GPU Programming in Computer Vision

Ramakrishna Nanjundaiah
Sadiq Huq

General Purpose GPU vs CPU

CPU

- Perform very quickly one or two tasks at a time
- Few ALU's
- Poor memory interface for large data sets

GPU

- Perform large number of tasks, relatively quick.
- Dozens or even hunderds of ALU's
- Massive parallel memory interface, typically 10x faster than CPU inteface

Source : www.supercomputingblog.com

Compute Unified Device Architecture

- NVIDIA ALU´s are fully programmable
- The type of applications suited for CUDA is limited
- Enables implementation of highly parallel algorithms
- Several hundreds of threads run the same kernel
- Best suited for number crunching
- Flexible choice of language
- Use of double should be avoided as only one 64-bit floating point units for every sixteen 32-bit unit.

What is Optical Flow?

- Given I1 , I2 : Ω → R, we find a vector field u : Ω → R2 such that

$$I(x+\delta x, y+, t+\delta t) = I(x,y,t) + \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t + H.O.T$$

$$Reduces\ to\ : \frac{\partial I}{\partial x}\delta x + \frac{\partial I}{\partial y}\delta y + \frac{\partial I}{\partial t}\delta t = 0$$

$$I_x U + I_y V = -I_t$$

$$Reduces\ to\ : \frac{\partial I}{\partial x}U + \frac{\partial I}{\partial y}V + \frac{\partial I}{\partial t} = 0\ (Aperture\ Problem)$$

$$\bigtriangledown I^T.\vec{V} = -I_t$$

Solution Methods

- Horn - Schunck Method
- Buxton - Buxton Method
- Black - Jepson Method
- General Variational Methods

Horn and Schunck Method

- Invloves imposing global constraint of smoothness for solving the aperture problem
- Solves the energy equation:

$$E(u) = \int (\nabla I^T u + I_t)^2 + \lambda \left|\nabla u\right|^2 \; dx$$

- The Euler – Lagrange equation is solved for the above variation problem

$$I_x I_y + I_y^2 u_2 + I_y I_t - \lambda \Delta u_2 = 0$$

$$I_x^2 u_1 + I_x I_y u_2 + I_x I_t - \lambda \Delta u_1 = 0$$

Horn - Schunck Method: Implementation

- The iteration must be repeated once the neighbors are updated

$$u^{k+1} = \overline{u}^k - \frac{I_x(I_x\overline{u}^k + I_y\overline{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2}$$

$$v^{k+1} = \overline{v}^k - \frac{I_y(I_x\overline{u}^k + I_y\overline{v}^k + I_t)}{\alpha^2 + I_x^2 + I_y^2}$$

- The system of equations is solved with Gauss - Seidel or Jordan method using over-relaxation techniques
- High density of flow vectors but more sensitive to noise

Advanced Penalty Function

- The energy functional is as follows:

$$E(u) = \int \phi_D((\nabla I^T + I_t)^2) + \lambda \phi_r(|\nabla u|^2)$$

$$\phi(\mathbf{s}^2) = \sqrt{\epsilon + \mathbf{s}^2}$$

- Discretization using finite difference scheme leads to sparse LSE that can be solved with SOR

- The penalty function yields better robustness against noise

Warping

- Warping methods are capable of dealing with large displacements (coarse to fine strategy)

$$E(u) = \int_{\Omega} \Phi_D((I_2(\mathbf{x}+u(\mathbf{x})) - I_1(\mathbf{x}))^2) + \lambda \Phi_R(|\nabla u|)^2 \quad \mathrm{d}\mathbf{x}$$

$$E(du^k) = \int_{\Omega} \Phi_D((\nabla I^k du + I_t^k)^2) + \lambda \Phi_R(|\nabla u^k + \nabla du^k|^2) \quad \mathrm{d}\mathbf{x}$$

Warping: Implementation

- Solving for the Euler – Lagrange equation

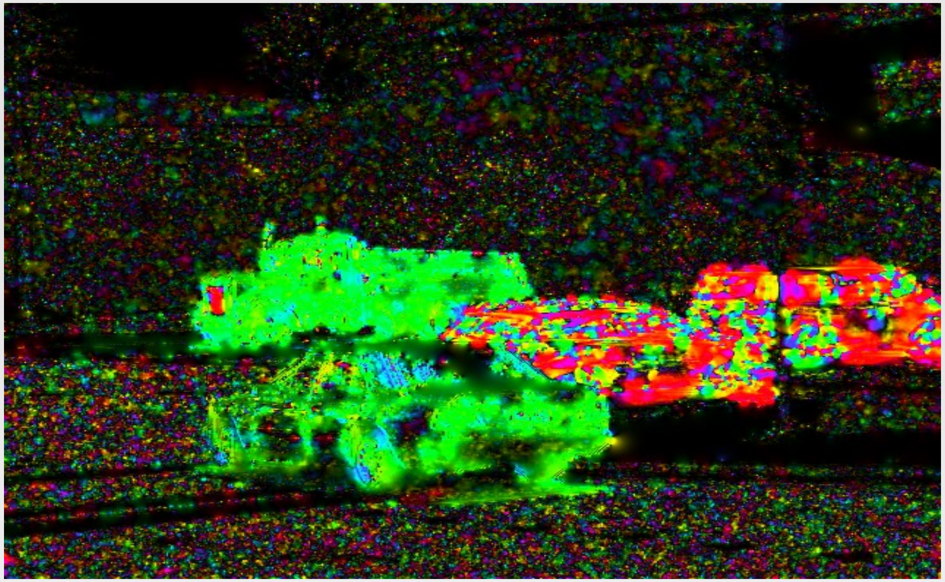
$$\begin{aligned} 0 &= \Phi'_D \cdot ({I_x^k}^2 du_1 + I_x^k I_y^k du_2 + I_x^k I_t^k) - \lambda \mathrm{div}(\Phi'_R \cdot \nabla u_1^k) - \lambda \mathrm{div}(\Phi'_R \cdot \nabla du_1^k) \\ 0 &= \Phi'_D \cdot (I_x^k I_y^k du_1 + {I_y^k}^2 du_2 + I_y^k I_t^k) - \lambda \mathrm{div}(\Phi'_R \cdot \nabla u_2^k) - \lambda \mathrm{div}(\Phi'_R \cdot \nabla du_2^k) \end{aligned}$$

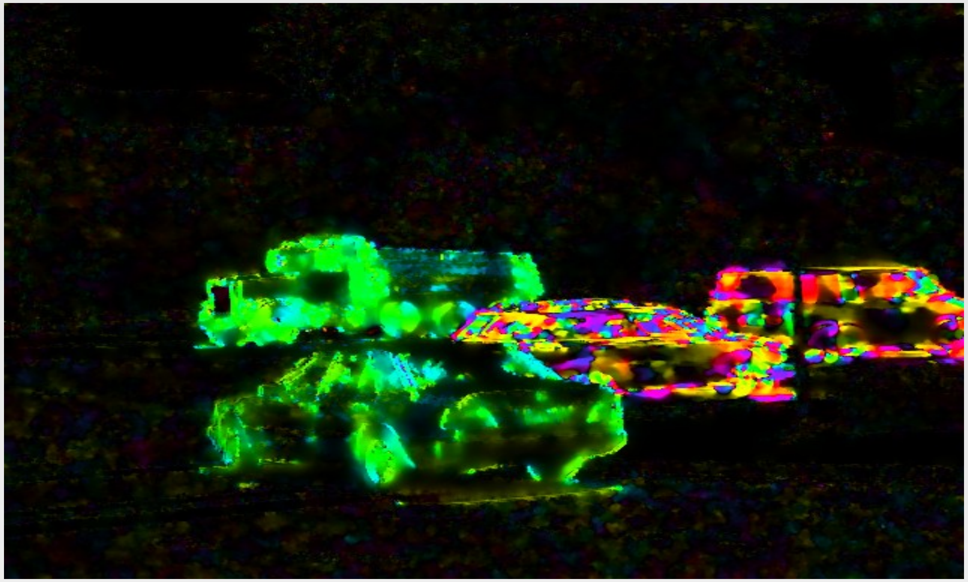
- The the results obtained are updated such that:

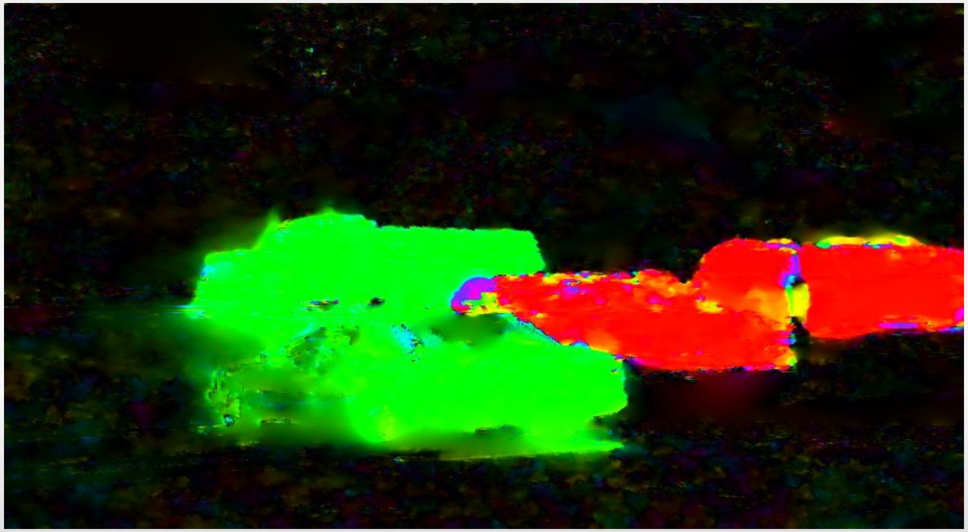
$$u^{k+1} := u^k + du^k$$

Results and Discussions

Results and Discussions







Live Demo

Results and Discussions

Questions?