Computer Vision on GPU with OpenCV

Anton Obukhov, NVIDIA (aobukhov@nvidia.com)
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
- JAVA (plans)

Technologies:
- CUDA
- SSE
- TBB

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

Development:
- Maintainers
- Contributors

QA:
- Buildbot
- Google Tests

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

Target archs:
- X86
- X64
- ARM
- CUDA

Target OS:
- Windows
- Linux
- Mac OS
- Android
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
Projects Using OpenCV

• Academic and Research
  – Large community

• Industry
  – Google street view
  – Structure from motion in movies
  – Robotics

• Over 3 000 000 downloads
Recent OpenCV release (2.3rc, June 2011)

- Added patent-free feature descriptors
- Improved camera calibration
- Added panorama stitching module
- GPU module release
- Android port
- Fine documentation (online too: http://opencv.itseez.com/)
- 250 bugfixes
- Full list on http://opencv.willowgarage.com/wiki/OpenCV Change Logs
Outline

• Introduction into OpenCV
• OpenCV GPU module
• Face Detection on GPU
• Pedestrian detection on GPU
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 Module Usage

• Prerequisites:
  – Get sources from SourceForge or SVN
  – CMake
  – NVIDIA Display Driver
  – NVIDIA GPU Computing Toolkit (for CUDA)

• Build OpenCV with CUDA support

• `#include <opencv2/gpu/gpu.hpp>`

http://opencv.willowgarage.com/wiki/InstallGuide
OpenCV GPU Data Structures

• Class GpuMat
  – For storing 2D image in GPU memory, just like class cv::Mat
  – Reference counting
  – Can point to data allocated by user

• 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);
void 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);
cv::gpu::HOGDescriptor hog;

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);

• Designed very similar!
OpenCV and NPP

• NPP is NVIDIA Performance Primitives library of signal and image processing functions (similar to Intel IPP)
  – NVIDIA will continue adding new primitives and optimizing for future architectures

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

• OpenCV extends NPP and uses it to build higher level CV
OpenCV GPU Module Performance

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

- Average speedup for primitives: $33 \times$
  - For “good” data (large images are better)
  - Without copying to GPU

What can you get from your computer?
- opencv\samples\gpu\performance
OpenCV GPU: Histogram of Oriented Gradients

- Used for pedestrian detection
- Speed-up ~ 8x
OpenCV GPU: Speeded Up Robust Features

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

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

• Hierarchical Dense Stereo
  – Belief Propagation (20x)
  – Constant space BP (50-100x)
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 \( \sim 1.8x \))
  - Multi-scale pedestrian detection: linear speed-up (scale-parallel)
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:  
  1 1 1 1 1 1 1 1 1 1

• Output:  
  1 2 3 4 5 6 7 8
GPU Face Detection Performance

![Graph showing FPS performance with different GPUs and processors.](image)
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
  – Edge/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

- **Gradients computation**
- **Block histograms calculation**
- **Histograms normalization**
- **Linear SVM**

- 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
Pedestrian Detection: Step 4

- **Linear SVM**
  - Classification is just a dot product
  - 1 thread block per window position
Pedestrian Detection Performance

• 8x times faster!
• Detection rate – Same as CPU
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
Thank you

http://opencv.willowgarage.com/wiki

anatoly.bakshev@itseez.com