Computer Vision on the GPU with OpenCV

James Fung
NVIDIA Developer Technology
Outline

- Introduction into OpenCV
- OpenCV GPU module
- Face Detection on GPU
- Pedestrian detection on GPU
OpenCV History

- **Original goal:**
  - Accelerate the field by lowering the bar to computer vision
  - Find compelling uses for the increasing MIPS out in the market

- **Staffing:**
  - Climbed in 1999 to average 7 first couple of years
  - Little development from 2002 - 2008
  - Willow entered in 2008 to accelerate development, NVIDIA joined in 2010
  - 8 full time professional developers, 3 of them dedicated to GPU
OpenCV Functionality Overview

Image processing

General Image Processing
Segmentation
Machine Learning, Detection
Image Pyramids
Transforms
Fitting

Video, Stereo, and 3D

Camera Calibration
Features
Depth Maps
Optical Flow
Inpainting
Tracking
OpenCV Architecture and Development

Languages:
- C
- C++
- Python
- CUDA

3rd party libs:
- Eigen
- IPP
- Jasper
- JPEG, PNG
- OpenEXR
- OpenNI
- QT
- TBB
- VideoInput

Development:
- Maintainers
- Contributors

Modules:
- Core
- HighGUI
- GPU
- ImgProc
- ML
- ObjDetect
- Video
- Calib3D
- Features2D
- FFMPEG
- FLANN

Target archs:
- X86
- X64
- ARM
- CUDA

Target OS:
- Windows
- Linux
- Mac OS
- Android

QA:
- Buildbot
- Google Tests
OpenCV License

Based on BSD license

- Free for commercial and research use
- Does not force your code to be open
- You need not contribute back
  - We hope you will contribute back!
Projects Using OpenCV

- Google Maps, Google street view, Google Earth
- Academic and Industry Research
- Security systems
- Image retrieval
- Video search
- Machine vision factory production systems
- Structure from motion in movies
- Robotics
Outline

- Introduction into OpenCV
- OpenCV GPU module
- Face Detection on GPU
- Pedestrian detection on GPU
OpenCV GPU Module

Motivation:

- Many computer vision tasks are inherently parallel
- GPUs provide **cheap** computational power
OpenCV GPU Module

Goals:

- Provide developers with a convenient computer vision framework on the GPU
- Maintain conceptual consistency with the current CPU functionality
- Achieve the best performance with GPUs
  - Efficient kernels tuned for modern architectures
  - Optimized dataflows (asynchronous execution, copy overlaps, zero-copy)
OpenCV GPU Module Contents

- Image processing building blocks:
  - Color conversions
  - Geometrical transforms
  - Per-element operations
  - Integrals, reductions
  - Template matching
  - Filtering engine
  - Feature detectors

- High-level algorithms:
  - Stereo matching
  - Face detection
  - SURF
OpenCV GPU: Histogram of Oriented Gradients

- Used for pedestrian detection
- Speed-up ~ $8 \times$

Mode: GPU
FPS (HOG only): 34.832
FPS (Total): 23.977
OpenCV GPU: Speeded Up Robust Features

- SURF (12×)
- Brute force matcher
  - K-Nearest search (20-30×)
  - In radius search (3-5×)
OpenCV GPU: Stereo Vision

- Stereo Block Matching (7×)
  - Can run Full HD real-time on Dual-GPU

- Hierarchical Dense Stereo
  - Belief Propagation (20×)
  - Constant space BP (50-100×)
OpenCV GPU: Viola-Jones Cascade Classifier

- Used for face detection
- Speed-up ~ 6x
- Based on NCV classes (NVIDIA implementation)
OpenCV with Multiple GPUs

- Algorithms designed with single GPU in mind
- You can split workload manually in slices:
  - Stereo Block Matching (dual-GPU speedup ~ 1.8×)
  - Multi-scale pedestrian detection: linear speed-up (scale-parallel)
OpenCV and NPP

- NPP is NVIDIA Performance Primitives library of signal and image processing functions (similar to Intel IPP)

- GPU module uses NPP whenever possible
  - Highly optimized implementations for all supported NVIDIA architectures and OS
  - Part of CUDA Toolkit - no additional dependencies

- NVIDIA will continue adding new primitives
  - Several hundred primitives added every CUDA release
  - If you feel like your function could be a primitive - go ahead and add it to NPP_staging! (part of NCV in OpenCV GPU module)
OpenCV GPU Module Usage

- Prerequisites:
  - Get sources from the website
  - CMake
  - NVIDIA Display Driver
  - NVIDIA GPU Computing Toolkit (for CUDA)

- Build OpenCV with CUDA support
- `#include <opencv2/gpu/gpu.hpp>`
OpenCV GPU Data Structures

- **Class GpuMat**
  - For storing 2D image in GPU memory, just like class cv::Mat
  - Reference counting

- **Class CudaMem**
  - For pinned memory support
  - Can be transformed into cv::Mat or cv::gpu::GpuMat

- **Class Stream**
  - Overloads with extra Stream parameter

// class GpuMat
GpuMat(Size size, int type);
GpuMat(const GpuMat& m);
explicit GpuMat (const Mat& m);
GpuMat& operator = (const GpuMat& m);
GpuMat& operator = (const Mat& m);
void upload(const Mat& m);
void upload(const CudaMem& m, Stream& stream);
download(Mat& m) const;
void download(CudaMem& m, Stream& stream) const;

// class Stream
bool queryIfComplete();
void waitForCompletion();
void enqueueDownload(const GpuMat& src, Mat& dst);
void enqueueUpload(const Mat& src, GpuMat& dst);
void enqueueCopy(const GpuMat& src, GpuMat& dst);
OpenCV GPU Module Example

Mat frame;
VideoCapture capture(camera);
cv::HOGDescriptor hog;

hog.setSVMDetector(cv::HOGDescriptor::
        getDefaultPeopleDetector());
capture >> frame;

vector<Rect> found;
hog.detectMultiScale(frame, found,
        1.4, Size(8, 8), Size(0, 0), 1.05, 8);

Mat frame;
VideoCapture capture(camera);

hog.setSVMDetector(cv::HOGDescriptor::
        getDefaultPeopleDetector());
capture >> frame;

GpuMat gpu_frame;
gpu_frame.upload(frame);

vector<Rect> found;
hog.detectMultiScale(gpu_frame, found,
        1.4, Size(8, 8), Size(0, 0), 1.05, 8);
OpenCV GPU Module Performance

Tesla C2050 (Fermi) vs. Core i5-760 2.8GHz (4 cores, TBB, SSE)

– Average speedup with GPU: 33.98×

What can you get from your computer?

– opencv\samples\gpu\performance
– 839 tests for 79 functions
OpenCV GPU Demo Pack

- Contains demos for high-level GPU algorithms:
  - Face detection (6x)
  - Keypoint detection (12x) / Point matching (20-30x)
  - Pedestrian detection (8x)
  - Image Stitching
  - Optical flow
  - Stereo matching (7x/20x/50x)

http://sourceforge.net/projects/opencvlibrary/
OpenCV Stitching Module

- Automatic stitching photos taken from the same point
  - Cylindrical, spherical or planar panoramas
  - Multi-band blending technique
  - Smart seam estimation (graph cut based approach)
  - **GPU acceleration** for the most time-consuming steps
Auto calibration

- Rotation camera movement model
  - Requires all photos to be taken from approximately the same position
  - A few tens of images are recommended for accurate work
  - Works without an initial guess of camera intrinsic parameters

Applications: stitching, augmented reality and many other
Auto calibration sample images
Auto calibration

- Relative errors:

**Dataset 1**

- focal x
- focal y
- principal point x
- principal point y

**Dataset 2**

- focal x
- focal y
- principal point x
- principal point y
OpenCV Needs Your Feedback!

- Help us set development priorities
  - Which OpenCV functions do you use?
  - Which are the most painful and time-consuming today?

- The more information you can provide about your end application, the better

- Feature request/feedback form on OpenCV Wiki:
  http://opencv.willowgarage.com/wiki/OpenCV_GPU
Outline

- Introduction into OpenCV
- OpenCV GPU module
- Face Detection on GPU
- Pedestrian detection on GPU
GPU Face Detection: Motivation

- One of the first Computer Vision problems
- Soul of Human-Computer interaction
- Smart applications in real life
GPU Face Detection: Problem

- Locate all upright frontal faces:

- Where face detection does not work:
GPU Face Detection: Approaches

Viola-Jones Haar classifiers framework:

Basic idea: reject most non-face regions on early stages
Classifiers Cascade Explained

- White points represent face windows passed through the 1, 2, 3, 6, and 20 classifier stages
- Time for **CUDA** to step in! (Parallel windows processing)
**GPU Face Detection: Haar Classifier**

Each stage comprises a strong classifier:

\[
H(X) = \begin{cases} 
1, & \sum_{i=1}^{K} h_i(X) \geq T \\
0, & \text{otherwise}
\end{cases}
\]
Haar Features Explained

Most representative Haar features for Face Detection
Integral Image Explained

- Each Integral Image “pixel” contains the sum of all pixels of the original image to the left and top

- Calculation of sum of pixels in a rectangle can be done in 4 accesses to the integral image
Integral Images with CUDA

Algorithm:
- Integrate image rows
- Integrate image columns

Known as Parallel Scan (one CUDA thread per element):
- Input: 
- Output: 

\[
\begin{array}{cccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \end{array}
\]
Scan Sample: 8 Numbers

\[ \begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]

Legend
- → set
Scan Sample: 8 Numbers

Legend

-> set
Scan Sample: 8 Numbers

Legend

⊕⊕ ⊕⊕ set
Scan Sample: 8 Numbers
GPU Face Detection

NVIDIA Computer Vision SDK: Face Detection in Video Feed

Space - Switch NCV (ON) / OpenCV
L - Switch FullScreen (ON) / LargestFace mode
O - Toggle unfiltered hypothesis localization in FullSearch (OFF)

Running at 39.048305 FPS on GPU
GPU Face Detection Performance

- **640x480_VGA**
- **1280x720_HD720p**
- **1920x1080_HD1080p**

- **GeForce 9800 GTX+**
- **Intel Core2 Duo 2.00GHz**
- **Intel Core2 Duo 3.00GHz**
- **Intel Core i7 965 3.20GHz**
- **GeForce GTX 260**
- **GeForce GTX 480**
OpenCV NCV Framework

Features:

- Native and Stack GPU memory allocators
- Protected allocations (fail-safety)
- Containers: NCVMatrix, NCVVector
- Runtime C++ template dispatcher
- NPP_staging - a place for missing NPP functions
  - Integral images
  - Mean and StdDev calculation
  - Vector compaction
OpenCV NCV Haar Cascade Classifiers

Haar Object Detection from OpenCV GPU module:

- Implemented on top of NCV
- Uses NPP with extensions (NPP_staging)
- Not only faces!
- Suitable for production applications
  - Reliable (fail-safe)
  - Largest Object mode (up to 200 fps)
  - All Objects mode
Outline

- Introduction into OpenCV
- OpenCV GPU module
- Face Detection on GPU
- Pedestrian detection on GPU
Pedestrian Detection

- HOG descriptor
  - Introduced by Navneet Dalal and Bill Triggs
  - Feature vectors are compatible with the INRIA Object Detection and Localization Toolkit
  http://pascal.inrialpes.fr/soft/olt/
Pedestrian Detection: HOG Descriptor

- Object shape is characterized by distributions of:
  - Gradient magnitude
  - Gradient orientation

- Grid of orientation histograms
Pedestrian Detection: Working on Image

- Gamma correction
- Gradients calculation
- Sliding window algorithm
- Multi-scale
Pedestrian Detection: Inside Window

- Compute histograms inside **cells**
- Normalize **blocks** of cells
- One **cell** may belong to >1 **block**
- Apply linear SVM classifier
Pedestrian Detection: Step 1

- Gamma correction improves quality
- Sobel filter 3x3 by columns and rows
- Output: magnitude and angle

\[
G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \ast \text{Image}
\]

\[
G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \ast \text{Image}
\]

\[
G = \sqrt{G_x^2 + G_y^2}
\]

\[
\Theta = \arctan \left( \frac{G_y}{G_x} \right)
\]
Pedestrian Detection: Step 2

- Big intersection in close positions
- Require window stride to be multiple of cell size
- Histograms of blocks are computed independently
Pedestrian Detection: Step 2

- Pixels vote in proportion to gradient magnitude
- Tri-linear interpolation
  - 2 orientation bins
  - 4 cells
- Gaussian
  - Decreases weight of pixels near block boundary
Pedestrian Detection: Step 3

- Normalization
  - L2-Hys norm
    - L2 norm, clipping, normalization
  - 2 parallel reductions in shared memory

- Gradients computation
- Block histograms calculation
- Histograms normalization
- Linear SVM
Pedestrian Detection: Step 4

- Linear SVM
  - Classification is just a dot product
  - 1 thread block per window position

GPU time, %

- Gamma + Gradients: 20%
- Histograms: 39%
- SVM: 30%
- Normalize: 5%
- Other: 6%
Pedestrian Detection Performance

- **8x** times faster!
- Detection rate
  - Same as CPU

![Chart showing performance comparison between CPU and GPU.](chart)

- **GPU Technology Conference**
- **Core i5 2.8 GHz TBB, 4 cores**
- **Tesla C2050**
Thank you

CUDA  http://developer.nvidia.com/cuda
OpenCV  http://opencv.willowgarage.com/wiki
GPU Technology Conference
Spring 2012 | San Francisco Bay Area

The one event you can’t afford to miss

- Learn about leading-edge advances in GPU computing
- Explore the research as well as the commercial applications
- Discover advances in computational visualization
- Take a deep dive into parallel programming

Ways to participate

- Speak - share your work and gain exposure as a thought leader
- Register - learn from the experts and network with your peers
- Exhibit/Sponsor - promote your company as a key player in the GPU ecosystem

www.gputechconf.com