实验四：立体视觉

1. 实验目的
   1. 掌握光度测量立体视觉
   2. 掌握平面扫描立体视觉

* 1. 学会利用基于泊松方程重建深度图

1. 实验原理

### 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：

Edittex

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

### 给定若干光线入射方向及所观察到的图像强度，就可以求解出每个点的反照率和法线方向。

### Edittex

### 用最小二乘法求解，可得到反照率和法线方向。

### 平面扫描立体视觉

，将每幅图像相对于每个深度平面投射到参考平面，使用单应映射，得到的卷绕图像记为

### ， 计算I\_ref和I ̆\_(k,m)的相似度，使用ZNCC

，对每个深度平面计算所有k幅图片的相似度

，对每个像素，选择最佳深度

### 首先，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

### Π=K [R | t] ，

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

### 映射公式为

### u=Π X ，

### 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)。

然后，对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：

其中为均值；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：

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

最后，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）。

### 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：

### n\_x=n\_z z\_(x+1,y)-n\_z z\_(x,y)

### 〖-n〗\_y=n\_z z\_(x,y+1)-n\_z z\_(x,y)

1. 实验步骤

1,完成compute\_photometric\_stereo\_impl

对于每个像素的每一个通道，进行如下运算：

I = images\_arr[:, i, j, k].reshape(length, 1)  
G = np.dot(np.linalg.inv(np.dot(lights.T, lights)), np.dot(lights.T, I)).reshape(3, 1)  
kd = np.linalg.norm(G)  
kd = 1 if kd == 0 else kdalbedo[i, j, k] = kd  
N = (G \* (1. / kd)).reshape(3)

2，完成project\_impl

在每个像素处，进行如下运算：

\_x, \_y, \_z = points[i, j, :]  
u = np.dot(PI, np.array([\_x, \_y, \_z, 1]).reshape(4, 1))  
u = u / u[2, 0]  
projections[i, j, :] = u[:2, 0]

3，完成preprocess\_ncc\_impl

在每个像素处，进行如下运算：

temp = image[(i - border):(i + border + 1), (j - border):(j + border + 1), :]  
means = np.mean(np.mean(temp, axis=1), axis=0)  
patch = temp - means  
patch\_t = np.sqrt(np.sum(patch \*\* 2))  
patch\_t = 1 if patch\_t == 0 else patch\_t#防止除0  
pre = patch / patch\_t  
patch = np.vstack(pre)  
normalized[i, j, :] = patch.T.reshape(1, 1, -1)

4，完成compute\_ncc\_impl

在每个像素处，执行

ncc[i, j] = np.sum(image1[i, j, :] \* image2[i, j, :])

以完成规格化。

5，完成form\_poisson\_equation\_impl

对于每个像素：

如果depth是none则执行以下代码：

row\_ind.append(row)  
row += 1  
col\_ind.append(i \* width + j)  
data\_arr.append(depth\_weight)  
b.append(depth\_weight \* depth[i, j])

如果normals不是none则执行以下代码：

row\_ind.append(row)  
col\_ind.append(i \* width + j)  
data\_arr.append(-normals[i, j, 2])  
row\_ind.append(row)  
col\_ind.append(i \* width + j + 1)  
data\_arr.append(normals[i, j, 2])  
row += 1  
b.append(normals[i, j, 0])  
  
row\_ind.append(row)  
col\_ind.append(i \* width + j)  
data\_arr.append(-normals[i, j, 2])  
row\_ind.append(row)  
col\_ind.append((i + 1) \* width + j)  
data\_arr.append(normals[i, j, 2])  
row += 1  
b.append(-normals[i, j, 1])

四，实验结果

* 1. 测试

运行nosetests，得到如下结果：



证明students.py基本正确，可以进行后续工作了。

* 1. 数据集运算

下载好相应的数据集。

2.1，光度测量立体视觉。

命令行运行

python photometric\_stereo.py tentacle

python photometric\_stereo.py cat

在output中成功得到了预期的图像。

### 2.2，平面扫描立体视觉

命令行运行

python plane\_sweep\_stereo.py tentacle

python plane\_sweep\_stereo.py Flowers

### 由于我使用的windows10系统，所以未能输出图像

### 2.3，基于泊松方程重建深度图

### 依次运行以下命令行：

### python combine.py tentacle both

### python combine.py tentacle depth

### python combine.py cat normal

### python combine.py Flowers depth

### 由于我使用的windows10系统，所以未能输出图像

五，实验心得：

此次实验总体较难的，而且我是一个人单独做的，所以感觉做的不太好。

这次的实验与以往的有很大不同，提供的文件非常多，数据集非常复杂，代码的注释也非常的多，我研究了好长时间才大体上知道了怎么做。

由于无法在pycharm中运行或者debug验证，所以实验过程还是比较繁琐的，运行一次都要花费挺长时间，所以做这个实验的时候遇到了很多的困难。

这次实验一个人做虽然花了我很多时间，但过程中锻炼了我搜集资料，独立思考和看论文的能力，还是很值得的。