Matthew England
2/26/2016

Computer Vision CS8690/ECE8690
Assignment 2: Lucas-Kanade Optical Flow Algorithm

Introduction:

Motion tracking is an important field of study within computer vision and is advancing everyday to tackle complex motion problems. Motion tracking allows researchers to track objects movement over time between different images. Various methods have been proposed and successfully implemented in this report I discuss the implementation and testing of the Lucas-Kanade Optical Flow Algorithm. The Lucas-Kanade Optical Flow algorithm invented in 1984 gave researchers at the time the ability to track objects over time through image chip frames. However, Lucas-Kanade Optical Flow does have limitations in used, the environment intensity must remain constant, which is impossible unless in an academic setting. Thus, improved algorithms used Harris Corner Detector and Multiscale Feature Tracking. That will discussed in the next report.

Implementation:

I used the equation below as my starting point to create motion tracking overtime. This is called the optical flow estimation equation. It's pixel intensity tracking of every pixel in an image overtime.

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

Now my implementation starts with loading both images without any conversion and then converting them into grayscale of double precision for later derivative calculations. After gray scale conversion I calculate the derivatives of the x,y, and time of the two images passed at the command line.

I calculate the partial derivatives and sums. After those sum and derivative calculations I must solve the least squares method from the equation below.

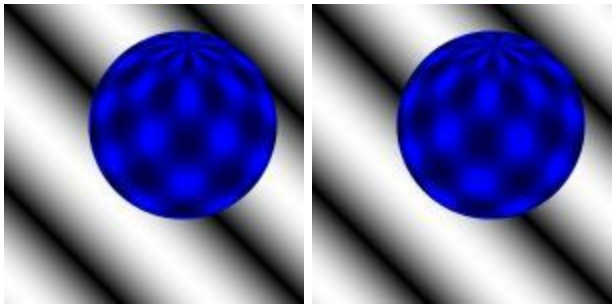
$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \mathbf{u} + \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} = 0$$

Matthew England
2/26/2016

Using a window radius sum pixels intensities around every pixel to approximate the velocity of the image. We have two unknowns that needed to be solved, thus I solve for **u** and then solve for **v**. After that I can now calculate the movement of the pixels in the image since I have **u** and **v** I can find the x and y coordinates to plot the vector movement.

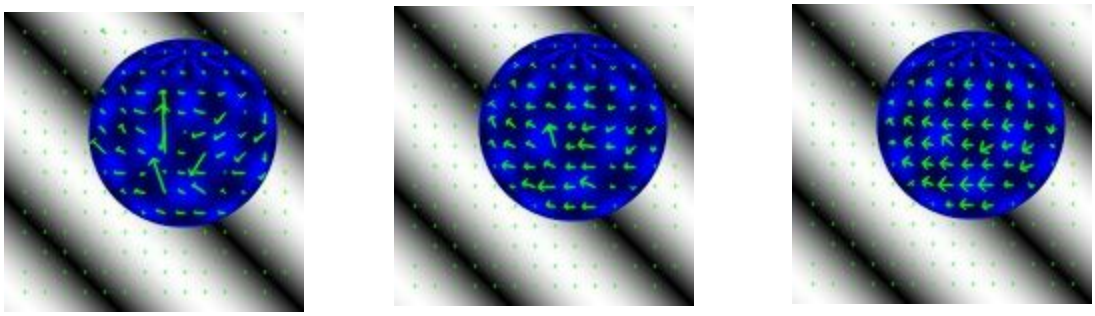
Results:

Sphere 1 **Sphere 2**

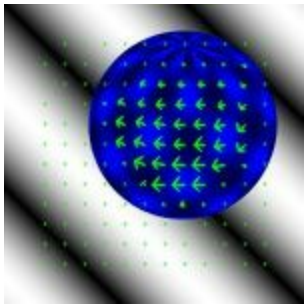


Sphere 2 vs Sphere 1

Window Radius = 1 **Window Radius = 3** **Window Radius = 5**



Window Radius = 11



Matthew England
2/26/2016

Basketball and People Movement

Window Radius = 3

Window Radius = 5

Matthew England
2/26/2016

Window Radius = 20

Conclusion:

As you increase the window radius too much you begin to see a degradation in pixel movement, this makes sense, since you are expanding the number of pixels intensity affecting the trueness of the vector. Basically you are losing precision in your pixel movement approximation. A too small window size does poorly because you need more than just the bare minimum of neighbors as shown the sphere window radius 1 and 3. Each image will have a sweet spot for a window radius, my best window radius for the spheres is 5, thus 121 neighbors are selected for the movement approximation. I then tested my algorithm on a pair sample from the http://vision.middlebury.edu/flow/data/ and picked the basketball which contains a lot of motion. As one can see again too many pixels for vector approximations causes degradation of the accuracy of the vectors. I believe that the picked window radius of 5 gave me best results for these two images. Future work is needed to get nicer looking lines. I can't get the curvature of the arrows to look nicely. My lines are straight and don't show the flow smoothly across a sphere example. Also my results look weird on the buildings, maybe it was the images I choose, but it looked to rigid between to frames.