University of Ljubljana, Faculty of Electrical Engineering
Robot Vision (RV)

May 17/18, 2016

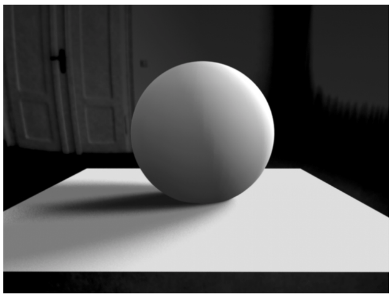
Exercise 9: Reconstruction of 3D Shapes

Created by: Žiga Špiclin | http://lit.fe.uni-lj.si/RV | Homework deadline: May 24/25, 2016

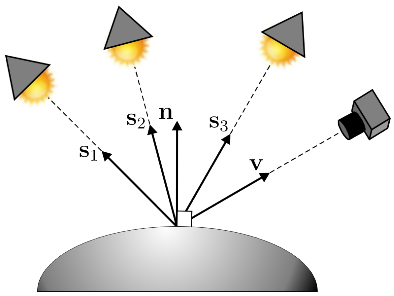
Instructions

Reconstruction of 3D shapes employs either one or more 2D images of an object of interest, where an image or images are acquired from a different view or under different light conditions, in order to find an optimal 3D shape of the object that best corresponds to the acquired image(s). **Fotometric stereo** is a reconstruction method, which extracts the 3D shape information based on the appearance of shadows on the object of interest. The shadows are cast by illuminating the object from different directions (e.g. $\boldsymbol{s}_1$,$\boldsymbol{s}_2$,$\boldsymbol{s}_3$), while the images are acquired from a fixed viewpoint ($\boldsymbol{v}$). Based on the acquired image an estimate of the 3D surface normal $\boldsymbol{n}$ is obtained in each image location (x, y) and by integrating these normals with respect to some reference point we may obtain a reconstruction of the 3D object shape.

Information about 3D shape based on light-object interaction



Fotometric stereo



Under the assumption that light-surface interaction follows a model of diffuse reflection, then the intensity of each pixel may be related to the shape of the observed surface. Namely, we employ the **Lambert's model of diffuse reflectance**:

$$I(x, y) = \frac{kc}{\pi} \rho_i \cos\theta_i = \frac{kc}{\pi} \rho_i \, \boldsymbol{s} \circ \boldsymbol{n} \,,$$

where k in the intensity of the light source, c the constant of the optical system and ρ the surface albedo or reflectance. Under the assumption that $\frac{kc}{\pi} = 1$ the j-th light source ($j = 1, \ldots, N$) is related to the intensity of a pixel according to the following equation: $I_j(x, y) = \rho_i \cos\theta_i = \rho_i \, \boldsymbol{s}_j \circ \boldsymbol{n}$. To estimate the surface normal $\boldsymbol{n}$ in each pixel (x, y) we need at least three light sources ($N = 3$), which illuminated the object from different directions ($\boldsymbol{s}_1$,$\boldsymbol{s}_2$,$\boldsymbol{s}_3$), so as to obtain a system of linear equation.

System of equations for three light sources: | System of equations in matrix form: | Solution:

$$\begin{aligned} I_1 &= \rho\, \boldsymbol{s}_1^{\mathrm{T}} \cdot \boldsymbol{n} \\ I_2 &= \rho\, \boldsymbol{s}_2^{\mathrm{T}} \cdot \boldsymbol{n} \\ I_3 &= \rho\, \boldsymbol{s}_3^{\mathrm{T}} \cdot \boldsymbol{n} \end{aligned} \quad \Longrightarrow \quad \underbrace{\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix}}_{\boldsymbol{I}_{(3\times1)}} = \underbrace{\begin{bmatrix} \boldsymbol{s}_1^{\mathrm{T}} \\ \boldsymbol{s}_2^{\mathrm{T}} \\ \boldsymbol{s}_3^{\mathrm{T}} \end{bmatrix}}_{\boldsymbol{S}_{(3\times3)}} \underbrace{\rho\boldsymbol{n}}_{\tilde{\boldsymbol{n}}_{(3\times1)}} \quad \Longrightarrow \quad \begin{aligned} \tilde{\boldsymbol{n}} &= \boldsymbol{S}^{-1}\boldsymbol{I} \\ \rho &= \|\tilde{\boldsymbol{n}}\| \\ \boldsymbol{n} &= \tilde{\boldsymbol{n}}/\|\tilde{\boldsymbol{n}}\| = \|\tilde{\boldsymbol{n}}\|/\rho \end{aligned}$$

The described method yields an estimate of surface normal $\boldsymbol{n}$ in each pixel (x, y). The estimate is generally more robust if more light sources from mutually different directions $\boldsymbol{s}_j$, $N > 3$ are used. Through interaction with the object a certain j-th light direction may produce low intensity in a region, thus the estimate of surface normal might be less reliable in such a region. This may be improved by weighting each of the equations by the intensity $I_j = I_j(x, y)$ of a pixel from each light direction $\boldsymbol{s}_j$.

System of equation for N light sources:

$$I_1^2 = I_1\left(\rho\, \boldsymbol{s}_1^{\mathrm{T}} \cdot \boldsymbol{n}\right)$$
$$\vdots$$
$$I_N^2 = I_N\left(\rho\, \boldsymbol{s}_N^{\mathrm{T}} \cdot \boldsymbol{n}\right)$$

$$\Longrightarrow$$

System of equations in matrix form:

$$\underbrace{\begin{bmatrix} I_1^2 \\ \vdots \\ I_N^2 \end{bmatrix}}_{\boldsymbol{I}_{(N\times 1)}} = \underbrace{\begin{bmatrix} I_1\, \boldsymbol{s}_1^{\mathrm{T}} \\ \vdots \\ I_N\, \boldsymbol{s}_N^{\mathrm{T}} \end{bmatrix}}_{\boldsymbol{S}_{(N\times 3)}} \underbrace{\rho \boldsymbol{n}}_{\tilde{\boldsymbol{n}}_{(3\times 1)}} \Longrightarrow$$

Solution:

$$\boldsymbol{I} = \boldsymbol{S}\,\tilde{\boldsymbol{n}}$$
$$\boldsymbol{S}^{\mathrm{T}}\boldsymbol{I} = \boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}\,\tilde{\boldsymbol{n}}$$
$$\tilde{\boldsymbol{n}} = \left(\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}\right)^{-1}\boldsymbol{S}^{\mathrm{T}}\,\boldsymbol{I}$$

For each pixel (x, y) the linear system of equations is formed and its solution yields the surface normal $\boldsymbol{n}(x, y)$, which can be further employed to reconstruct the 3D object's shape in several different ways, for instance, in the form of a depth image $z(x, y)$. The depth image $z(x, y)$ can be computed by analyzing partial derivatives of the surface, i.e. $\partial z/\partial x$ and $\partial z/\partial y$. At each pixel (x, y) these two derivatives yield a pair of vectors that are orthogonal to the surface normal $\boldsymbol{n}$.

Based on all the equations for an image of size $X \times Y$ we can form a linear system $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{p}$, which can be solved for $z(x, y)$. Matrix $\boldsymbol{M}$ has dimensions $(2XY \times XY)$ and vector $\boldsymbol{p}$ has $(2XY \times 1)$ elements. Matrix $\boldsymbol{M}$ is relatively large, but sparse as most of the values are zero. To lower the memory requirements we will use a Python library package `scipy.sparse` to define a sparse matrix with function `csc_matrix()`, whereas the values may be inserted into the matrix simply by indexing the rows and columns. The solution of the linear system of equations can be obtained using function `lsqr()` in Python library package `scipy.sparse.linalg`.

Partial surface derivatives are vectors orthogonal to the surface normal $\boldsymbol{n}$:

$$\boldsymbol{z}_x = \partial z/\partial x$$
$$\boldsymbol{z}_y = \partial z/\partial y$$
$$\rightarrow \boldsymbol{n} \circ \boldsymbol{z}_x = 0$$
$$\rightarrow \boldsymbol{n} \circ \boldsymbol{z}_y = 0$$

$$\Rightarrow$$

Partial derivatives can be computed by finite element method for each point (x, y) so as to get two linear equations:

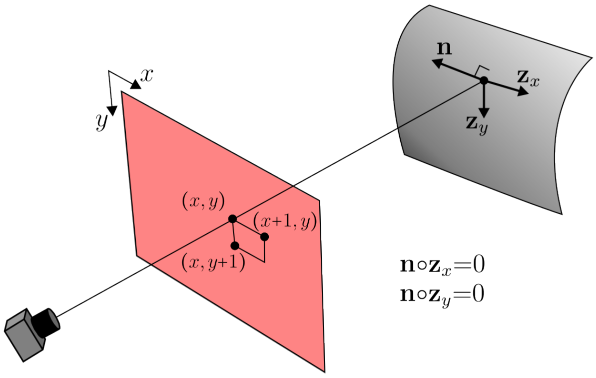
$$\begin{aligned} \boldsymbol{z}_x = \ & (x+1, y, z(x+1, y)) - (x, y, z(x, y)) \\ & (1, 0, z(x+1, y) - z(x, y)) \\ \boldsymbol{z}_y = \ & (x, y+1, z(x, y+1)) - (x, y, z(x, y)) \\ & (0, 1, z(x, y+1) - z(x, y)) \\ \rightarrow \ & n_x + n_z\left(z(x+1, y) - z(x, y)\right) = 0 \\ \rightarrow \ & n_y + n_z\left(z(x, y+1) - z(x, y)\right) = 0 \end{aligned}$$

$$\Rightarrow$$

The linear equations across all pixels (x, y) form a linear system, which can be solved for $z(x, y)$:

$$\boldsymbol{M} z = \boldsymbol{p}$$
$$\boldsymbol{M} \in \{0, n_z, -n_z\}$$
$$\boldsymbol{p} \in \{n_x, n_y\}$$
$$\rightarrow \boldsymbol{z} = (M^{\mathrm{T}}M)^{-1}M^{\mathrm{T}}\boldsymbol{p}$$

Reconstruction of 3D surface by the analysis of partial differentials



Materials for the exercise are given in file `owl.zip`, which contain a file `lights.txt` with the directions of incident light with respect to the object, images of the object `owl.k.tif`, where $k = 0, \ldots, 11$ and corresponding object mask `owl.mask.tif`.

1. Write a function for reading a text file containing light direction vectors:

```python
def readTxt( iFileName ):
        return oLightDir
```

where parameter `iFileName` represents the filename (i.e. `'lights.txt'`). The file can be opened and closed with respective functions `open()` and `close()`, while each line can be read using function `readline()`. To separate and convert the string of numbers into numerical type you can use functions `split()` and `float()`, respectively. Function should return an array `oLightDir`, which for a given file `'lights.txt'` should have dimensions 12×3.

2. Create a variable of type `list` and load all the color images `owl.k.tif` in the order `k=0,...,11`. An image with index `k` corresponds to a light direction given in array `oLightDir` with index `[k,:]`. Load the mask `owl.mask.tif` in a variable of type `numpy.array`.

   - Compute the mean color image, e.g. `iMeanImage`.
   - Convert all color images into grayscale images and form a `numpy.array`, e.g. `iImages`, which has dimensions $Y \times X \times 12$, where X, Y correspond to dimensions of 2D images.
   - Convert the mask image into grayscale, e.g. as variable `iMask`.

3. Write a function that computes the object's surface normals $\boldsymbol{n}$ from three images:

```python
def computeNormals( iImages, iMask, iLightDir ):
        return oNormals
```

   where `iImages` is an array of dimensions $Y \times X \times 3$ (X, Y correspond to dimensions of 2D images), `iMask` is the object's mask and `iLightDir` a 3×3 array of light directions corresponding to the images in variable `iImages`. Function should return a 3D array of non-normalized surface normals $\tilde{\boldsymbol{n}}$ of dimensions $Y \times X \times 3$ in variable `oNormals`. Verify the function using three arbitrary input images ($N = 3$), however, you should make sure to select the input images and form a correct array of corresponding light directions $\boldsymbol{s}_j$.

   Display the mean color image `iMeanImage` and plot the computed surface normals in the form of a vector field using function `quiver()` in Python library package `matplotlib.pyplot`. Verify that the surface normals are sensible based on the given image of a 3D shape.

4. Write a function that computes the albedo or surface reflection ρ based on non-normalized surface normals $\tilde{\boldsymbol{n}}$:

```python
def computeAlbedo( iNormals, iMask ):
        return oAlbedo
```

   where `iNormals` is an array of dimensions $Y \times X \times 3$ (X, Y correspond to 2D image dimensions, while each $\tilde{\boldsymbol{n}}$ has three components), `iMask` is the image of object mask with dimensions $Y \times X$. Function should return a 2D array or image of albedo ρ with dimensions $Y \times X$ in variable `oAlbedo`. Ensure that the values of `oAlbedo` will be in the range $[0, 1]$.

   Verify the function based on the surface normals computed in the previous assignment.

5. Write a function that reconstructs a 3D shape of an object by computing a depth value $z(x, y)$ based on the analysis of partial surface derivatives and a solution of a corresponding linear system of equations:

```python
def computeDepthLinSys( iNormals, iMask ):
        return oDepth
```

   where `iNormals` is an array of dimensions $Y \times X \times 3$ and `iMask` the object's mask with dimensions $Y \times X$. Ensure that surface normals `iNormals` are normalized, i.e. $\boldsymbol{n}$. To solve this assignment you need to form a linear system $\boldsymbol{M}\boldsymbol{z} = \boldsymbol{p}$, where matrix $\boldsymbol{M}$ (dimensions $2XY \times XY$) is initialized by function `csc_matrix()` in library package `scipy.sparse`, while individual matrix elements are inserted through indexing. The solution of the linear system is obtained using function `lsqr()` in library package `scipy.sparse.linalg`. Function should return a 2D array of depth $z(x, y)$ with dimensions $Y \times X$ in variable `oDepth`.

   Verify the function based on the surface normals computed in the first assignment. Display the reconstructed depth as a 3D surface using function `plot_surface()` in Python library package `mpl_toolkits.mplot3d` and apply a color texture to the surface in the form of mean color image `iMeanImage`.

Homework Assignments

Homework report in the form of a Python script entitled `NameSurname_Exercise9.py` should execute the requested computations and function calls and display requested figures and/or graphs. It is your responsibility to load library packages and provide supporting scripts such that the script is fully functional and that your results are reproducible. The code should execute in a block-wise manner (e.g. `#%% Assignment 1`), one block per each assignment, while the answers to questions should be written in the corresponding block in the form of a comment (e.g. `# Answer: ...`).

1. Display the surface normals $\tilde{\boldsymbol{n}}$ as a color image, where you should convert the components of normals to the intensity range `[0, 255]`.

2. Revise the function `computeNormals()` such that an arbitrary number of input images N may be used to reconstruct the surface normals $\boldsymbol{n}$. The input parameter `iImages` should be an array of dimension $Y \times X \times N$ (X, Y are the dimension of the 2D image, N is the number of images with different light directions), `iMask` is a binary mask of the object with dimensions $Y \times X$ and `iLightDir` an $N \times 3$ array encoding the light directions that corresponding to images in variable `iImages`. Function should return a 3D array of non-normalized normals $\tilde{\boldsymbol{n}}$ of dimensions $Y \times X \times 3$ in variable `oNormals`. Verify the function by using all input images ($N = 12$) and their corresponding light directions $\boldsymbol{s}_j$.

   Display the mean color image `iMeanImage` and plot the surface normals in the form of vector field using function `quiver()`. Please elaborate in which part of the object you observe an improvement in the direction of the normals compared to those obtained using only three images.

3. Expand the function `computeNormals()` such that the surface normals $\boldsymbol{n}$ are computed in a robust manner using the grayscale weighting, i.e. the basic equation used to form a linear system has the form $I_j^2 = I_j\,(\rho\, \boldsymbol{s}_j^{\mathrm{T}} \cdot \boldsymbol{n})$. Verify the function by using all input images ($N = 12$) and their corresponding light directions $\boldsymbol{s}_j$.

   Display the mean color image `iMeanImage` and plot the surface normals in the form of vector field using function `quiver()`. Please elaborate in which part of the object you observe an improvement in the direction of the normals compared to those obtained using only three images.

4. Use the revised function `computeNormals()` to reconstruct a 3D object surface using all the input images ($N = 12$) to compute the surface normals and then compute $z(x, y)$ using `computeDepthLinSys()`. Display the reconstructed 3D surface and apply color texture in the form of mean color image `iMeanImage` to the surface.