ACCELERATING COMPUTER VISION WITH OPENCV AND CUDA

Kirill Kornyakov (Itseez)
AGENDA

1. Slides: OpenCV with CUDA
2. Lab: Video Stabilization
3. QA
OpenCV
Open Source Computer Vision Library

1. 2500+ algorithms
2. More than 8M downloads, active community
3. Permissive BSD license
4. Cross-platform
5. Real-time performance
OPENCV ARCHITECTURE

Bindings
- Python
- Java
- C

Library
- Eigen
- IPP
- JPEG
- PNG
- Jasper
- Multimedia
- TBB

Threading API
- Concurrency
- GCD
- TBB

Operating System
- Windows
- Linux
- MacOSX
- iOS
- Android
- WinRT

Hardware
- x86, x64

Acceleration API
- SSE, AVX
- OpenCL
- GPU
- CUDA
- ARM
- NEON
- MIPS

Dependences:
- CUDA
- OpenCL
- NEON

Your application

Tegra K1
**OPENCV CUDA API**

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

hog.setSVMType(cv::HOGDescriptor::getDefaultPeopleDetectorector());
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.setSVMType(cv::HOGDescriptor::getDefaultPeopleDetectorector());
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 CUDA
WHY CUDA IS SO GOOD FOR VISION?

Two analogies:

- Computer Vision vs Computer Graphics
- Human Vision
COMPUTER VISION AND GPU

High-level information about a scene

Red ball  Human face

Computer Vision

Raster image

Computer Graphics

The same hardware boosts both!
HUMAN VISION

Norbert Kruger et al.

*Deep Hierarchies in the Primate Visual Cortex: What Can We Learn For Computer Vision?*

- 50% of neocortex is about vision (color area)
- 60% of it is for primitive image processing (yellow area)

So, ~30% of our brainpower is spent on simple image processing!
TEGRA K1
ADDITIONAL INFORMATION

GTC talks
- Joe Stam (NVIDIA) - Extending OpenCV with GPU Acceleration
- Anton Obukhov (NVIDIA) - Computer Vision on GPU with OpenCV
- James Fung (NVIDIA) - Computer Vision on GPU with OpenCV

NVIDIA webinars
- Shalini Gupta (NVIDIA) - OpenCV - Accelerated Computer Vision using GPUs
- Anatoly Baksheev (Itseez) - Getting Started with GPU-accelerated Computer Vision using OpenCV and CUDA

Documentation
- Homepage: http://opencv.org/platforms/cuda.html
Computational Photography using Mobile CUDA

Anna Kogan¹, Vlad Vinogradov¹, Kirill Kornyakov¹, Dmitry Retinskiy¹, Colin Tracey²
¹Itseez, ²NVIDIA

Introduction
Computational strength of modern devices brings personal photography to a new level.

We implement advanced photo effects using stacks of images:
• **Object Removal** is a technique for removing occluding objects.
• **Object Cloning** creates collages showing movement of an object.

Object Removal

Object Cloning

Algorithm

- Image Sequence (7 frames)
- Image Registration
- BG/FG Segmentation
- Object Removal
- Object Cloning
- Final Image

Implementation Details

Target platform
- SoC: Tegra K1
- OS: Android
- Technologies: CUDA
- Libraries: VisionWorks, OpenCV for Tegra

Image Processing primitives
- Image registration: ORB feature detection,
  Optical Flow, Homography Estimation,
  Perspective Warping
- BG/FG Segmentation: Background Modeling,
  FG mask construction
- Removal and Cloning: Poisson Blending

BG/FG Segmentation on Tegra K1 with CUDA

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>BG/FG Segmentation</td>
<td>3.38 sec</td>
<td>1.84 sec</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.8x</td>
</tr>
</tbody>
</table>

Acknowledgements

This research was supported by NVIDIA corporation.
CUDA-optimized Panorama Stitching on Tegra K1

Leonid Beynenson¹, Marina Kolpakova¹, Dmitry Retinskiy¹, Colin Tracey²
¹Itseez. ²NVIDIA

Introduction
Described method allows to create panoramic images using pixel-precise alignment of frames from the camera. Though computationally expensive, it can be implemented on a mobile device with a CUDA capable GPU. As the result application works in real-time on a Tegra K1 tablet.

Target Platform
- OS: Android
- SoC: Tegra K1
- Technologies: ARM NEON, CUDA
- Camera resolution: 640x480

Algorithm Outline
- VideoStream
- Tracking
- Finalization
- Panorama

Tracking
- Camera pose is tracked in real-time w.r.t the closest of already stitched keyframes.
- New keyframe is stitched when overlapping with the closest of already stitched keyframes is low.

Finalization
- Simultaneous optimization of all camera poses for all overlapping keyframes pairs.
- Improving “pixel to pixel” correspondence.
- The result is a panorama without artifacts.

CUDA-optimized functions
1. Image warping: 15x speedup due to low latency L1 on Tegra K1.
2. Sobel operator: 15x speedup (2x faster than NEON SIMD optimization) due to kernel unrolling.
3. Estimation of Jacobians: 6x speedup due to fast floating point math and warp shuffle instructions.

Results of optimization for Tegra K1

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tracking</td>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>Finalization</td>
<td>8x</td>
<td>3x</td>
</tr>
</tbody>
</table>

Object Detection: GPU-friendly soft cascades

Marina Kolpakova, Anatoly Baksheev, Alexander Smorkalov

Introduction
- Fast on-road object detection is an important ADAS feature (advanced driver assistance systems).
- We propose CUDA implementation of soft cascade detector [1] that allows real-time object detection on Tegra K1 platform.
- Applicable for pedestrian and vehicle detection.

Soft Cascade

\[ H_t(x) = \sum_{i=1}^{t} c_i(x) \]

Based on AdaBoost approach and features, but is able to reject negative patches after each new feature evaluated.

Naive approach: Thread per window

Proposed approach: Warp per N windows

Optimization notes
- Thread / window
  - Access to gmem is coalesced in the beginning, but sparse at the latest stages; unbalanced workload, warp processes relatively long sequence of 32 windows, they likely diverge.
  - Time of block residence on SM(X) is \( T(b) = \max \{ T(w_0), ..., T(w_{n\text{, block\_size}}) \} \);
- Warp / N windows
  - Each warp loads 8 features for 4 windows. Each feature consists of 4 pixels. Total warp transactions: 8x4 of 16 bytes (instead of 32x4 of 4 bytes). Warp is active while at least one of window positions is active.
  - Balanced workload; spatial locality.

Results: Caltech [2], Pedestrians

<table>
<thead>
<tr>
<th>Sequence</th>
<th>thread / window (ms)</th>
<th>warp / window (ms)</th>
<th>warp / 4 windows (ms)</th>
<th>speedup w/w (X-factor)</th>
<th>speedup w/4w (X-factor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq06</td>
<td>169.13</td>
<td>98.29</td>
<td>27.13</td>
<td>1.72</td>
<td>6.23</td>
</tr>
<tr>
<td>seq07</td>
<td>166.92</td>
<td>100.12</td>
<td>36.52</td>
<td>1.66</td>
<td>4.57</td>
</tr>
<tr>
<td>seq08</td>
<td>172.89</td>
<td>98.12</td>
<td>38.87</td>
<td>1.76</td>
<td>4.44</td>
</tr>
<tr>
<td>seq09</td>
<td>175.82</td>
<td>102.54</td>
<td>34.18</td>
<td>1.76</td>
<td>4.45</td>
</tr>
<tr>
<td>seq10</td>
<td>144.13</td>
<td>96.87</td>
<td>32.40</td>
<td>1.71</td>
<td>5.14</td>
</tr>
</tbody>
</table>
VIDEO STABILIZATION
APPLICATION LIFECYCLE

POC (x86) → Porting → Profiling → Optimize bottlenecks → Fine Tuning → Productization

Regression Tests → Performance Tests
<table>
<thead>
<tr>
<th>Feature</th>
<th>CPU version (ms)</th>
<th>Naive GPU version (ms)</th>
<th>GPU w/ buffer reuse (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature Detection</td>
<td>43</td>
<td>90</td>
<td>20</td>
</tr>
<tr>
<td>Feature Tracking</td>
<td>51</td>
<td>64</td>
<td>11.8</td>
</tr>
<tr>
<td>Image Warping</td>
<td>24</td>
<td>7.5</td>
<td>0.45</td>
</tr>
<tr>
<td>TOTAL</td>
<td>125 (118)</td>
<td>312 (160)</td>
<td>46 (32)</td>
</tr>
</tbody>
</table>
CONCLUSION

- GPU greatly helps in real-time Computer Vision apps
- It helps even better on mobile devices, where we are usually power/performance bound