Extra Credit Project

Out: 4 / 28 / 2021
Due: 5 / 12 / 2021 (deadline: 11:55PM)

Extra credit: This project is for extra credit and is therefore optional. You can earn up to 5% extra credit which will be added to your overall percentage in the course.

Late submissions: Late submissions result in 10% deduction for each day. The assignment will no longer be accepted 3 days after the deadline.

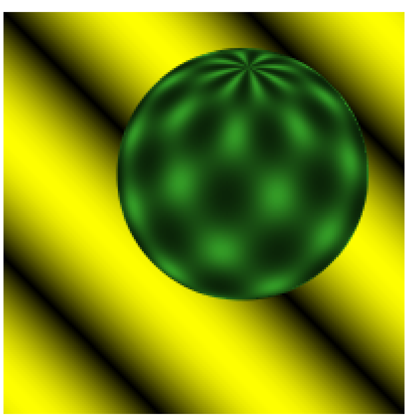
Please read the instructions carefully. Note: we will not be running code. Rather, we will check your code to make sure your implementation is your own, and it matches your results. Your grade is primarily based on your written report. This means going beyond just showing results. You should produce a standalone lab report, describing results in enough detail for someone else to follow. Please submit a single PDF/HTML with all code included as an appendix.

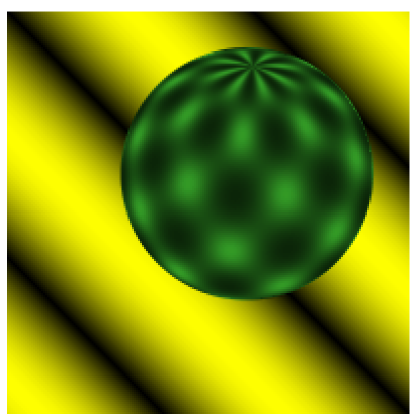
A) Optical Flow

In this project, you will implement the Lucas-Kanade and Horn-Schunck optical flow methods. Both methods are based on spatial and temporal derivatives, differing mainly in the optimization approach.

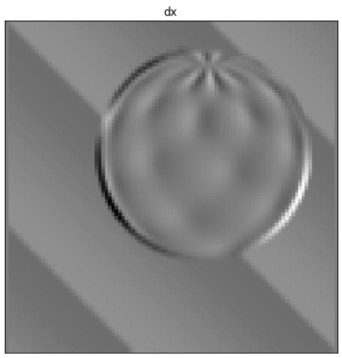
Implementation:

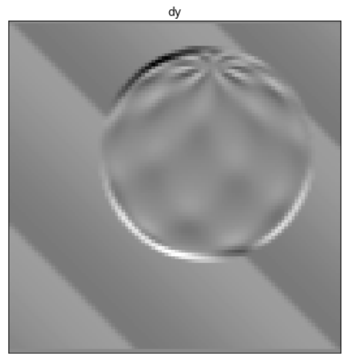
1. Assume two images with the same dimensions, e.g. im1 = sphere0.png and im2 = sphere1.png.



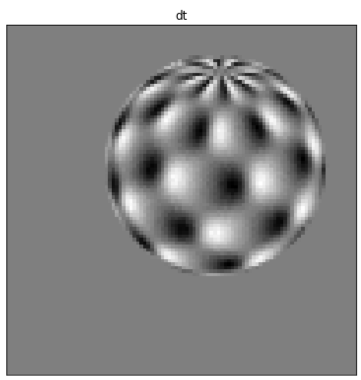


2. Compute <u>spatial partial derivative images I_x and I_y for im1</u> using any method of your choice.





3. Compute temporal partial derivative I_t. This can be approximated by image difference im2 - im1.



Implement Lucas-Kanade

This method solves for optical flow by assuming constant flow within a neighborhood (patch), centered around the current pixel *(i,j)*. Using this assumption, a linear system is constructed and solved to find $u_{i,j}$ and $v_{i,j}$. This means you must repeat this process for every pixel in the image.

Consider using a neighborhood size of 5x5. This results in the linear system shown below, with a matrix consisting of spatial partial derivatives and a vector of temporal partial derivatives at all points in the neighborhood p_i. Solve this system of equations via linear least squares to find $u_{i,j}$ and $v_{i,j}$.

<u>Note, this is not the value of u and v for the entire neighborhood, only for the pixel *(i,j)*. Repeat this process for every pixel in the image.</u> You may ignore the boundary of the image where you cannot sample a full neighborhood.

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_{25}) & I_y(p_{25}) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_{25}) \end{bmatrix}$$

Matrix form of the system of eqns

A	x	b
25 x 2	2 x 1	25 x 1
known	<u>unknown</u>	known

Implement Horn-Schunck

This method solves for optical flow by imposing a smoothness penalty on the optical flow field, and uses gradient descent to iteratively estimate optical flow. The cost function to minimize is composed of two parts:

<u>Smoothness on optical flow:</u>

$$E_s(i,j) = \frac{1}{4}\Bigg[(u_{i,j} - u_{i+1,j})^2 + (u_{i,j} - u_{i,j+1})^2 + (v_{i,j} - v_{i+1,j})^2 + (v_{i,j} - v_{i,j+1})^2\Bigg]$$

<u>Brightness constancy:</u>

$$E_d(i,j) = [I_x(i,j)u_{i,j} + I_y(i,j)v_{i,j} + I_t(i,j)]^2$$

<u>The overall cost function:</u>

$$E(\mathbf{u},\mathbf{v}) = \sum_{i,j}\Big[E_d(i,j) + \gamma E_s(i,j)\Big]$$

Below is a pseudocode algorithm for Horn-Schunck optical flow:

```
# Initialize u and v to zeros

while not converged:

    # Loop over all pixels
    for i in rows:
        for j in cols:

            # Compute ubar and vbar
            ubar = ...
            vbar = ...

            # Update u and v
            u[i,j] = ubar - ...
            v[i,j] = vbar - ...
```

Where the update equations are (for iteration k):

$$\bar{u}_{i,j} = \frac{1}{4}\Big[u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1}\Big] \quad \bar{v}_{i,j} = \frac{1}{4}\Big[v_{i+1,j} + v_{i-1,j} + v_{i,j+1} + v_{i,j-1}\Big]$$

$$u_{i,j}^{k+1} = \bar{u}_{i,j}^{k} - \frac{I_x(i,j)(I_x(i,j)\bar{u}_{i,j}^{k} + I_y(i,j)\bar{v}_{i,j}^{k} + I_t(i,j))}{\gamma^2 + I_x(i,j)^2 + I_y(i,j)^2}$$

$$v_{i,j}^{k+1} = \bar{v}_{i,j}^{k} - \frac{I_y(i,j)(I_x(i,j)\bar{u}_{i,j}^{k} + I_y(i,j)\bar{v}_{i,j}^{k} + I_t(i,j))}{\gamma^2 + I_x(i,j)^2 + I_y(i,j)^2}$$

You may check for convergence by computing the overall cost and checking that it doesn't change much between iterations, or you may just run the procedure for a certain number of iterations until you are satisfied with the results. It may take many iterations.

Experiments:

Synthetic Sphere Images

1. Using sphere0.png and sphere1.png, explore the **Lucas-Kanade** method using a neighborhood size of **3**, **5, 11, and 21**.
    - For each neighborhood size, plot optical flow as a vector field **and** the magnitude of the optical flow field at every pixel.
    - Describe the impact of the neighborhood size. Do you conclude that it plays a large role in the estimation of the optical flow field?

2. Using sphere0.png and sphere1.png, explore the **Horn-Schunck** method using values of λ = **100, 10, 1, and 0.1**.
    - For each value of λ, plot optical flow as a vector field **and** the magnitude of the optical flow field at every pixel.
    - Describe the impact of λ. Do you conclude that it plays a large role in the estimation of the optical flow field?

Real Traffic Images

3. Using traffic0.png and traffic1.png, get the best results you can using the **Lucas-Kanade** method.
    - Show the best result you were able to obtain along with any parameter values used. Plot optical flow as a vector field **and** the magnitude of the optical flow field at every pixel.
    - Discuss the process of obtaining good results, i.e. was it easy/difficult?

4. Using traffic0.png and traffic1.png, get the best results you can using the **Horn-Schunck** method.
    - Show the best result you were able to obtain along with any parameter values used. Plot optical flow as a vector field **and** the magnitude of the optical flow field at every pixel.
    - Discuss the process of obtaining good results, i.e. was it easy/difficult?

Summary

5. Give your overall thoughts of the two methods:
    - Do each have certain strengths and weaknesses, or is there one method that you find to be superior?
    - Compare the methods in terms of difficulty in choosing parameters.
    - Compare the methods (qualitatively) in terms of run time.
    - Were both methods equally challenging to implement. Did you find one to be easier?

Helpful hints:

- **Plotting vectors over an image**

```
# Subsample the vector field to make it less dense
subsample = 6
sub_u = u[0:rows:subsample, 0:cols:subsample]
sub_v = v[0:rows:subsample, 0:cols:subsample]

xc = np.linspace(0, cols, sub_u.shape[1])
yc = np.linspace(0, rows, sub_u.shape[0])

# Locations of the vectors
xv, yv = np.meshgrid(xc, yc)

fig1 = plt.figure(figsize = (14,7))

plt.imshow(im1,cmap = 'gray')
plt.title('Optical Flow'), plt.xticks([]), plt.yticks([])

# Plot the vectors
plt.quiver(xv, yv, sub_u, sub_v, color='y')
```

- **Blurring before computing derivatives**

  You may find you get better (or worse) results if you blur before you compute derivatives.

- **Downsampling input images**

  You may find you get better (or worse) results if you downsample the input images.

- **Solving Lucas-Kanade via linear least squares instead of exact solution**

  You saw from lecture that you can construct an exact linear system via A^TA and A^Tb. However, this matrix A^TA is problematic and will cause solver errors for many pixels in the image. The real solution involves heuristics about the eingenvalues of A^TA. Solving via linear least squares avoids having to deal with this issue.