|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学生姓名： |  | 学号： |  | 指导教师： |  |

1. **实验名称**  光学反射模型-光度立体视觉
2. **实验条件**

计算机视觉及Python 相关软件库(OpenCV，Numpy，Matplotlib，Scipy)；学生自备电脑；满足漫反射模型的6幅球面图像（LMImage0.bmp, …, LMImage5.bmp）及光源照射方向;带有印刷体痕迹纸币的6幅图像（image0.bmp, …, image5.bmp）及光源照射方向；采用RGB三色光光谱复用采集的彩色图像。

1. **实验目的**
2. 加强对光度立体视觉运算过程的认识
3. 熟悉表面法向量的计算方法
4. 了解通过法向量积分进行3D重建的方法（拓展）
5. 了解对3D数据进行交互可视化的方法（拓展）
6. **实验要求**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***Task1：通过光度立体视觉恢复球面法向量***   1. 根据提供的已知光照方向的6幅球面漫反射仿真图像，采用光度立体视觉方法计算球面的法向量。 2. 分析图像数量对法向量重建精度的影响。 3. 研究理想图像存在不同程度噪声情况下，法向量重建的准确性变化，并基于实验结果提出光度立体视觉方法在实际应用中的建议。 | |  |  | | --- | --- | | 6幅球面仿真图像 | | | E:\CV\T5Reflection\LMImage0.bmp | E:\CV\T5Reflection\LMImage1.bmp | | E:\CV\T5Reflection\LMImage2.bmp | E:\CV\T5Reflection\LMImage3.bmp | | E:\CV\T5Reflection\LMImage4.bmp | E:\CV\T5Reflection\LMImage5.bmp | | 对应6个光照方向（单位向量）：  [0.5000 0.0000 0.8660]  [0.2500 0.4330 0.8660]  [-0.2500 0.4330 0.8660]  [-0.5000 0.0000 0.8660]  [-0.2500 -0.4330 0.8660]  [0.2500 -0.4330 0.8660] | | |
| *选做1：探索基于法向量积分的三维表面重建技术，将梯度场转换为完整的三维模型，并实现模型的可视化呈现。* |  |
| *选做2：设计并实现三维可视化GUI工具，为用户提供对三维重建数据的直观、易用的交互操作界面。* |  |
| ***Task2：利用光度立体视觉技术，复原纸币表面印刷字体的微观凸凹痕迹（可通过光照方向和强度的改变或表面法向量扰动，显示纸币的凹凸（Bumpmap）图或法线贴图（Normalmap），或直接生成并显示三维重建结果，以揭示肉眼难以直接观察的细微痕迹）***  光照方向分别为：  [-0.6238 -0.3953 -0.674 ]，  [ 0.0032 -0.7478 -0.6636]，  [ 0.6353 -0.3979 -0.6616]  [ 0.6257 0.2996 -0.7202]，  [ 0.0103 0.6676 -0.7444]，  [-0.6109 0.2910 -0.7362] | |

***Task3（选做）：利用所提供的视频序列（******RGB光谱复用形成的彩色图像），应用光度立体视觉技术重建桡动脉处因脉搏搏动而产生的皮肤表面微观动态形变，并通过动态法线贴图或三维波形动画等形式实现动态可视化，以直观展示脉搏波的传播过程。***

光照方向为：

[-0.3098, 0.5032, 0.8067]

[-0.3003, -0.5316, 0.7920]

[0.35240, 0.0224, 0.9356]

******

1. **实验步骤**

***Task1：通过光度立体视觉恢复球面法向量***

# 读取图像数据

img\_files = [

    "Sphere/LMImage0.bmp",

    "Sphere/LMImage1.bmp",

    "Sphere/LMImage2.bmp",

    "Sphere/LMImage3.bmp",

    "Sphere/LMImage4.bmp",

    "Sphere/LMImage5.bmp",

]

images\_list = []

for f in img\_files:

    img = imread(f)

    img = img.astype(np.float64)

    # 归一化到 [0,1]

    img = img / img.max()

    images\_list.append(img)

# 将图像列表转换为 NumPy 数组

images = np.stack(images\_list, axis=0)   # (m, H, W)

m, H, W = images.shape

S = np.array([

    [ 0.5000,  0.0000,  0.8660],

    [ 0.2500,  0.4330,  0.8660],

    [-0.2500,  0.4330,  0.8660],

    [-0.5000,  0.0000,  0.8660],

    [-0.2500, -0.4330,  0.8660],

    [ 0.2500, -0.4330,  0.8660],

])  # shape: (6,3)

# 6 张图像取平均，再阈值

mean\_img = images.mean(axis=0)

mask = mean\_img > 0.05

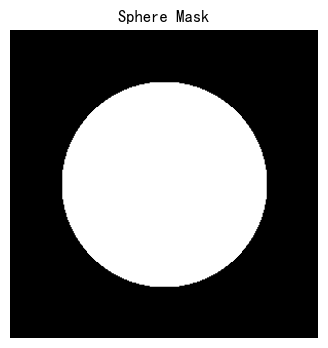
plt.figure(figsize=(4,4))

plt.imshow(mask, cmap="gray")

plt.title("Sphere Mask")

plt.axis("off")

plt.show()

******

def photometric\_stereo(images, S, mask):

    """

    images: (m, H, W)

    S: (m, 3)

    mask: (H, W)  True 表示在物体范围内

    """

    m, H, W = images.shape

    # 伪逆 (3, m)

    S\_pinv = np.linalg.pinv(S)

    # 把所有像素展平

    I = images.reshape(m, -1)  # (m, H\*W)

    # g = rho \* n,  (3, H\*W)

    g = S\_pinv @ I

    # 反照率 rho

    rho = np.linalg.norm(g, axis=0)  # (H\*W,)

    # 法向量 n

    n = g / (rho + 1e-8)             # (3, H\*W)

    # 还原成图像格式

    rho\_img = rho.reshape(H, W)

    n\_img = n.reshape(3, H, W).transpose(1, 2, 0)  # (H, W, 3)

    # mask 之外直接置零

    rho\_img[~mask] = 0

    n\_img[~mask] = 0

    return rho\_img, n\_img

rho\_6, n\_6 = photometric\_stereo(images, S, mask)

# 法向量可视化成 RGB

# 把 [-1,1] 映射到 [0,1]

N\_rgb = (n\_6 + 1.0) / 2.0

N\_rgb[~mask] = 0

fig, axes = plt.subplots(1, 2, figsize=(8,4))

axes[0].imshow(rho\_6, cmap="gray")

axes[0].set\_title("Albedo (rho)")

axes[0].axis("off")

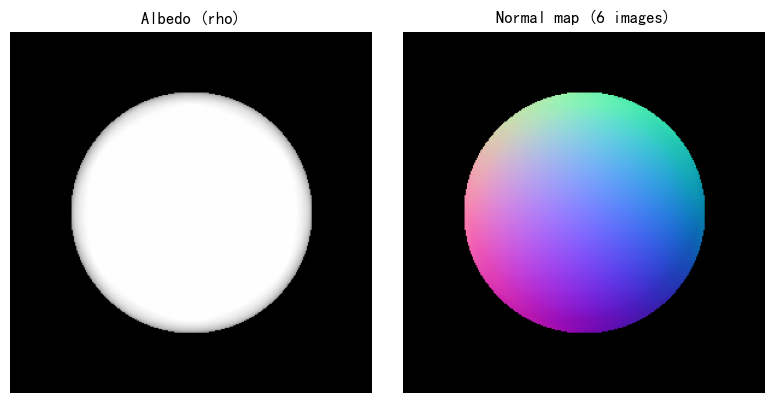
axes[1].imshow(N\_rgb)

axes[1].set\_title("Normal map (6 images)")

axes[1].axis("off")

plt.tight\_layout()

plt.show()

******

# 从 mask 估计球心和半径

ys, xs = np.nonzero(mask)

cx = xs.mean()

cy = ys.mean()

R = np.sqrt(((xs - cx)\*\*2 + (ys - cy)\*\*2).max())

print("cx, cy, R =", cx, cy, R)

# 构造网格坐标

x\_coords = np.arange(W)

y\_coords = np.arange(H)

X, Y = np.meshgrid(x\_coords, y\_coords)

Xc = X - cx

Yc = Y - cy

# 球面内的点满足 Xc^2 + Yc^2 + Z^2 = R^2

inside = Xc\*\*2 + Yc\*\*2 <= R\*\*2 + 1

Z = np.zeros\_like(Xc, dtype=np.float64)

Z[inside] = np.sqrt(R\*\*2 - Xc[inside]\*\*2 - Yc[inside]\*\*2)

# 真实法向

n\_true = np.stack([-Xc, -Yc, Z], axis=-1)

norms = np.linalg.norm(n\_true, axis=-1, keepdims=True) + 1e-8

n\_true = n\_true / norms

n\_true[~inside] = 0

N\_rgbT = (n\_true + 1.0) / 2.0

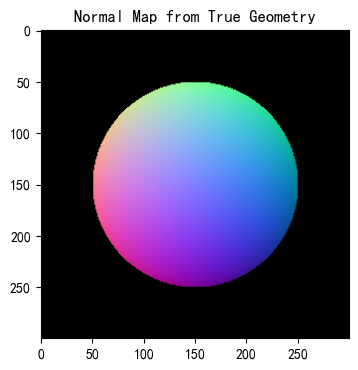
N\_rgbT[~mask] = 0

plt.figure(figsize=(12,4))

plt.subplot(1,1,1)

plt.imshow(N\_rgbT)

plt.title("Normal Map from True Geometry")



def normal\_angle\_error(n\_est, n\_true, mask):

    """

    n\_est, n\_true: (H, W, 3)

    mask: bool (H, W)

    返回平均角度误差(度)

    """

    dot = np.sum(n\_est \* n\_true, axis=-1)

    dot = np.clip(dot, -1.0, 1.0)

    theta = np.arccos(dot)  # radians

    theta\_deg = theta \* 180.0 / np.pi

    return theta\_deg[mask].mean()

err\_6 = normal\_angle\_error(n\_6, n\_true, mask)

print(f"6张图像的平均法向误差 = {err\_6:.3f} degrees")



errors\_num\_imgs = {}

for k in [3, 4, 5, 6]:

    imgs\_k = images[:k]       # (k, H, W)

    S\_k = S[:k]               # (k, 3)

    rho\_k, n\_k = photometric\_stereo(imgs\_k, S\_k, mask)

    err\_k = normal\_angle\_error(n\_k, n\_true, mask)

    errors\_num\_imgs[k] = err\_k

ks = sorted(errors\_num\_imgs.keys())

vals = [errors\_num\_imgs[k] for k in ks]

plt.figure(figsize=(4,3))

plt.plot(ks, vals, marker="o")

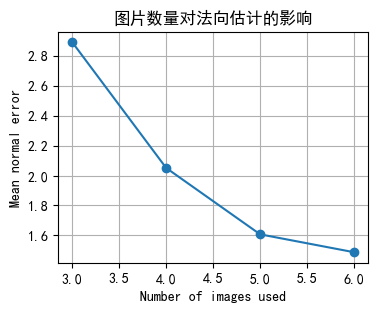
plt.xlabel("Number of images used")

plt.ylabel("Mean normal error")

plt.title("图片数量对法向估计的影响")

plt.grid(True)

plt.show()

******

sigmas = [0.00, 0.01, 0.02, 0.05, 0.10]  # 噪声标准差，可以自己改

errors\_noise = {}

for sigma in sigmas:

    noisy = images + np.random.normal(scale=sigma, size=images.shape)

    noisy = np.clip(noisy, 0.0, 1.0)

    rho\_n, n\_n = photometric\_stereo(noisy, S, mask)

    err\_n = normal\_angle\_error(n\_n, n\_true, mask)

    errors\_noise[sigma] = err\_n

sig\_list = sorted(errors\_noise.keys())

err\_list = [errors\_noise[s] for s in sig\_list]

plt.figure(figsize=(4,3))

plt.plot(sig\_list, err\_list, marker="o")

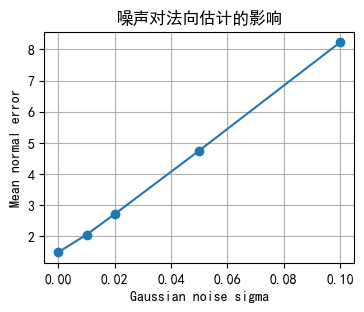
plt.xlabel("Gaussian noise sigma")

plt.ylabel("Mean normal error")

plt.title("噪声对法向估计的影响")

plt.grid(True)

plt.show()

******

# === 选做 1：法向量积分重建三维表面 ===

from mpl\_toolkits.mplot3d import Axes3D

# 保存一下真实球面高度，后面 GUI 要用

Z\_true = Z.copy()

def integrate\_normals(n, mask):

    """

    根据法向量场 n 和 mask，积分得到高度图 Z

    n:    (H, W, 3)，法向量

    mask: (H, W) 的 bool，True 表示物体区域

    """

    H, W, \_ = n.shape

    nx = n[..., 0]

    ny = n[..., 1]

    nz = n[..., 2]

    eps = 1e-6  # 防止除零

    p = nx / (nz + eps)   # dz/dx

    q = ny / (nz + eps)   # dz/dy

    # 物体外区域不参与积分

    p[~mask] = 0.0

    q[~mask] = 0.0

    # 沿 x 方向积分

    Zx = np.zeros((H, W), dtype=np.float64)

    Zx[:, 1:] = np.cumsum(p[:, :-1], axis=1)

    # 沿 y 方向积分

    Zy = np.zeros((H, W), dtype=np.float64)

    Zy[1:, :] = np.cumsum(q[:-1, :], axis=0)

    # 两种路径取平均，减小累积误差

    Z\_rec = (Zx + Zy) / 2.0

    # mask 外设为 NaN，画图时会被忽略

    Z\_rec[~mask] = np.nan

    return Z\_rec

# 用 photometric stereo 得到的法向量 n\_6 做重建

Z\_rec = integrate\_normals(n\_6, mask)

# 3D 可视化

fig = plt.figure(figsize=(6, 5))

ax = fig.add\_subplot(111, projection="3d")

ax.plot\_surface(X, Y, Z\_rec,

                rstride=3, cstride=3,

                cmap="viridis",

                linewidth=0, antialiased=True)

ax.set\_title("基于法向量积分重建的三维表面")

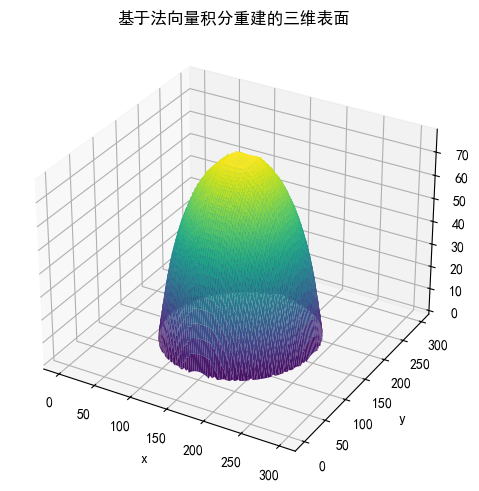
ax.set\_xlabel("x")

ax.set\_ylabel("y")

ax.set\_zlabel("z")

plt.tight\_layout()

plt.show()



# 选做 2

# 保存数据供 GUI 使用

import numpy as np

np.save("X.npy", X)

np.save("Y.npy", Y)

np.save("Z\_rec.npy", Z\_rec)

import matplotlib

# 使用 TkAgg 后端

matplotlib.use("TkAgg")

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D  # noqa: F401

from matplotlib.widgets import Slider

def main():

    # === 1. 读入在 notebook 里保存的数组 ===

    X = np.load("X.npy")

    Y = np.load("Y.npy")

    Z\_rec = np.load("Z\_rec.npy")

    # === 2. 创建图窗和 3D 坐标轴 ===

    fig = plt.figure(figsize=(8, 7))

    ax = fig.add\_subplot(111, projection="3d")

    # 初始视角

    init\_elev = 30

    init\_azim = 45

    surf = ax.plot\_surface(

        X, Y, Z\_rec,

        cmap="viridis",

        linewidth=0, antialiased=True

    )

    ax.view\_init(elev=init\_elev, azim=init\_azim)

    ax.set\_title("3D Surface Viewer (TkAgg, with Sliders)")

    ax.set\_xlabel("X")

    ax.set\_ylabel("Y")

    ax.set\_zlabel("Z")

    # === 3. 在图窗下方预留区域放滑块 ===

    # [left, bottom, width, height]，都是 0~1 的相对坐标

    fig.subplots\_adjust(left=0.1, right=0.95, bottom=0.20)

    ax\_elev = fig.add\_axes([0.15, 0.08, 0.70, 0.03])

    ax\_azim = fig.add\_axes([0.15, 0.03, 0.70, 0.03])

    s\_elev = Slider(

        ax=ax\_elev,

        label="Elevation",

        valmin=0,

        valmax=90,

        valinit=init\_elev

    )

    s\_azim = Slider(

        ax=ax\_azim,

        label="Azimuth",

        valmin=0,

        valmax=180,

        valinit=init\_azim

    )

    # === 4. 定义当滑块变化时的回调函数 ===

    def update(val):

        elev = s\_elev.val

        azim = s\_azim.val

        ax.view\_init(elev=elev, azim=azim)

        fig.canvas.draw\_idle()

    s\_elev.on\_changed(update)

    s\_azim.on\_changed(update)

    # === 5. 显示窗口===

    plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

1. **讨论及结论**