数学实验第四次实验报告 multi-view 3D reconstruction

PB20000264 韩昊羽

一.实验要求

利用网站上的数据集或者框架中给定的测试样例，恢复三维点云模型，要求：

- 利用数据集中的相机内参，框架中只需要通过EXIF读取相机的焦距，并直接计算内参矩阵即可。
- 利用课件中的方法计算每两个相机之间的变换矩阵，来恢复每个视角的点云，并将点云融合在一起。
- 需要计算两张之间的相机变换和稠密点云恢复
  在框架中，只需要我们补充求基础矩阵Fundamental matrix、求相机内参矩阵K、求一张图片相对于另一张图片的$[R|t]$,求最后的空间坐标。并可以使用框架中自带的测试样例进行测试。

环境配置

- matlab R2021a

二.实验原理

1.坐标系变换

在整个三维重建的过程中存在三个坐标系，一个是世界坐标系，标志着点在世界中的绝对坐标，一个是相机坐标系，从相机出发在一个视角锥中看到的物体，最后一个是显示屏，也就是画布坐标系，是将相机坐标系中的物体投影到平面画布上得到的拍摄出来的图片的坐标系。
首先，对于相机坐标系变换到画布坐标系，只需要使用相机的内参，我们有二者之间的变换公式：

$$u = f_x \frac{x_C}{z_C} + o_x$$
$$v = f_y \frac{y_C}{z_C} + o_y$$

事实上，可以将这个变换公式写成矩阵形式：

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \widetilde{u} \\ \widetilde{v} \\ \widetilde{w} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ z_C \\ 1 \end{bmatrix}$$

我们记

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$

为相机的内参矩阵，在框架中，我们不考虑矩阵的伸缩，并且认为原点就在中心，所以这时内参矩阵简化为

$$K = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

其中f为相机的焦距。
接下来我们考虑怎么从世界坐标系变换到相机坐标系，由于世界坐标系和相机坐标系都是三维直角坐标系，所以事实上二者之间只相差一个旋转和一个平移，即$[R|t]$矩阵。最后，我们得到总的坐标变换公式为：

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$x = K[R|t]X$$

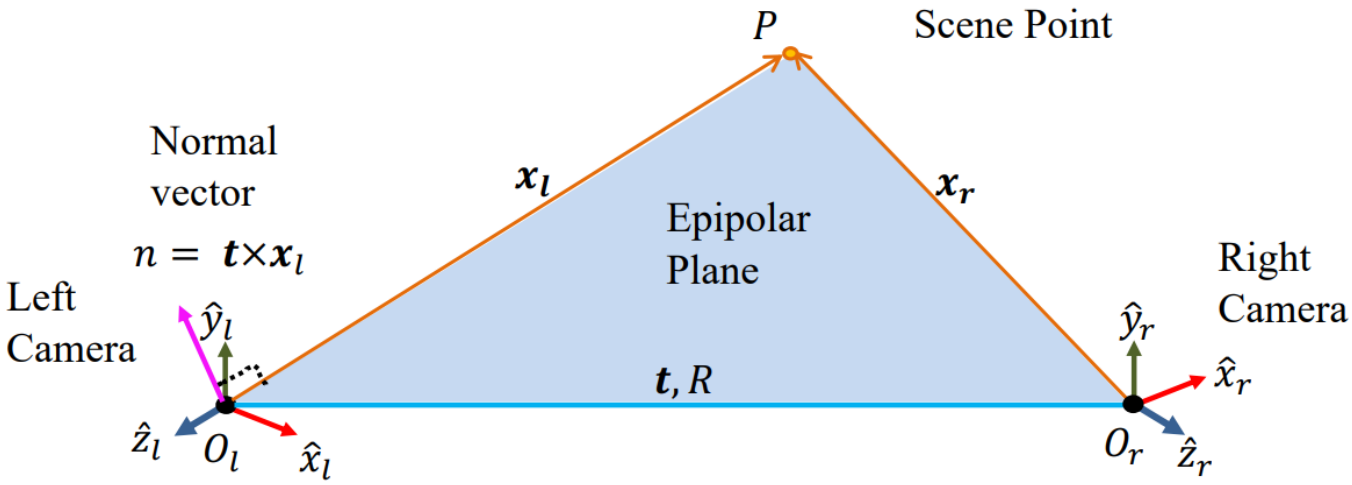
$$P = K[R|t]$$

$$x = PX$$

2.特征点匹配

通过封装好的SIFT方法实现

3.基础矩阵Fundamental matrix的计算



对于相机平面，我们有

$$x_l \cdot (t \times x_l) = 0$$

一系列变形后我们可以得到

$$x_l^T E x_r = 0$$

若我们再把x_r, x_l变换到画布坐标系，可以得到

$$\begin{bmatrix} u_l & v_l & 1 \end{bmatrix} F \begin{bmatrix} u_r \\ v_r \\ 1 \end{bmatrix} = 0$$

其中$E = K_l^T F K_r$

4.相邻图片之间[R|t]的计算

对于给定的essential matrix E，我们可以用SVD分解来得到[R|t]

$$E = U diag(1, 1, 0) V^T$$

$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}\mathbf{V}^T \middle| +\mathbf{u}_3 \right]$$

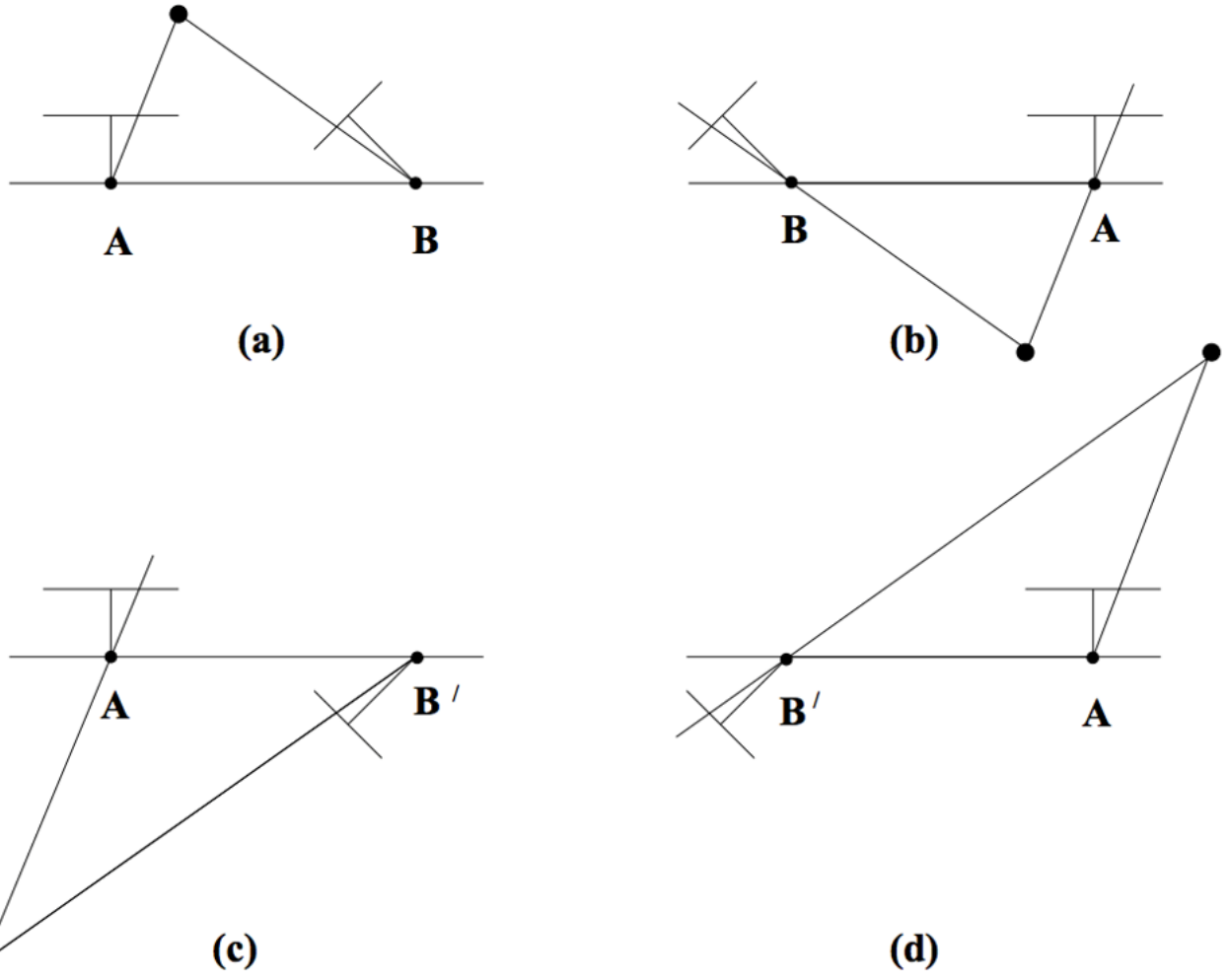
$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}\mathbf{V}^T \middle| -\mathbf{u}_3 \right]$$

$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}^T\mathbf{V}^T \middle| +\mathbf{u}_3 \right]$$

$$\mathbf{P}_2 = \left[\mathbf{U}\mathbf{W}^T\mathbf{V}^T \middle| -\mathbf{u}_3 \right]$$

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

所以我们会得到四种可能的[R|t]，需要判别出我们希望得到的是哪个



判别式为

$$(X - C) \cdot R(3, :)^T > 0$$

其中$(X-C)$为空间中点到相机原点位移，$R(3,:)^T$事实上提供了照相机看的方向，二者做点乘事实上在判断物体是在我们的正面被看到了，而不是背面。$(X-C)\cdot R(3,:)^T>0$提供了B的判断标准，X在里面的z坐标，事实上是衡量是不是沿z轴正方向，衡量了A是否正确。根据这个，我们就能选出正确的[R|t]。

5.世界坐标的计算

事实上我们现在已经得到了每个矩阵的[R|t]和之间的[R|t]。现在做的是全都移到一个大场景中，首先我们要作merge和bundle adjustment，然后就可以计算相机位置，计算真正的点坐标了。
相机原点计算：由于变换过后相机原点的坐标是0，故

$$RC+t=\vec{0}$$
$$C=-R^Tt$$

三.编程实现

框架已经实现了大部分，我们需要补充求基础矩阵Fundamental matrix、求相机内参矩阵K、求一张图片相对于另一张图片的$[R|t]$,求最后的空间坐标等。并使用网上的数据集测试结果。

文件结构

未对框架做出改变

1.内参矩阵K计算

```
frames.K = f2K(frames.focal_length);
```

事实上助教已经给了公式，又发现在文件中已经写好了f2K函数，只需要传入焦距直接调用即可。

2.estimateF函数补全及E的计算

```
[F, inliers] = ransacfitfundmatrix(pair.matches(1:2,:), pair.matches(3:4,:), t,0);

pair.E = frames.K'*pair.F*frames.K;
```

助教已经给出了详细的注释，只需要看懂ransacfitfundmatrix函数是传入两组点对和参数t，方法会自动计算基础矩阵F。这之中最后一个参数默认取0，可以不用管。
然后根据F和上面的公式$E=K_l^TFK_r$直接计算E即可。

3.寻找中心图片

```
P{1} = frames.K*[eye(3),zeros(3,1)];

for i=1:4 % ToDo:计算四种可能，你需要读懂vgg_X_from_xP_nonlin()函数（在相应的.m文件下），PS:如果你够仆
    clear X;
    % first:先得到的是相对1的K[R2|t2]
    P{2} = frames.K*Rt(:,:,i);
    % second:应用vgg_X_from_xP_nonlin()计算
    for j = 1 : size(pair.matches,2)
         x = [pair.matches(1,j) pair.matches(3,j);pair.matches(2,j) pair.matches(4,j)];
         X(:,j) = vgg_X_from_xP_nonlin(x,P,repmat([frames.imsize(2);frames.imsize(1)],1,length(P
    end
    %注意事实上现在第三行还是带着Z的，要把Z消去
    X = X(1:3,:) ./ X([4 4 4],:);
    dprd = Rt(3,1:3,i) * ((X(:,:) - repmat(Rt(1:3,4,i),1,size(X,2))));
    goodCnt(i) = sum(X(3,:)>0 & dprd > 0);

end
```

这一块感觉是最难的部分了，首先我们需要读懂 `vgg_X_from_xP_nonlin()` 函数，就是给定两组pairs和之间的变换矩阵和图像的大小，来计算点在三维空间中的近似值。这里由于只有两个相机，所以把第一个相机的$[R|t]$取成$[I|0]$,第二个$[R|t]$依次取四个可能的$[R|t]$中的一个。特别注意，这里存的X不是标准化了的，即此时每个点的第四个分量并不是1，需要同除来校正。最后的判断就像上面所说，两个不等式来衡量相机位置对不对。

4.世界坐标计算

```
X = X(1:3,:) ./ X([4 4 4],:);
x1= P{1} * [X; ones(1,size(X,2))];
x2= P{2} * [X; ones(1,size(X,2))];
x1 = x1(1:2,:) ./ x1([3 3],:);
x2 = x2(1:2,:) ./ x2([3 3],:);
Graph{frame}.denseX = X;
% ToDo:给出稠密重建的error，所以你只需要读懂denseMatch()函数，直接在这里补充好sum()函数里面的差值是

Graph{frame}.denseRepError = sum(([x1; x2] - Graph{frame}.denseMatch(1:4,:)).^2,1);
%待会再回来看

Rt1 = mergedGraph.Mot(:, :, frame);
Rt2 = mergedGraph.Mot(:, :, frame+1);
% ToDo: 计算得到相机中心，分别用上面的Rt1和Rt2;
C1 = - Rt1(1:3, 1:3).' * Rt1(:, 4);
C2 = - Rt2(1:3, 1:3).' * Rt2(:, 4);
% ToDo: 仿照给的view_dirs_1，补充2，并读懂这里是如何计算两个向量的！！！！！
view_dirs_1 = bsxfun(@minus, X, C1);
view_dirs_2 = bsxfun(@minus, X, C2);
view_dirs_1 = bsxfun(@times, view_dirs_1, 1 ./ sqrt(sum(view_dirs_1 .* view_dirs_1))); %看起
view_dirs_2 = bsxfun(@times, view_dirs_2, 1 ./ sqrt(sum(view_dirs_2 .* view_dirs_2)));
% ToDo: 两个向量求出相机的拍摄角度cos值
Graph{frame}.cos_angles = dot(view_dirs_1,view_dirs_2);
```

事实上这里已经给出了比较完整的框架，对称的算出世界坐标系中的点，和对应到两个图片中的像素坐标，同时计算error和相机原点的位置。这里注意error事实上是三维重建的误差，后面我们会用这个误差来衡量哪些点是好点，哪些点是坏点。这里采用的是将计算的坐标和原始坐标做差平方求和来衡量到底偏离了多少。相机原点的计算直接参考$C = -R^T t$公式即可。

四.遇到的问题

1.焦距读入问题

发现很多图片都不能从参数中读取焦距，特别有可能会出现不是jpg文件。不过我们可以人为指定一个焦距，而且bundle adjustment也会修改。

2.参数理解问题

对于ransacfitfundmatrix里面最后一个参数我一直没搞懂，因为在estimateE里面feedback传了1，而默认的是传0，和助教同学讨论过之后知道其实只是个标记变量，直接不写取默认值0即可。

3.方法阅读问题

对于RtFromE里面的vgg_X_from_xP_nonlin()函数一开始并没有读懂，也不知道为什么要把第一个$[R|t]$取成$[I|0]$，后来看到后面求世界坐标系里面坐标的时候又调用了这个函数，即

```
X(:,j) = vgg_X_from_xP_nonlin(reshape(Graph{frame}.denseMatch(1:4,j),2,2),P,repmat([frames.imsi
```

这才理解了是给定两张图中间对应点的坐标和$[R|t]$矩阵恢复出世界坐标系中点的信息，也理解了为什么第一个要设成$[I|0]$，因为这时候我们只需要判断哪个$[R|t]$是正确的，所以把坐标系直接取在第一个相机的坐标系，这样后面比较的时候也可以直接取z坐标来衡量了。
另外，一开始有一些误差，但我没找到问题，后面发现似乎需要对X作一个归一化处理，然后取前三个坐标才是准确的。

4.坐标系显示问题

这个问题大概全怪我自己在调matlab的时候不知道修改了什么设定，最后得到的点云坐标是个细长的盒子，后来才发现可以人为修改坐标系的值，可以采用

```
axis equal
```

来指定坐标轴的大小，并且发现在显示点云的时候，有可能会出现一些比较远的点，这样在显示的时候会让真正的点云缩成一团，我一度以为没有跑出结果，这个问题可以通过衡量这个点在两张图之间的夹角来解决，即限制

```
Graph{frame}.cos_angles < cos(5 / 180 * pi)
```

六.拓展思考

关于点云去噪的思考

我发现对于最后得到的点云，其实噪声很大，主要是有很多边缘附近的点事实上有一些小偏移，这些小偏移聚在一起就变成了表面参差不齐。而且会有一些单点游离在整个图像外面。由于不是网格形式，所以不能用相邻点插值（laplace坐标插值去噪）来实现，最终我觉得试一试半径滤波，即扫描每个点附近点，剔除掉那些游离在大部队之外的点。

```
for i = 1 : size(X,2)
       u = 0;
       for j = 1 : size(X,2)
          if(sum((X(:,i)-X(:,j)).^2))<R^2
               u = u +1;
          end
       end
       if(u<10)
           X(:,i) =  [];
       end
   end
```

实际并没有成功，因为点的数量实在是太庞大了，如果每两对点之间都会做一次检测的话直接爆炸，需要跑很长时间，所以这个计划暂时搁浅，不过或许后面可以通过对点云分组来加快速度实现，或者使用KDtree。

关于参数的思考

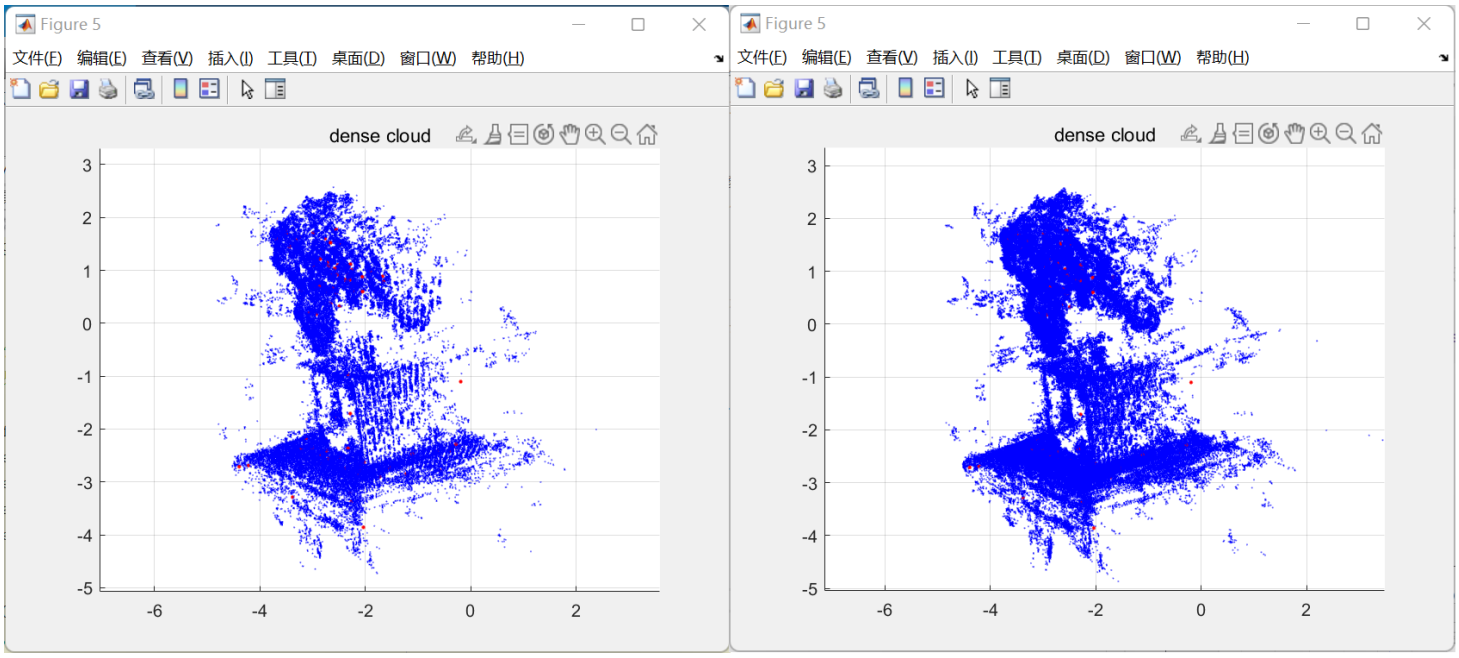
其实给定的0.05参数是一个很紧的结果，我们并不需要限制的这么死，有的时候我们希望用杂点来换取更高的还原度，所以我对这个参数进行了一些修改，发现0.30也是一个很好的平衡点，虽然多了一些杂点，但还原度确实得到了提升。而且对于有多张图片的数据集，有时候两个相机之间的夹角并不会很大，我们可以适当放宽 `Graph{frame}.cos_angles < cos(5 / 180 * pi)` 的限制

关于mesh重建的思考

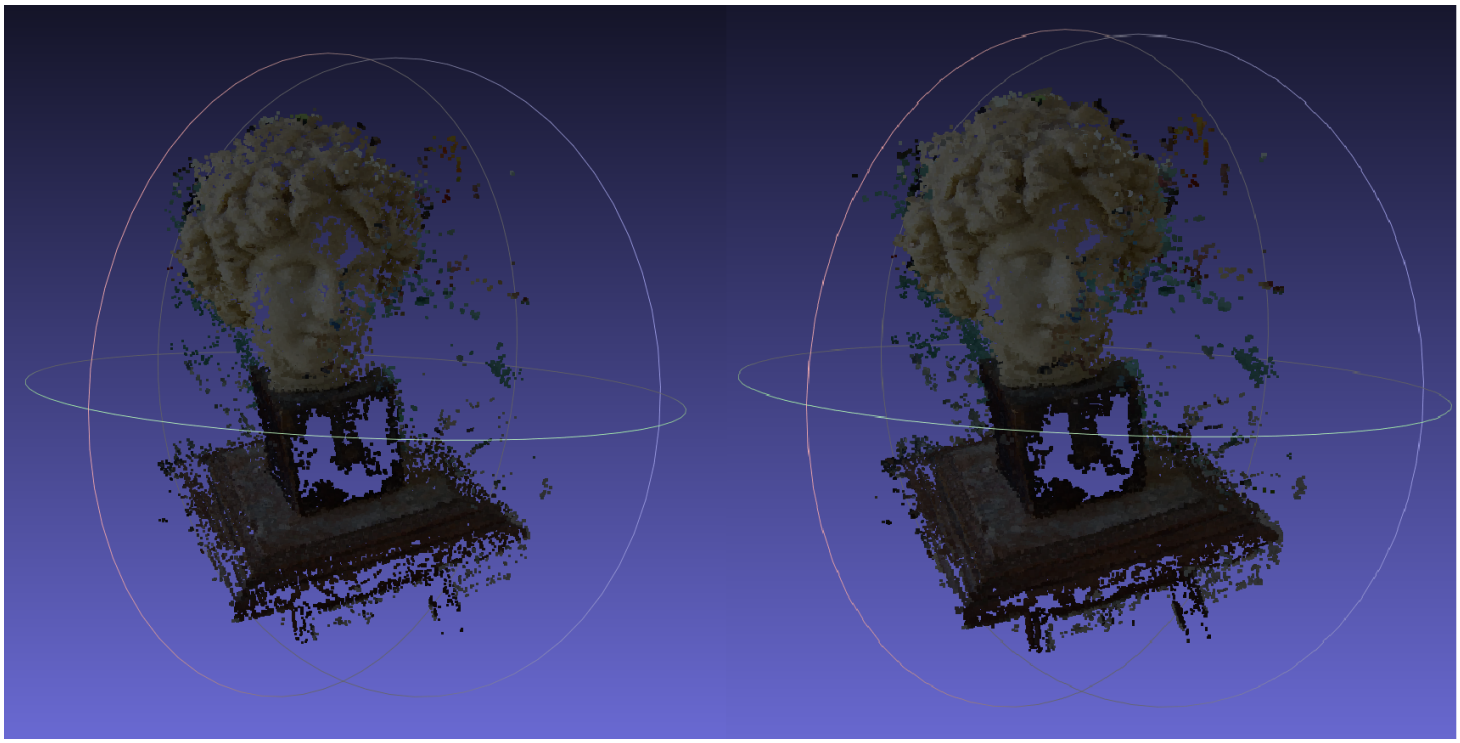
助教给的ppt里提到了得到点云之后我们还可以对点云进行进一步加工最终生成一张mesh，我查阅资料后发现可以通过meshlab的smooth normal对法向量进行去噪，然后根据法向量生成mesh。

五.结果展示

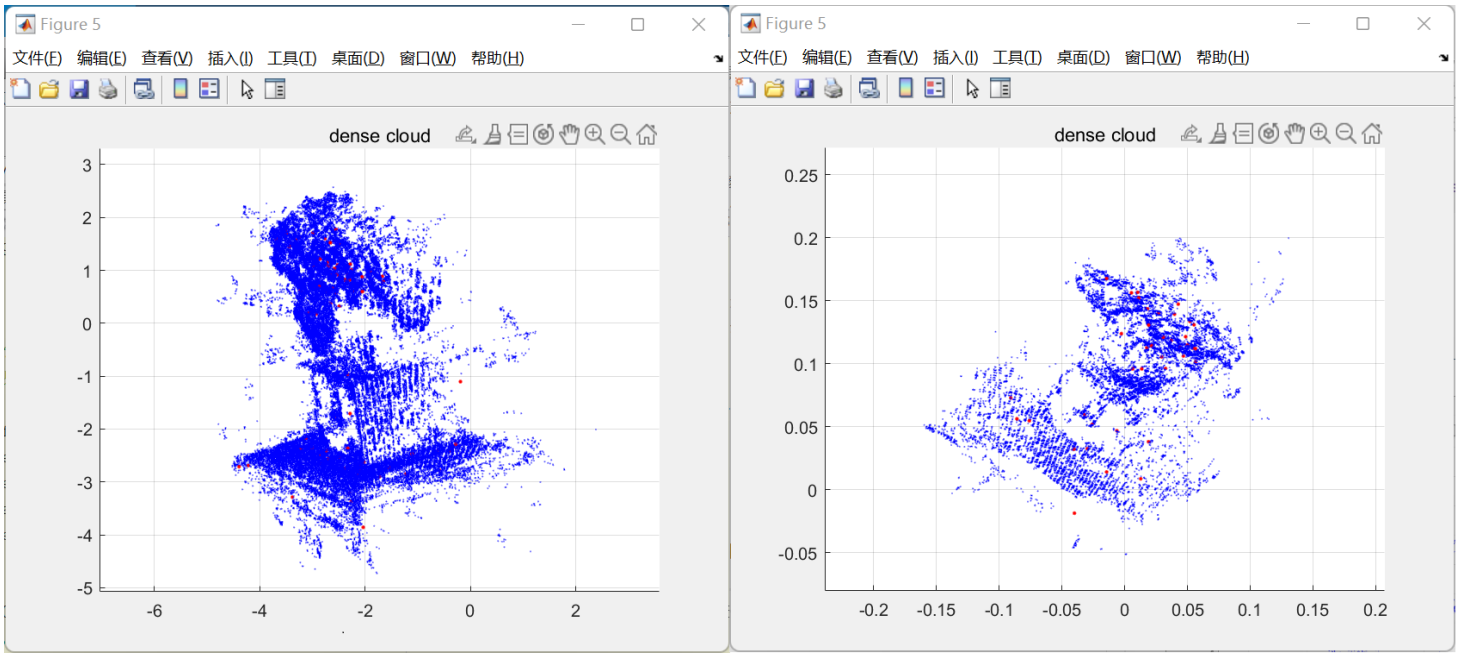
初始测试样例



左侧是取error，即最后限定的上界为0.05，右侧为限定为0.30。二者都是在前面采用estimateF方法求解F。它们的点云文件生成为：



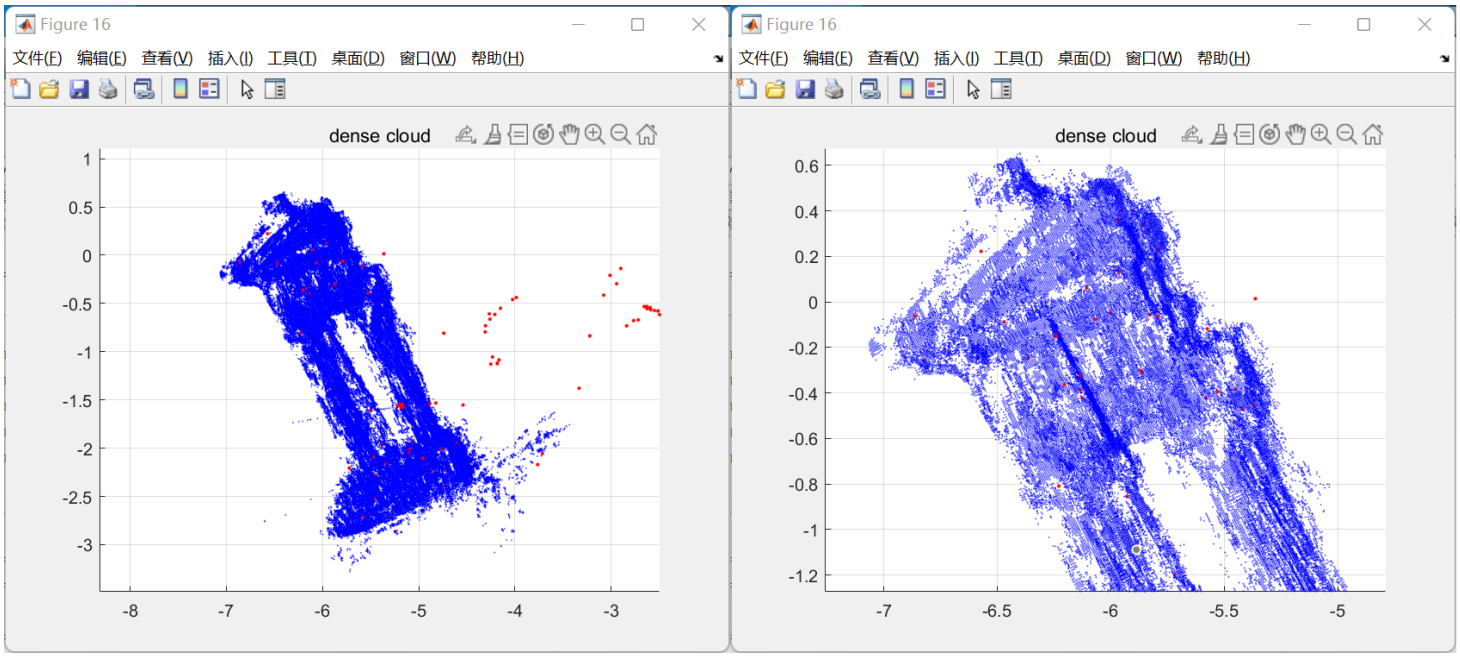
可以看到稠密了不少，对于底座部分补全了不少，但同时也让角色的脸部旁边出现了很多杂点，总之各有利弊。我们再比较一下estimateF和estimateE方法：

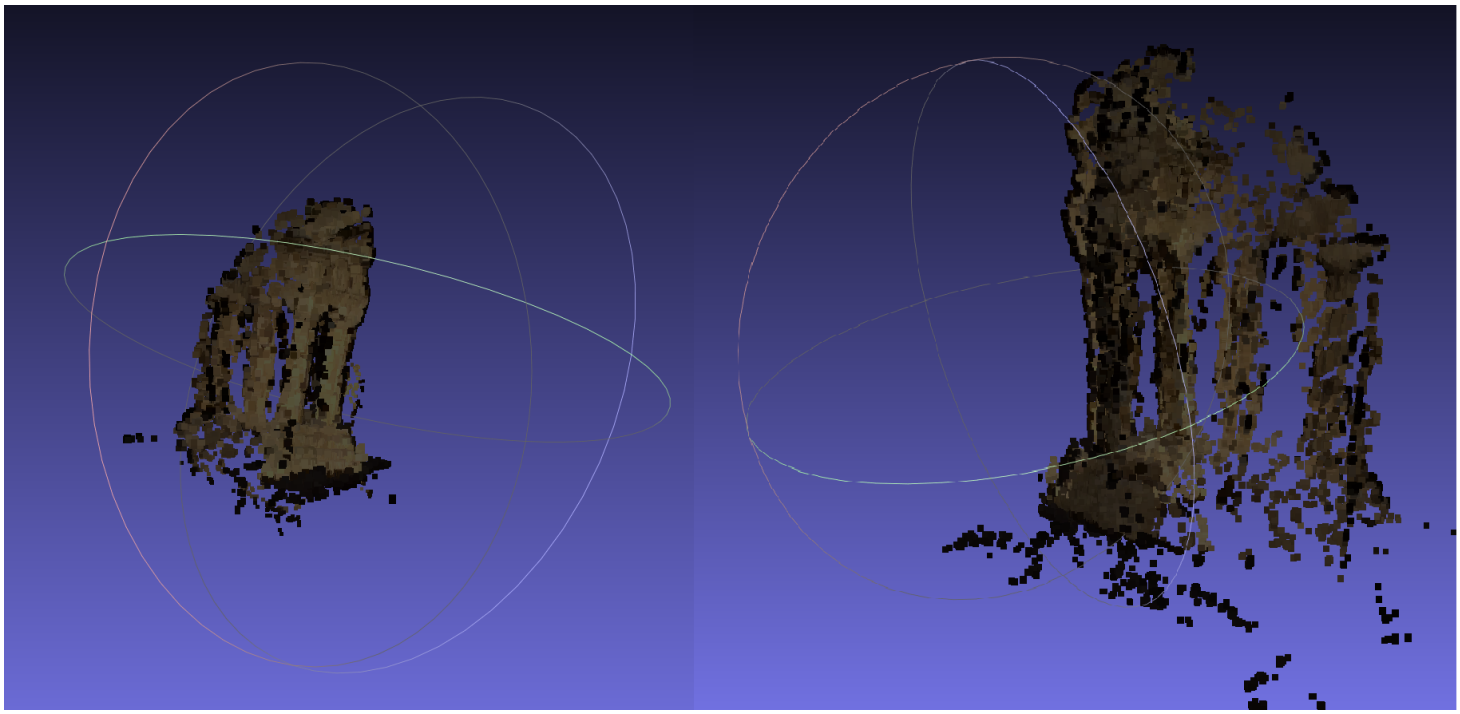


肉眼可见的稀疏了不少

其他样例

templeSparseRing 16张





templeRing 47张

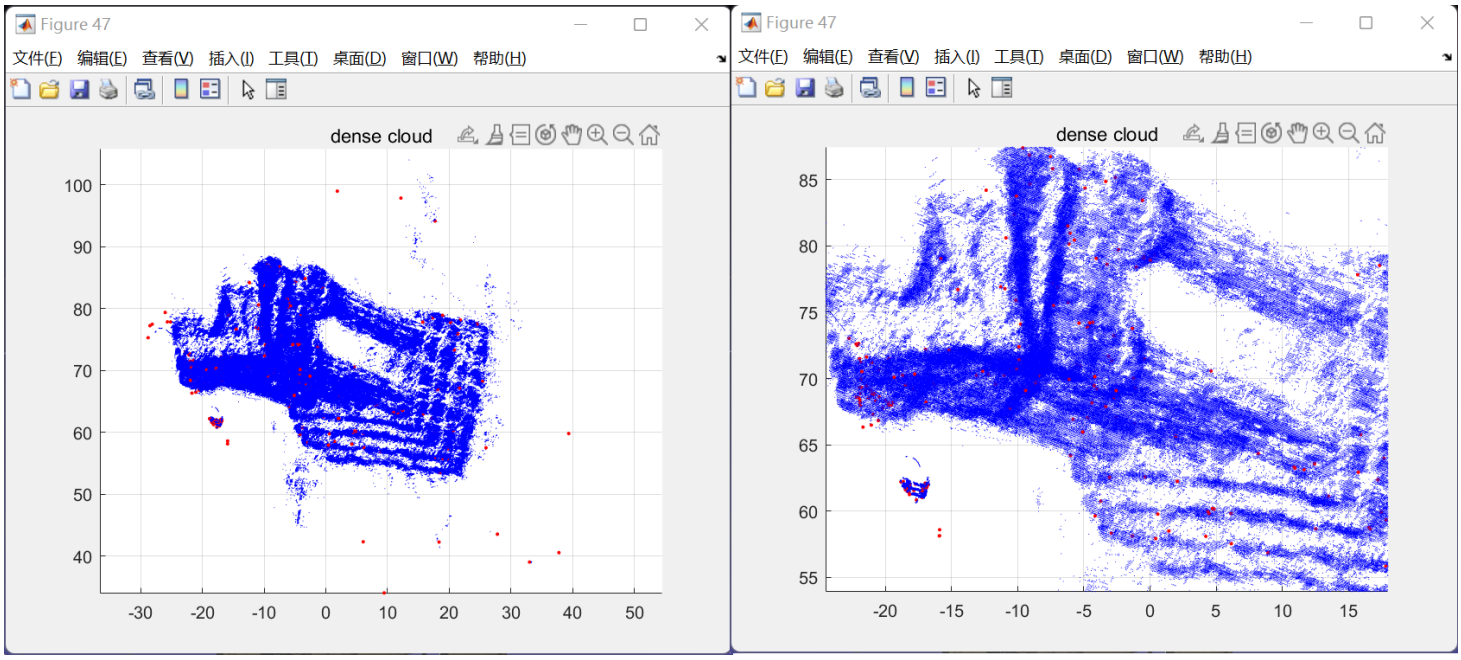
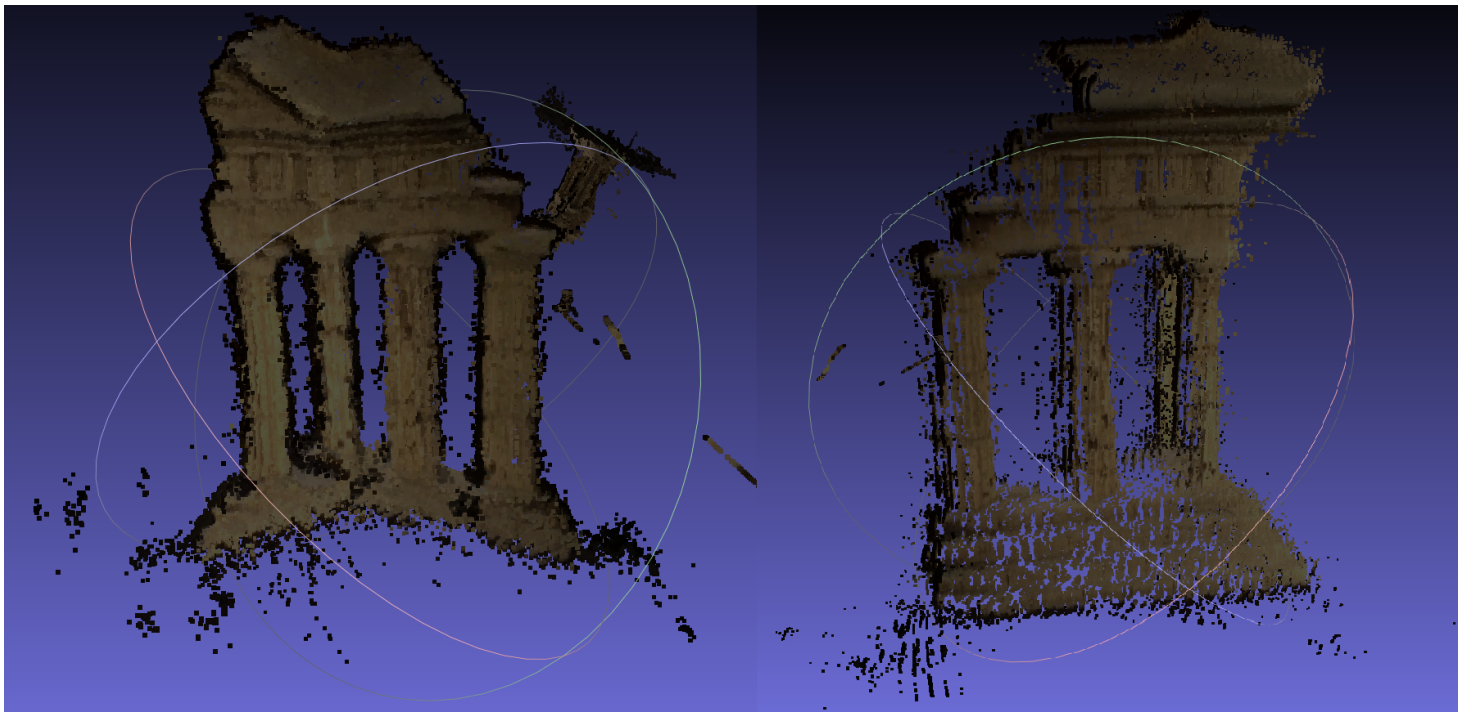
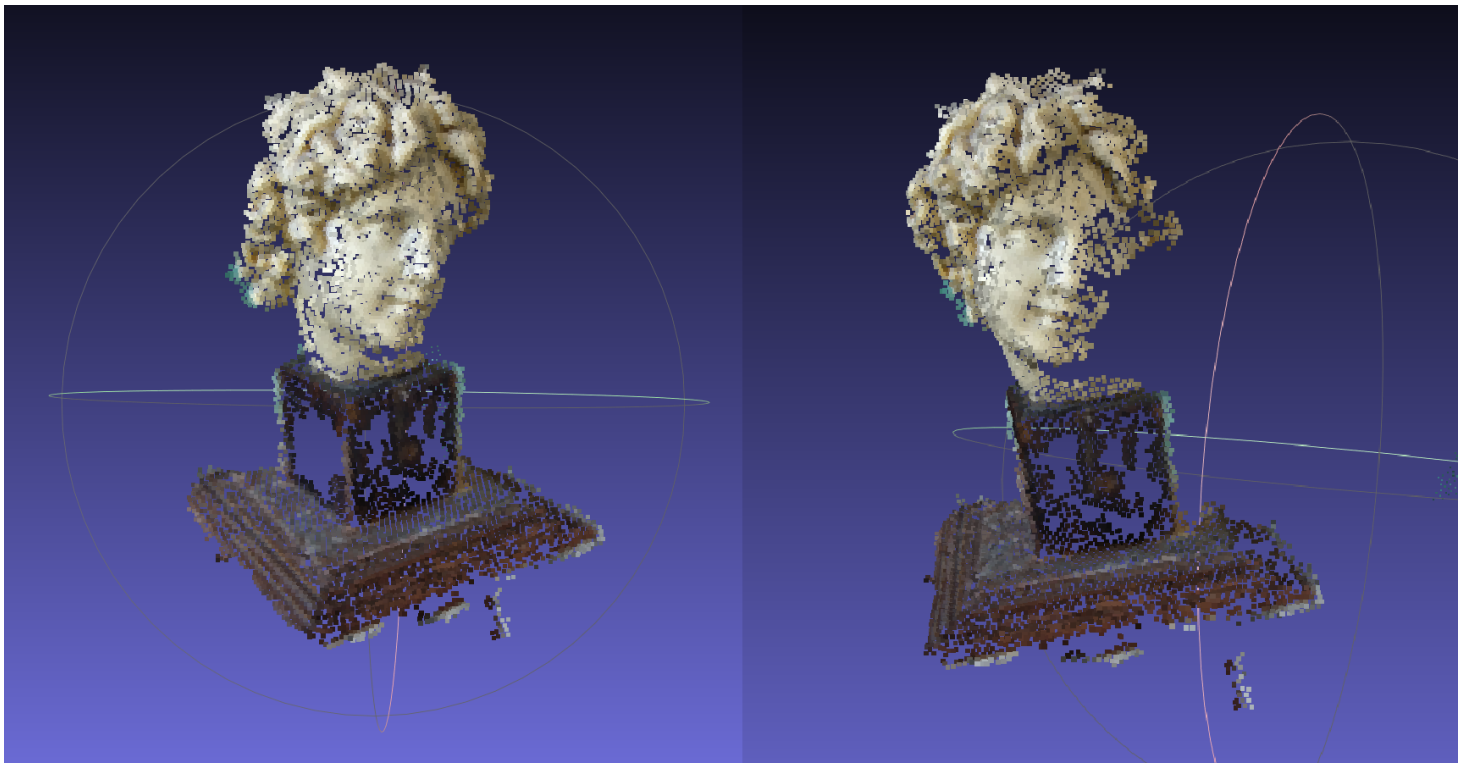


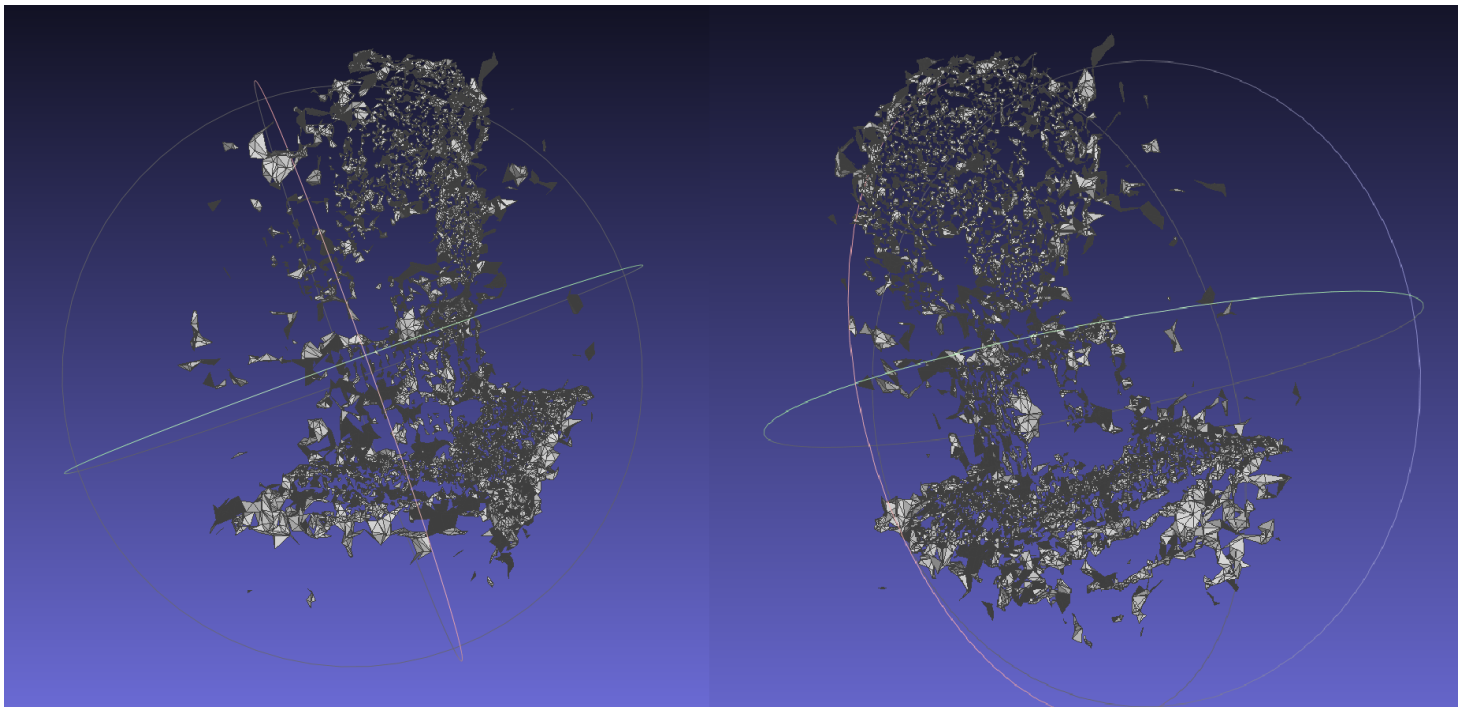
Figure 47
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
dense cloud
100
90
80
70
60
50
40
-30 -20 -10 0 10 20 30 40 50
Figure 47
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)
dense cloud
85
80
75
70
65
60
55
-20 -15 -10 -5 0 5 10 15

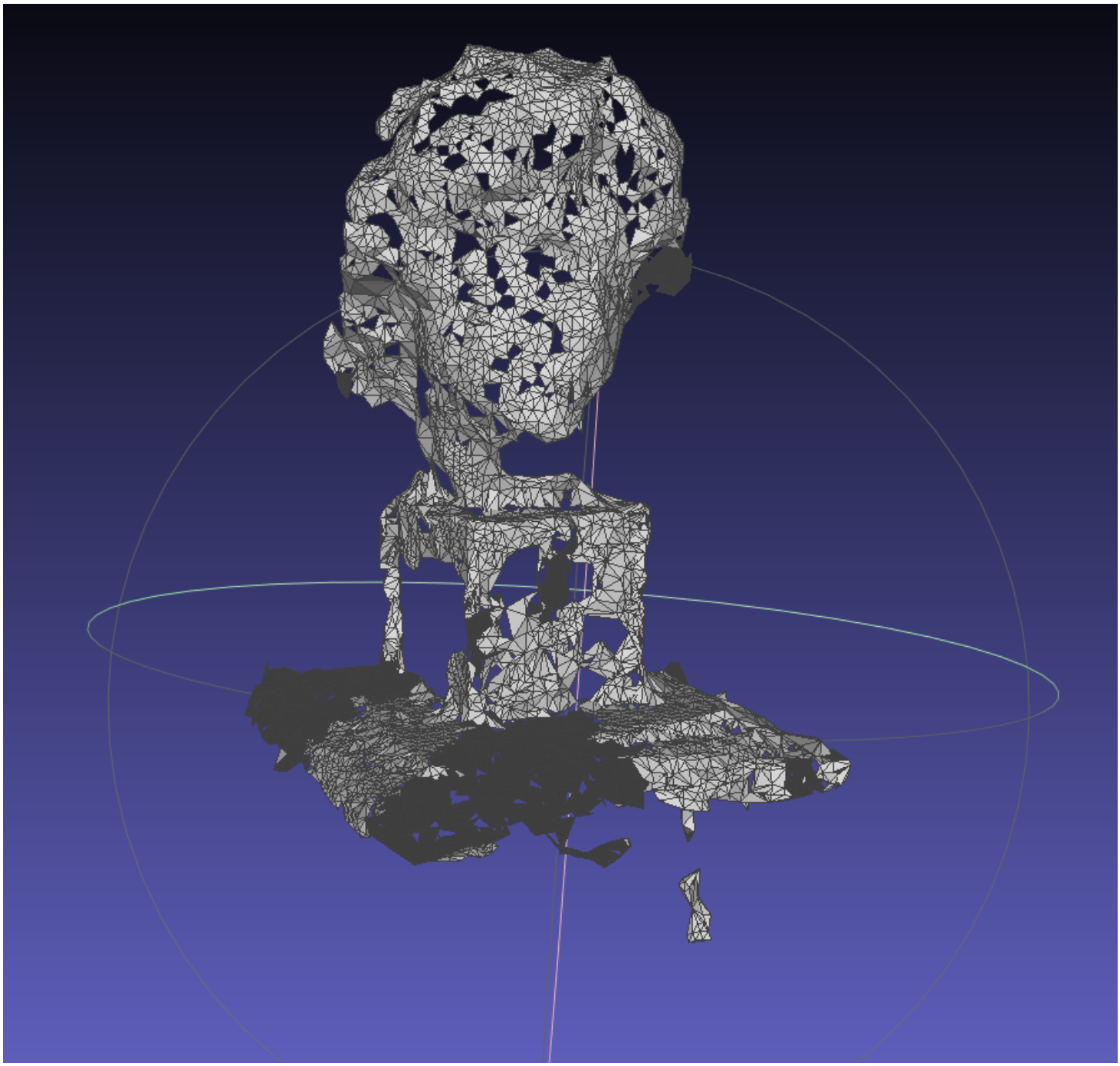


CUDA-SFM比较



mesh重建





结果分析

我们可以发现，从性能上来看estimateF方法比estimateE方法好了很多，点云更加稠密，建模也更加真实。而且适当放宽error的限制也可以让求得的结果更完善，当然，也有可能带来更多的杂点。从其他样例来说，速度明显慢了很多，16张图片可能要跑5min，而47张图跑了6-7个小时才成功。但不得不说47张图的效果很好，因为是360度角拍摄，所以各个方位都能重建出来，边界和细节也都有体现。
对于使用CUDA跑的GPU版SFM(来自 `http://ccwu.me/vsfm/doc.html#faq`),速度快了很多倍，2s就能出结果，但从结果上来看准确度不如框架高，脸的右端有一些模糊，缺点。不过这个算法对点云进行了优化，可以发现看起来很平整而且基本没有游离的散点。
mesh重建效果不是很好，事实上应该是因为点太散了，会有很多噪声造成的。但对于GPU上的SFM比较平滑，可以有很好的效果。