实验四：立体视觉

0. 关键信息

本实验 **2 人 1 组**完成（报告中要注明两人工作量的比例）。

代码与报告提交信箱：visionexp@126.com

截止日期：2019 年 12 月 25 日 23 点 59 分

1. 概述

本实验将运用平面扫描立体视觉与光度测量立体视觉的方法，来恢复图像深度，并建立立体图。实验包含三个部分：

（1）光度测量立体视觉（详见讲义第 18 讲）：给定在不同的已知光照方向下从相同视角拍摄的一组图像，从中恢复物体表面的反照率(albedo)和法线方向(normals)。

（2）平面扫描立体视觉（详见讲义第 16 讲）：给定同一场景从不同的视角拍摄的两幅校准图像，从中恢复出粗略的深度图。

（3）基于泊松方程重建深度图（详见讲义第 18 讲）：根据法线图及粗略深度图，恢复出物体每个点的深度，并重建 3D 网格。

你要实现的所有代码都在 student.py 中。

2. 实施细节

2.1 光度测量立体视觉

物体表面对入射光的漫反射可以用朗伯(Lambertian)方程描述：

$$I = k_d \mathbf{N} \cdot \mathbf{L}$$

其中k_d表示该点的反照率(albedo)，N为该点的表面法线方向(normal)，L为光线入射方向，I为观察到的该点图像强度。

给定若干光线入射方向L及所观察到的图像强度I，就可以求解出每个点的反照率k_d和法线方向N。例如给定物体上一个点的三个方程：

$$\begin{aligned} I_1 &= k_d \mathbf{N} \cdot \mathbf{L_1} \\ I_2 &= k_d \mathbf{N} \cdot \mathbf{L_2} \\ I_3 &= k_d \mathbf{N} \cdot \mathbf{L_3} \end{aligned}$$

可以将其写为矩阵方程：

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = k_d \begin{bmatrix} \mathbf{L_1}^T \\ \mathbf{L_2}^T \\ \mathbf{L_3}^T \end{bmatrix} \mathbf{N}$$

$$\underbrace{\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}}_{\mathbf{I}} = \underbrace{\begin{bmatrix} \mathbf{L_1}^T \\ \mathbf{L_2}^T \\ \mathbf{L_3}^T \end{bmatrix}}_{\mathbf{L}} \underbrace{k_d \mathbf{N}}_{\mathbf{G}}$$

$3 \times 1 \qquad 3 \times 3 \quad 3 \times 1$

解此方程得：$\mathbf{G} = \mathbf{L}^{-1}\mathbf{I}$

$$k_d = \|\mathbf{G}\|$$

$$\mathbf{N} = \frac{1}{k_d}\mathbf{G}$$

当有更多光源时，

$$\begin{bmatrix} I_1 \\ \vdots \\ I_n \end{bmatrix} = \begin{bmatrix} \mathbf{L_1} \\ \vdots \\ \mathbf{L_n} \end{bmatrix} k_d \mathbf{N}$$

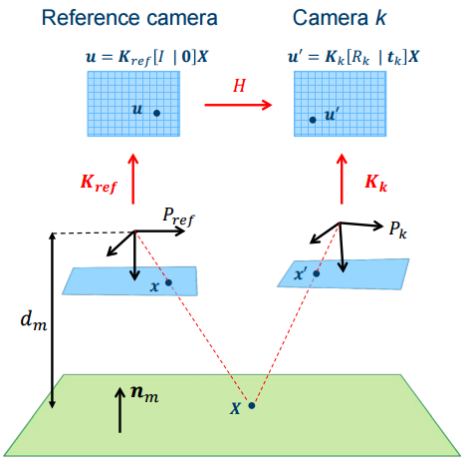
用最小二乘法求解：

$$\begin{aligned} \mathbf{I} &= \mathbf{L}\mathbf{G} \\ \mathbf{L^T I} &= \mathbf{L^T L G} \\ \mathbf{G} &= (\mathbf{L^T L})^{-1}(\mathbf{L^T I}) \end{aligned}$$

对于多通道图像，如 RGB 图，每个像素处每个通道有自己的反照率，但是法线方向对所有通道是相同的。

你需要实现 student.py 中的 compute_photometric_stereo_impl 函数。

2.2 平面扫描立体视觉



平面扫描立体视觉的具体步骤如下（详见讲义第 16 讲：多视图立体视觉）：

1. 将每幅图像I_k相对于每个深度平面Π_m投射到参考平面P_{ref}，使用单应映射$H^{-1}_{\Pi_m,P_k}$，得到的卷绕图像记为$\check{I}_{k,m}$
2. 计算I_{ref}和$\check{I}_{k,m}$的相似度

   使用 ZNCC (Zero-mean Normalized Cross Correlation)
3. 对每个深度平面计算所有 k 幅图片的相似度

$$M(u,v,\Pi_m) = \sum_k ZNCC_W\left(I_{ref}, \check{I}_{k,m}\right)$$

4. 对每个像素，选择最佳深度

$$\check{\Pi}(u,v) = \underset{m}{\operatorname{argmax}}\, M(u,v,\Pi_m)$$

本实验中，你只需要实现上述步骤中的三个具体函数：project_impl、preprocess_ncc_impl、compute_ncc_impl，这三个函数都在 student.py 中。

2.2.1 project_impl

该函数将三维点坐标映射到一个标定过的相机坐标（详细原理见第 10 讲：相机），投影矩阵为

$$\Pi = K\,[R \mid t] \quad ,$$

其中 K 为 3*3 的内参数矩阵，$[R \mid t]$为 3*4 的外参数矩阵。

映射公式为

$$u = \Pi\, X \quad ,$$

其中 X 为三维点的齐次坐标(4*1)，u 为映射到相机平面上的二维点的齐次

坐标(3*1)。

2.2.2 preprocess_ncc_impl

该函数为 ZNCC (Zero-mean Normalized Cross Correlation)计算做预处理。

图像中每个像素处的 ZNCC 是对以该像素为中心的一小块区域(patch)做以下计算：

$$\mathrm{ZNCC} = \frac{\sum_{x,y}(W_1(x,y)-\overline{W_1})(W_2(x,y)-\overline{W_2})}{\sqrt{\sum_{x,y}(W_1(x,y)-\overline{W_1})^2}\sqrt{\sum_{x,y}(W_2(x,y)-\overline{W_2})^2}}$$

其中$\overline{W_i} = \frac{1}{n}\sum_{x,y} W_i(x,y)$为均值；$W_i(x,y)$是图像 *i* 中坐标(*x,y*)处的像素值。该区域(patch)的大小由 preprocess_ncc_impl 的参数 ncc_size 决定；patch 为 ncc_size*ncc_size 的正方形。

preprocess_ncc_impl 为预处理，即计算一幅图像每个像素点周围 patch 中的值：

$$\frac{W(x,y)-\overline{W}}{\sqrt{\sum_{x,y}(W(x,y)-\overline{W})^2}}$$

当通道数为 channels 时，得到一个长度为 channels * ncc_size* ncc_size 的向量。

求平均时，每个通道单独做。归一化时（即做除法时），对所有通道一起做（即求$\sqrt{\sum_{x,y}(\mathrm{W}(x,y)-\overline{W})^2}$是对 channels * ncc_size* ncc_size 个值一起做）。

当小区域(patch)有部分在图像边界以外时，将整个向量设为 0 值。

2.2.3 compute_ncc_impl

对 preprocess_ncc_impl 得到的 patch 向量，将两幅图中每个像素处的两个 patch 向量做内积（即对应点相乘并求和）。

2.3 基于泊松方程重建深度图

泊松方程根据法线方向计算深度（详见讲义第 18 讲：光度测量立体视觉）。

每个点处的两个方程为：

$$\mathrm{n}_x = \mathrm{n}_z z_{x+1,y} - \mathrm{n}_z z_{x,y}$$
$$-\mathrm{n}_y = \mathrm{n}_z z_{x,y+1} - \mathrm{n}_z z_{x,y}$$

其中(n_x, n_y, n_z)为该点处的法向量(即法线方向)，z 为深度。注：此处法向量沿+x、+y、-z 轴为正。

你需要实现的 form_poisson_equation_impl 函数返回线性方程 Ax=b 的参数；其中 x 为所有点的深度，x 为 height*width 大小的向量，是未知数；A 和 b 是要返回的参数。之后会使用最小二乘法求解该方程得到每个点的深度。

其余请仔细阅读 student.py 中的注释。

3. 测试

我们使用 Nose 进行测试。在实验目录下运行命令行 nosetests，可以运行 tests.py 中的所有测试例。

当你第一次运行 nosetests 时，将看到所有测试都被跳过。

```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS
----------------------------------------------
Ran 40 tests in 0.869s

OK（SKIP = 40）
```

我们已经配置了 tests.py 来跳过任何已经引发 NotImplementedError 的测试。跳过的测试结果为一个 S。

实现一个函数后，您可能会看到类似这样的内容。

```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS ... S
----------------------------------------------
Ran 40 tests in 0.639s

OK（SKIP = 37）
```

在这里，我们通过了三个测试，每个用一个点表示。

如果测试用例失败， 则会打印 F。例如：

```
SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSFFFS
==============================================
FAIL: tests.unproject_Rt_identity_1x1_test
----------------------------------------------------------
-------------
Traceback (most recent call last):
 File "/usr/lib/python2.7/site-packages/nose/case.py",
line 197, in runTest
  self.test(*self.arg)
 File "/Users/ubuntu/cs5670/pa4/tests.py", line 15, in
wrapper   unc()
 File "/Users/ubuntu/cs5670/pa4/tests.py", line 385, in
assert (point[0, 0, 0] == -0.5).all()
AssertionError
...
----------------------------------------------------------
-------------
Ran 40 tests in 0.885s

FAILED (SKIP=37, failures=3)
```

你的实现应首先通过所有测试例。

4. 实验环境与数据集

本实验在 Python3 下进行，除了需要 Numpy、Scipy、OpenCV for python 以外，还需要安装 imageio（用于读写图片文件）、Nose（用于测试）、ImageMagick（用于生成 gif 动态图）和 MeshLab（用于查看点云数据）。

在实验开始，首先运行以下代码：

```
cd data

sh download.sh

cd ..

mkdir output

mkdir temp
```

其中，data 目录下的 download.sh 里指明了所需数据集的下载链接，在 windows 环境下可以直接拷贝这些链接去下载。tentacle 数据集本实验环境已经自带了，其它数据集需要自行下载。

5. 实验结果验证

5.1 光度测量立体视觉

在编码完成并通过 nosetests 测试后，在命令行中运行

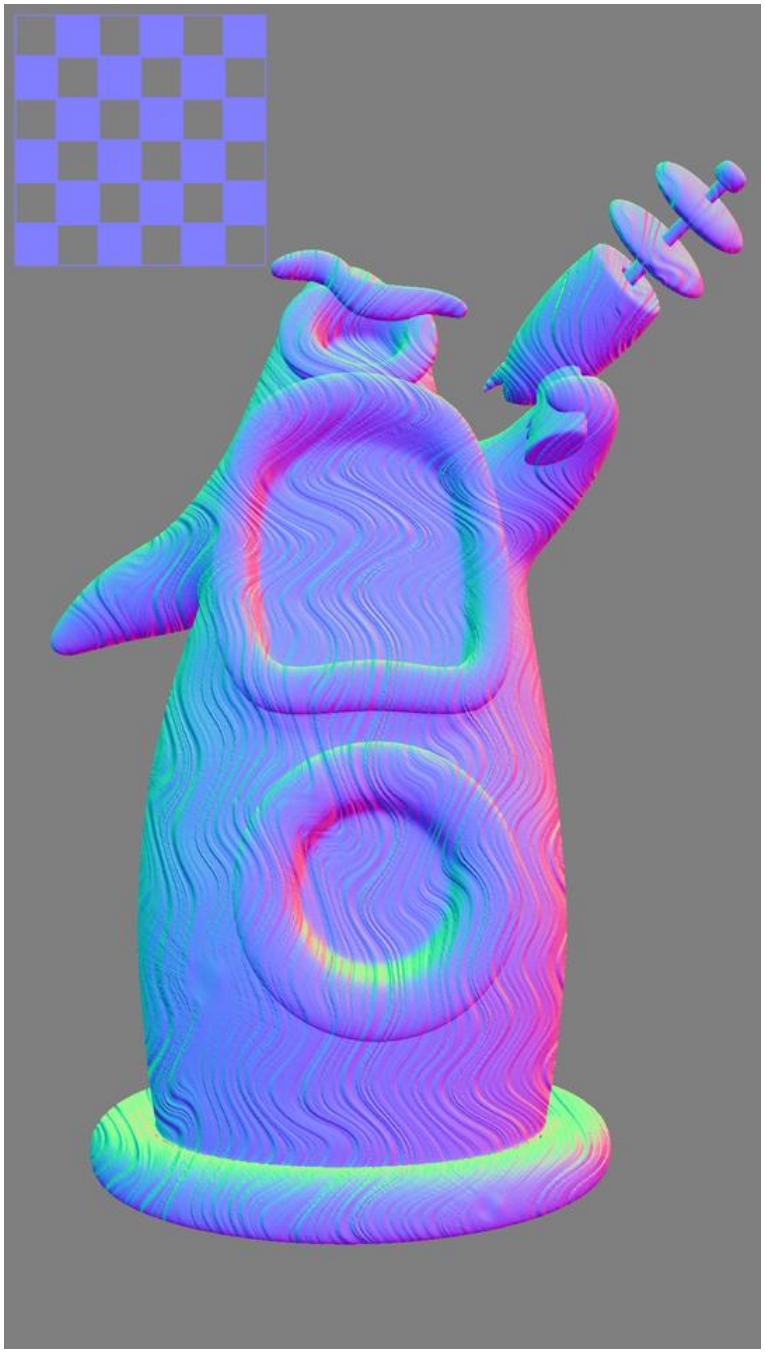
```
python photometric_stereo.py <dataset>
```

其中<dataset>为 ('tentacle', 'cat', 'frog', 'hippo', 'lizard', 'pig', 'scholar', 'turtle')中的一个。例如，如果使用 tentacle 数据集，则运行

```
python photometric_stereo.py tentacle
```

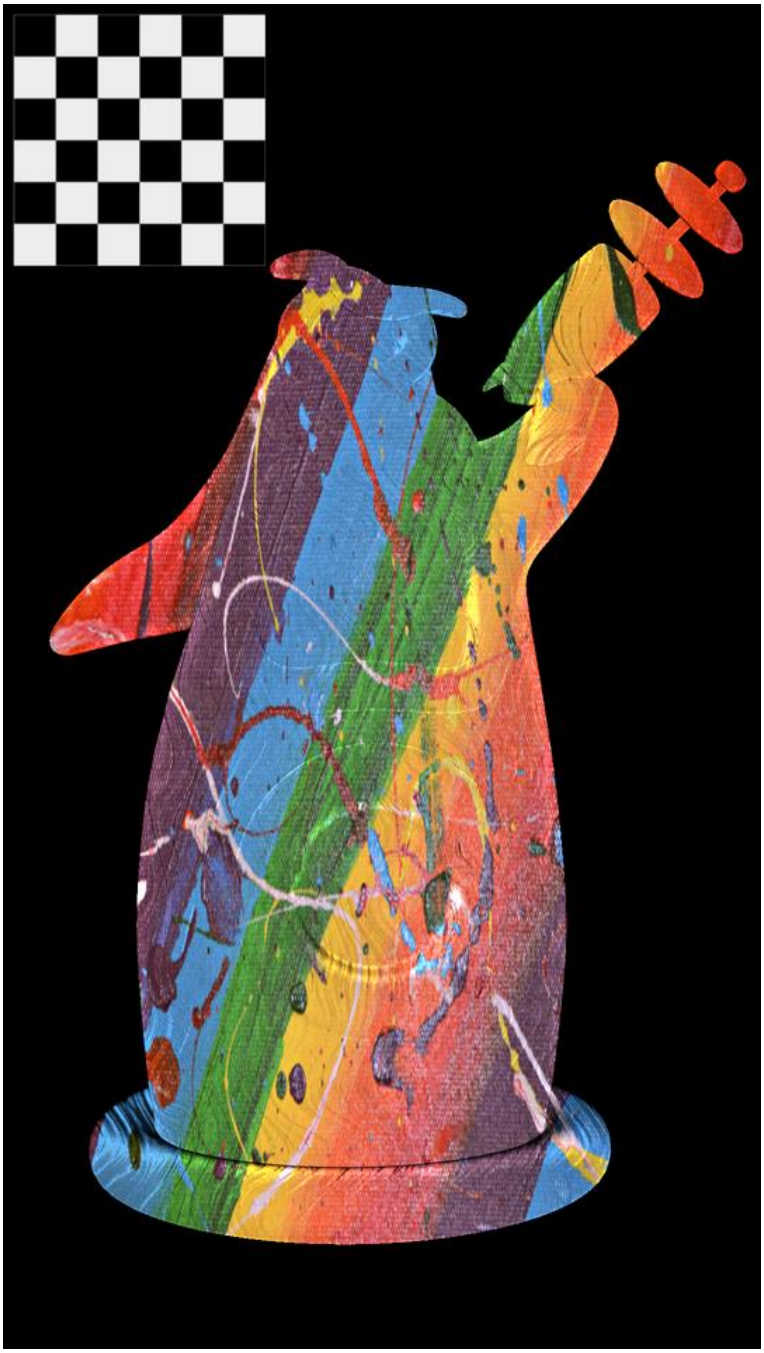
输出将为 `output/tentacle_{normals.png,normals.npy,albedo.png}`。

正确的 `tentacle_normals.png` 应为：



其中，红色表示法线指向右侧（+ x 方向），绿色表示法线指向上方（+ y 方向），蓝色表示法线指向屏幕外（+ z 方向）。我们提供的照明光线方向已经在此坐标系中，因此默认情况下最简单的解决方案应该是正确的。

正确的 tentacle_albedo.png 应为：



运行时间大概为几十秒到 2 分钟，取决于你的实现。

你应运行 tentacle 数据集和 cat 数据集，并提交 output/{tentacle,cat}_normals.png 和 output/{tentacle,cat}_albedo.png。

5.2 平面扫描立体视觉

在编码完成并通过 nosetests 测试后，在命令行中运行

```
python plane_sweep_stereo.py <dataset>
```

其中 <dataset> 为 ('tentacle', 'Adirondack', 'Backpack', 'Bicycle1', 'Cable', 'Classroom1', 'Couch', 'Flowers', 'Jadeplant', 'Mask', 'Motorcycle', 'Piano', 'Pipes', 'Playroom', 'Playtable', 'Recycle', 'Shelves', 'Shopvac', 'Sticks', 'Storage', 'Sword1',

'Sword2', 'Umbrella', 'Vintage')中的一个。你需要取消 data\download.sh 中的注释，并下载相应数据集。

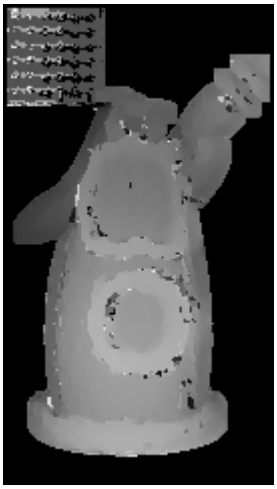
例如，如果使用 tentacle 数据集

```
python plane_sweep_stereo.py tentacle
```

输出将为 output / tentacle_ {ncc.png，ncc.gif，depth.npy，projected.gif}。

其中正确的 tentacle_ projected.gif 如“示例 tentacle_projected.gif”所示。tentacle_ncc.gif 如“示例 tentacle_ncc.gif”所示。（注意：生成 gif 图像需要事先安装 ImageMagick。）

正确的 tentacle_ncc.png 如下图所示：



图中白色表示近，黑色表示远。

你应运行 tentacle 和 Flowers 两个数据集，并提交 output/{tentacle,Flowers}_ {ncc.png,ncc.gif, projected.gif}

5.3 基于泊松方程重建深度图

在编码完成并通过 nosetests 测试后，在命令行中运行

```
python combine.py <dataset> <mode>
```

其中<dataset> 为('tentacle', 'cat', 'frog', 'hippo', 'lizard', 'pig', 'scholar', 'turtle', 'Adirondack', 'Backpack', 'Bicycle1', 'Cable', 'Classroom1', 'Couch', 'Flowers', 'Jadeplant', 'Mask', 'Motorcycle', 'Piano', 'Pipes', 'Playroom', 'Playtable', 'Recycle', 'Shelves', 'Shopvac', 'Sticks', 'Storage', 'Sword1', 'Sword2', 'Umbrella', 'Vintage')中的一个，<mode> 为('normals', 'depth', 'both')中的一个。

例如，如果使用 tentacle 数据，你应输入

```
python combine.py tentacle both
```

tentacle 数据集是唯一可以用 both 模式的数据集。其它数据集只能用 normals（光度测量得到）或 depth（平面扫描得到）中的一个。

运行时需要在 temp 目录中写临时文件，因此请事先建立一个空的 temp 目录。

执行完成后得到在 output 目录中的 ply 文件，可以用 Meshlab 打开（使用 Import Mesh 按钮）。在 Meshlab 中正确的显示如视频 tentacle_mesh_both.mp4 所示。

运行以下数据集和模式组合：

- tentacle 数据集，模式 both
- tentacle 数据集，模式 depth
- cat 数据集，模式 normals
- Flowers 数据集，模式 depth

对每一个组合，在 Meshlab 中查看点云，截图保存，并简单描述三维点云中哪些部分看起来比较好，哪些看起来有问题，并简单分析光度测量立体视觉和平面扫描立体视觉算法为什么会产生这样的错误。

6. 所提供的文件

student.py：所需要实现的所有函数都在此文件中。
tests.py：用于 nosetests 测试。
photometric_stereo.py：用于产生光度测量实验结果。
plane_sweep_stereo.py：用于产生平面扫描实验结果。
combine.py：用于产生泊松方程求解深度实验结果。
dataset.py：用于读取数据集。
gifwriter.py：用于平面扫描部分输出 gif 图。
util.py：辅助函数实现。
util_sweep.py：平面扫描辅助函数实现。
示例 tentacle_projected.gif 和示例 tentacle_ncc.gif：平面扫描示例输出结果。
tentacle_mesh_both.mp4：Meshlab 读取泊松方程求解深度结果示例。
data\download.sh：所需要下载的数据集链接。
Input 目录：tentacle 数据集。
test_materials 目录：测试所用数据。

7. 提交文件

student.py：所完成的所有代码都在此文件中。
tentacle_normals.png、**tentacle_albedo.png**、**cat_normals.png**、**cat_albedo.png**：光度测量部分的实验结果，运行完毕后在 output 目录中。
tentacle_ncc.png、**tentacle_ncc.gif**、**tentacle_ projected.gif**、**Flowers_ncc.png**、**Flowers_ncc.gif**、**Flowers_projected.gif**：平面扫描部分的实验结果，运行完毕后在 output 目录中。
泊松方程求解深度部分的输出结果在 Meshlab 中的查看截图。
report(.docx,.pdf)：报告中应评价 Meshlab 中显示的各个实验结果，分析光度测量与平面扫描立体视觉算法的问题。报告还可以包括其它你认为实验过程中有趣的内容。报告中应注明两人工作量的比例。

将以上文件打包压缩，压缩后的文件命名为“学号 1+姓名 1+学号 2+姓名 2+expX”; 如“16030140095 李乐天+16030140012 王学斌 exp4.rar”、“16030140095 李乐天+16030140012 王学斌 exp4.zip”。Email 的标题类似命名。

8. 调试注意事项

本实验推荐在 Linux 上进行

•测试的时候发现 Windows 平台上大概率运行不起来第二个子实验(因此第三个子实验也会受到影响)，无论 python 2 还是 python 3，都会报出"找不到文件的错误"。这个跟 Windows 和 Linux 的地址编码方式不同以及底层 lib、命令行有关系。这个问题比较复杂，我在三个虚拟机上进行了测试，发现有的可以有的不行，似乎还跟 Win10 的版本有关，所以最好还是自己解决，请参考 gifwriter.py 第 14 行附近的中文注释。简单来说，如果报错：FileNotFoundError: [WinError 2] 系统找不到指定的文件，请参考 StackOverFlow 上的解决方案：
In Windows , to use echo in subprocess, you would need to use shell=True . This is because echo is not a separate executable, but rather a built-in command for the windows command line.
•同样，针对 Windows 平台，测试的时候我发现如果电脑里装了 Git，并添加了相应的环境变量，能有效减少各种玄学问题。原因不详，可能跟 Git 自带的 sh 文件有关。
•可以尝试使用 Win10 的 Power Shell(快捷键 Win+x, a)运行命令行，或许有助于减少玄学问题。

本实验原为 python2，改到 python3 修改了以下内容：
•TypeError: 'map' object is not subscriptable，把报错行或相关行的内容进行 python 3 风格的转化 map(...)->list(map(...))
•TypeError: object of type 'map' has no len()，这个问题一般来自 datasets.py 文件，修改方法同上；详细解决方案见 photometric_stereo.py 中的 15 行左右的中文注释内容。
•gifwriter.py 文件针对 Windows 平台进行了调整；对于未调整前的文件，我猜测理论上只有 Win7 或更早之前的版本(或者使用 Win10 的 Power Shell)才能运行这个实验。
仅测试了 tentacle 数据集，其它数据集如果有以上类似报错，按以上方法修改。