OpenCV on a GPU

Shalini Gupta, Shervin Emami, Frank Brill
NVIDIA
GPU access
• To access NVIDIA cluster send email to jlevites@nvidia.com
• Subject line: “OpenCV GPU Test Drive”
• Add your name and phone number

Webinar Feedback
Submit your feedback for a chance to win Tesla K20 GPU
https://www.surveymonkey.com/s/OpenCV_Webinar

More questions on OpenCV and GPUs
• Stay tuned with NVIDIA webinars:
  http://www.nvidia.com/object/cuda_signup_alerts.html
• Refer to OpenCV Yahoo! Groups
Outline

- OpenCV
- Why GPUs?
- An example - CPU vs. CUDA
- OpenCV CUDA functions
- Discussion
- Future
- Summary
Introduction
- Open source library for computer vision, image processing and machine learning
- Permissible BSD license
- Freely available (www.opencv.org)

Portability
- Real-time computer vision (x86 MMX/SSE, ARM NEON, CUDA)
- C (11 years), now C++ (3 years since v2.0), Python and Java
- Windows, OS X, Linux, Android and iOS
Usage

Usage:
- >6 million downloads, > 47,000 user group
- Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota

Applications:
- Street view image stitching
- Automated inspection and surveillance
- Robot and driver-less car navigation and control
- Medical image analysis
- Video/image search and retrieval
- Movies - 3D structure from motion
- Interactive art installations
Functionality

Desktop
- x86 single-core (Intel started, now Itseez.com) - v2.4.5 >2500 functions (multiple algorithm options, data types)
- CUDA GPU (Nvidia) - 250 functions (5x – 100x speed-up)
  http://docs.opencv.org/modules/gpu/doc/gpu.html
- OpenCL GPU (3rd parties) - 100 functions (launch times ~7x slower than CUDA*)

Mobile (Nvidia):
- Android (not optimized)
- Tegra – 50 functions NEON, GLSL, multi-core (1.6 – 32x speed-up)

*Shengen Yan, AMD Fusion Developer Summit 2012.
Functionality

- Image/video I/O, processing, display (core, imgproc, highgui)
- Object/feature detection (objdetect, features2d, nonfree)
- Geometry-based monocular or stereo computer vision (calib3d, stitching, videostab)
- Computational photography (photo, video, superres)
- Machine learning & clustering (ml, flann)
- CUDA and OpenCL GPU acceleration (gpu, ocl)
Outline

- OpenCV
- Why GPUs?
- An example - CPU vs. CUDA
- OpenCV CUDA functions
- Discussion
- Future
- Summary
Why GPU?

**CPU**
- Reached speed and thermal power limit!
- Incremental improvements (memory caches and complex architectures)
- Multi-core (4/8), but software rarely multi-core

**GPU**
- Highly parallel with 100s of simple cores
- Easier to extend by adding more cores
- Continue to grow exponentially!
GPU > CPU (compute and memory)

*http://hemprasad.wordpress.com/2013/03/03/introduction-to-cuda-5-0/
GPU for OpenCV

Graphics

Inverse Problems

Massively Parallel

Computer Vision

Render Images From Scenes

Understand Scenes From Images
Outline

- OpenCV
- Why GPUs?
- An example - CPU vs. CUDA
- OpenCV CUDA functions
- Discussion
- Future
- Summary
OpenCV CPU example

#include <opencv2/opencv.hpp>
using namespace cv;

int main() {
    Mat src = imread("car1080.jpg", 0);
    if (!src.data) exit(1);
    Mat dst;
    bilateralFilter(src, dst, -1, 50, 7);
    Canny(dst, dst, 35, 200, 3);
    imwrite("out.png", dst);
    return 0;
}
OpenCV CPU example

```cpp
#include <opencv2/opencv.hpp>
using namespace cv;
int main() {
    Mat src = imread("car1080.jpg", 0);
    if (!src.data) exit(1);
    Mat dst;
    bilateralFilter(src, dst, -1, 50, 7);
    Canny(dst, dst, 35, 200, 3);
    imwrite("out.png", dst);
    return 0;
}
```
OpenCV CUDA example

```cpp
#include <opencv2/opencv.hpp>
#include <opencv2/gpu/gpu.hpp>
using namespace cv;

int main() {
    Mat src = imread("car1080.jpg", 0);
    if (!src.data) exit(1);
    gpu::GpuMat d_src(src);
    gpu::GpuMat d_dst;
    gpu::bilateralFilter(d_src, d_dst, -1, 50, 7);
    gpu::Canny(d_dst, d_dst, 35, 200, 3);
    Mat dst(d_dst);
    imwrite("out.png", dst);
    return 0;
}
```

- OpenCV GPU header file
- Upload image from CPU to GPU memory
- Allocate a temp output image on the GPU
- Process images on the GPU
- Process images on the GPU
- Download image from GPU to CPU memory
OpenCV CUDA example

```cpp
#include <opencv2/opencv.hpp>
#include <opencv2/gpu/gpu.hpp>
using namespace cv;

int main() {
    Mat src = imread("car1080.jpg", 0);
    if (!src.data) exit(1);
    gpu::GpuMat d_src(src);
    gpu::GpuMat d_dst;
    gpu::bilateralFilter(d_src, d_dst, -1, 50, 7);
    gpu::Canny(d_dst, d_dst, 35, 200, 3);
    Mat dst(d_dst);
    imwrite("out.png", dst);
    return 0;
}
```
#include <opencv2/opencv.hpp>
#include <opencv2/gpu/gpu.hpp>
using namespace cv;
int main() {
    Mat src = imread("car1080.jpg", 0);
    if (!src.data) exit(1);
    Mat dst;
    bilateralFilter(src, dst, -1, 50, 7);
    Canny(dst, dst, 35, 200, 3);
    imwrite("out.png", dst);
    return 0;
}

#include <opencv2/opencv.hpp>
#include <opencv2/gpu/gpu.hpp>
using namespace cv;
int main() {
    Mat src = imread("car1080.jpg", 0);
    if (!src.data) exit(1);
    Mat d_src, d_dst;
    gpu::GpuMat d_src(src);
    gpu::GpuMat d_dst;
    gpu::bilateralFilter(d_src, d_dst, -1, 50, 7);
    gpu::Canny(d_dst, d_dst, 35, 200, 3);
    Mat dst(d_dst);
    imwrite("out.png", dst);
    return 0;
}

**TOTALS:**

CPU = 2540ms  CUDA = 200ms*

*results obtained over many frames*
CUDA Matrix Operations

Point-wise matrix math
- `gpu::add()`, `::sum()`, `::div()`, `::sqrt()`, `::sqrSum()`, `::meanStdDev()`, `::min()`, `::max()`, `::minMaxLoc()`, `::magnitude()`, `::norm()`, `::countNonZero()`, `::cartToPolar()`, etc..

Matrix multiplication
- `gpu::gemm()`

Channel manipulation
- `gpu::merge()`, `::split()`

*www.shervinemami.info/blobs.html*
CUDA Geometric Operations

Image resize with sub-pixel interpolation
- `gpu::resize()`

Image rotate with sub-pixel interpolation
- `gpu::rotate()`

Image warp (e.g., panoramic stitching)
- `gpu::warpPerspective()`, `::warpAffine()`

*www.skyscrapercity.com*
CUDA other Math and Geometric Operations

Integral images (e.g., object detection and recognition, feature tracking)
- `gpu::integral()`, `::sqrIntegral()`

Custom geometric transformation (e.g., lens distortion correction)
- `gpu::remap()`, `::buildWarpCylindricalMaps()`, `::buildWarpSphericalMaps()`
CUDA Image Processing

Smoothing
- `gpu::blur()`, `::boxFilter()`, `::GaussianBlur()`

Morphological
- `gpu::dilate()`, `::erode()`, `::morphologyEx()`

Edge Detection
- `gpu::Sobel()`, `::Scharr()`, `::Laplacian()`, `gpu::Canny()`

Custom 2D filters
- `gpu::filter2D()`, `::createFilter2D_GPU()`, `::createSeparableFilter_GPU()`

Color space conversion
- `gpu::cvtColor()`
CUDA Image Processing

Image blending
- `gpu::blendLinear()`

Template matching (automated inspection)
- `gpu::matchTemplate()`

Gaussian pyramid (scale invariant feature/object detection)
- `gpu::pyrUp()`, `::pyrDown()`

Image histogram
- `gpu::calcHist()`, `gpu::histEven`, `gpu::histRange()`

Contract enhancement
- `gpu::equalizeHist()`

*OpenCV Histogram Equalization Tutorial*
CUDA De-noising

Gaussian noise removal
- `gpu::FastNonLocalMeansDenoising()`

Edge preserving smoothing
- `gpu::bilateralFilter()`

*www.cs.cityu.edu.hk/~qiyang/publications.html*
CUDA Fourier and MeanShift

Fourier analysis
- `gpu::dft()`, `::convolve()`, `::mulAndScaleSpectrums()`, etc.

MeanShift
- `gpu::meanShiftFiltering()`, `::meanShiftSegmentation()`

* [www.lfb.rwth-aachen.de/en/education/Workshop/t.schoenen.html](www.lfb.rwth-aachen.de/en/education/Workshop/t.schoenen.html)
CUDA Shape Detection

Line detection (e.g., lane detection, building detection, perspective correction)
- gpu::HoughLines(), ::HoughLinesDownload()

Circle detection (e.g., cells, coins, balls)
- gpu::HoughCircles(), ::HoughCirclesDownload()

*www.potucek.net/projects.html
+www.cs.bgu.ac.il/~icbv071/StudentProjects.php
CUDA Object Detection

HAAR and LBP cascaded adaptive boosting (e.g., face, nose, eyes, mouth)
- `gpu::CascadeClassifier_GPU::detectMultiScale()`

HOG detector (e.g., person, car, fruit, hand)
- `gpu::HOGDescriptor::detectMultiScale()`

*glowingpython.blogspot.com/2011/11/
+src: www.cvc.uab.es/~dvazquez/wordpress/?page_id=234
CUDA Object Recognition

Interest point detectors
- `gpu::cornerHarris()`, `::cornerMinEigenVal()`, `::SURF_GPU`, `::FAST_GPU`, `::ORB_GPU()`, `::GoodFeaturesToTrackDetector_GPU()`

Feature matching
- `gpu::BruteForceMatcher_GPU()`, `::BFMatcher_GPU()`

*Kathleen Tuite, CSE576 Project, U of W, 2008.*
CUDA Stereo and 3D

- RANSAC (e.g., object 3D pose, structure from motion, stereo vision)
  - `gpu::solvePnP_Ransac()`

- Stereo correspondence (disparity map)
  - `gpu::StereoBM_GPU()`, `::StereoBeliefPropagation()`, `::StereoConstantSpaceBP()`, `::DisparityBilateralFilter()`

- Represent stereo disparity as 3D or 2D
  - `gpu::reprojectImageTo3D()`, `::drawColorDisp()`

*www.cyverse.co.jp/eng/*
CUDA Optical Flow

- Dense/sparse optical flow (with simple block matching, pyramidal Lucas-Kanade, Brox, Farnebac, TV-L1)
  - gpu::FastOpticalFlowBM(), ::PyrLKOpticalFlow, ::BroxOpticalFlow(), ::FarnebackOpticalFlow(), ::OpticalFlowDual_TV_L1_GPU(), ::interpolateFrames()
- Applications: motion estimation, object tracking, image interpolation

*www.scs.ch/blog/en/2013/01/dsp-optimisation-optical-flow/*
CUDA Background Segmentation

- Foreground/background segmentation (e.g., object detection/removal, motion tracking, background removal)
  - gpu::FGDStatModel, ::GMG_GPU, ::MOG_GPU, ::MOG2_GPU

*www.technologytell.com/apple/96538/inpaint-pro-4-appidemic/*
Custom CUDA code

- CPU OpenCV provides access to image pixels to write custom functions
- ~ GPU-accelerated pixel access to write custom CUDA kernels – requires knowledge of CUDA
- http://docs.opencv.org/modules/gpu/doc/gpu.html
Outline

- OpenCV
- Why GPUs?
- An example - CPU vs. CUDA
- OpenCV CUDA functions
- Discussion
- Future
- Summary
CUDA Advantages

- Similar to CPU code – same API
- Great for long parallel operations and low data transfers – slowest CPU functions
- Significant boosts on GPU (e.g., bilateralFilter() – 12.7x speedup)
- Makes CPU compute bound CV tasks feasible in real-time (e.g., stereo vision, pedestrian detection, dense optical flow)
- Runtime check and use of CUDA acceleration
CUDA Disadvantages

- Only 250 functions
- Limited data types
  - GPU: 8-bit & 32-bit grayscale
  - CPU: +16-bit (HDR) & 32-bit color, ROI
- Explicitly program for CUDA
- Handle data transfers between CPU and GPU
- Only on NVIDIA GPU
- Some serial operations not sped up, e.g., Canny()
- CUDA has startup delay
CUDA Start Up Delay

- First CUDA call initializes CUDA module
- Typical first call – CPU to GPU transfer (~2000ms and 1ms after that)
- Affects single frame applications, videos OK
Serial functions on CUDA

- Serial functions don’t port well
- Equivalent efficient CUDA parallel algorithms exist (e.g., image sums, intergal images, histogram) – see [www.moderngpu.com](http://www.moderngpu.com) or Udacity’s CS344
- Serial GPU code saves transfer time
- CUDA CV algorithms actively being researched
- New CUDA generations (hw+sw) allow more algorithms
## GPU Memory Access

<table>
<thead>
<tr>
<th>Dedicated GPU</th>
<th>Integrated GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Own high speed memory</td>
<td>Shares CPU’s slow memory</td>
</tr>
<tr>
<td>High data transfer time</td>
<td>Free data transfers</td>
</tr>
<tr>
<td>Higher memory BW (~10x)</td>
<td>Lower memory BW</td>
</tr>
<tr>
<td>Desktops/workstations</td>
<td>Laptops</td>
</tr>
<tr>
<td>Functions with lots of processing</td>
<td>Functions with little processing</td>
</tr>
</tbody>
</table>
Outline

- OpenCV
- Why GPUs?
- An example - CPU vs. CUDA
- OpenCV CUDA functions
- Discussion
- Future
- Summary
Future - CUDA on Mobile

- Tegra with CUDA GPU (Logan) – mobile CUDA openCV possible!
- Low power and area (automotive, mobile)
- Kayla$^1$ and Jetson$^2$ (Tegra 3 + dGPU)
- Currently on mobile (Tegra) – NEON, GLES, and multi-threading(OpenCV4Tegra)
- Custom NEON/GLES programming hard, CUDA easier

$^1$www.nvidia.com/object/seco-dev-kit.html
$^2$www.nvidia.com/object/jetson-automotive-development-platform.html
Future - Khronos OpenVX

- “OpenVX” - new standard for hw accelerated CV
  - Khronos (e.g., OpenGL, OpenCL, OpenVG)
  - NVIDIA, Texas Instruments, Samsung, Qualcomm, ARM, Intel
  - For mobile acceleration hw (CPU, GPU, DSP, fixed-function)
- Graph model vs. synchronous programming model
- CV nodes linked in graph at initialization, efficient hw specific processing pipeline automatically generated
- OpenCV to use OpenVX internally to better use hw acceleration
Outline

- OpenCV
- Why GPUs?
- An example - CPU vs. CUDA
- OpenCV CUDA functions
- Discussion
- Future
- Summary
Summary

- OpenCV a well established comprehensive library
- GPU > CPU and growing
- Many CV algorithms great for GPU
- CUDA OpenCV - 250 functions, custom GPU kernels
  - [http://docs.opencv.org/modules/gpu/doc/gpu.html](http://docs.opencv.org/modules/gpu/doc/gpu.html)
- OpenVX extends beyond GPU (DSP, fixed function hw)
GPU Everywhere!

- **Tegra Tablets, Smartphones, Shield**
  - Tegra 3 CPU & GPU, running Android, WinRT or Linux.
- **Kayla Development Board**
  - Tegra 3 CPU + laptop GPU, running Linux.
- **Jetson Automotive Platform**
  - Tegra 3 CPU + laptop GPU, running Linux.
- **Desktop**
  - Intel or AMD CPU with GeForce, Quadro or Tesla GPUs.
- **Cloud & Supercomputer centers**
  - Amazon Cloud with Fermi GPUs, Nvidia GRID
GPU access
- To access NVIDIA cluster send email to jlevites@nvidia.com
- Subject line: “OpenCV GPU Test Drive”
- Add your name and phone number

Webinar Feedback
Submit your feedback for a chance to win Tesla K20 GPU
https://www.surveymonkey.com/s/OpenCV_Webinar

More questions on OpenCV and GPUs
- Stay tuned with NVIDIA webinars:
  http://www.nvidia.com/object/cuda_signup_alerts.html
- Refer to OpenCV Yahoo! Groups
Upcoming GTC Express Webinars

June 12 - Easily Accelerating Existing Monte Carlo Code: CVA and CCR Examples

June 20 - GPU Accelerated XenDesktop for Designers and Engineers

June 26 - Understanding Dynamic Parallelism at Any Scale with Allinea’s Unified Tools

July 9 - NVIDIA GRID VCA: A Turnkey Appliance for Design and Engineering Applications

July 10 - Introduction to the CUDA Toolkit as an Application Build Tool

Register at www.gputechconf.com/gtcexpress