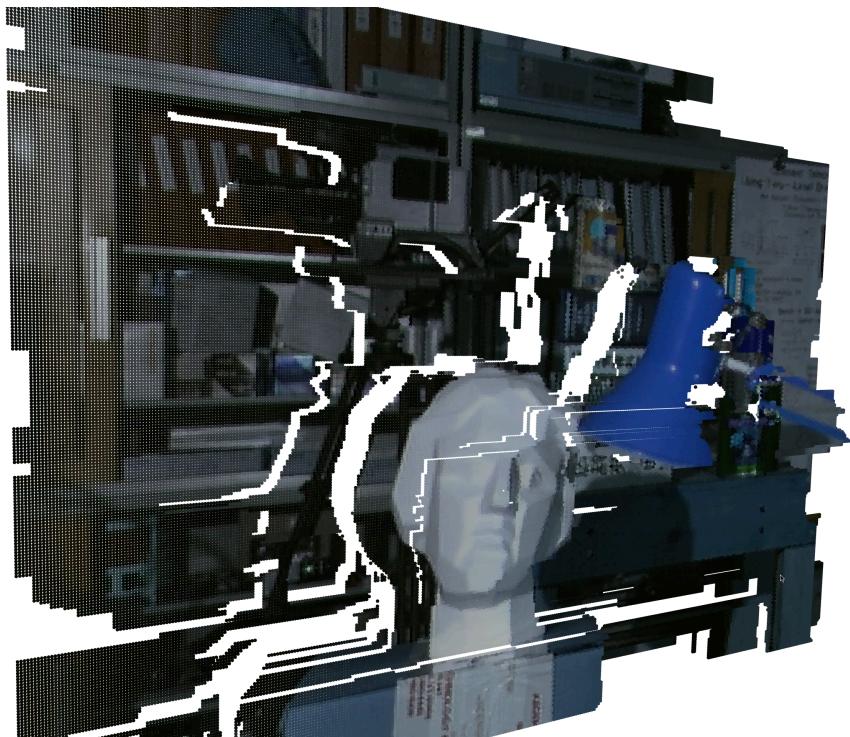


Figure 6: 点云重建结果

虽然点云重建的时候出现了一些被错误扩展的区域，但是整体一致性不错，边界也较为清晰，相较于没有做任何优化的 SSD 来说，有了很大的提升。