GeoInt Accelerator
• Platform to start developing with GPUs
• GPU access, GPU-accelerated apps and libraries
• Register to learn more: http://goo.gl/Eui6k6

Webinar Feedback
Submit your feedback for a chance to win Tesla K20 GPU*
https://www.surveymonkey.com/s/OpenCV_July30
* Offer is valid until August 13th 2013

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
  OpenCV.org: http://answers.opencv.org/questions/
Anatoly Baksheev, Itseez Inc.
Outline

• Getting and building OpenCV with CUDA
• GPU module API
• Overlapping operations
• Using GPU module with your CUDA code
• Questions & Answers
What is OpenCV

• Popular Computer Vision library
  – 6M downloads
  – BSD license
  – 1000s CV functions for solving various problems
  – Various optimizations
    • OpenCV GPU module contains CUDA acceleration
    • Intel TBB, IPP, SSE, ARM NEON & GLSL (Tegra)
Getting OpenCV

• Source Forge
    • Source package for each release (recommended)
    • OpenCV GPU Demo Pack binaries
    • (Precompiled OpenCV binaries with CUDA support)

• Git repository (modified daily)
  – `git clone` [git://github.com/Itseez/opencv.git](git://github.com/Itseez/opencv.git) (GitHub mirror)
  – `git checkout -b 2.4 origin/2.4`

• Two branches
  – **2.4** with binary compatibility between releases (tagged as 2.4.x)
  – **master** (coming OpenCV 3.0 with algorithms API changes, [separate webinar](#))
Building OpenCV with GPU support

• Prerequisites
  – OpenCV sources
  – CMake 2.8.8 (http://www.cmake.org/)
    • CMake-GUI
  – NVIDIA Display Driver
  – NVIDIA GPU Computing Toolkit (for CUDA)
    • https://developer.nvidia.com/cuda-toolkit

  – And your favorite IDE/compiler
Building OpenCV with GPU support

- **Build steps** (screenshots for Windows 7, Visual Studio)
  - Run CMake GUI and set **source** and **build** directories, press **Configure** and select your compiler to generate project for.

  ![CMake GUI](image1)

  The base folder of OpenCV source code

  ![Directory selection](image2)

  Where to put compiled OpenCV library

  ![Compiler selection](image3)
Building OpenCV with GPU support

• Build steps
  – Run CMake GUI and set **source** and **build** directories, press **Configure** and select your compiler to generate project for.
  – Enable **WITH_CUDA** flag and ensure that CUDA Toolkit is detected correctly by checking all variables with ‘CUDA_’ prefix.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA_TOOLKIT_INCLUDE</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/include</td>
</tr>
<tr>
<td>CUDA_TOOLKIT_ROOT_DIR</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0</td>
</tr>
<tr>
<td>CUDA_VERBOSITY_BUILD</td>
<td>0</td>
</tr>
<tr>
<td>CUDA_VERSION</td>
<td>5.0</td>
</tr>
<tr>
<td>CUDA_cublas_LIBRARY</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/lib32/cublas.lib</td>
</tr>
<tr>
<td>CUDA_cufft_LIBRARY</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/lib32/cufft.lib</td>
</tr>
<tr>
<td>CUDA_cupti_LIBRARY</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/lib32/cupti.lib</td>
</tr>
<tr>
<td>CUDA_curand_LIBRARY</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/lib32/cublas.lib</td>
</tr>
<tr>
<td>CUDA_cusparse_LIBRARY</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/lib32/cusparse.lib</td>
</tr>
<tr>
<td>CUDA_npp_LIBRARY</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/lib32/npp.lib</td>
</tr>
<tr>
<td>CUDA_nvcuvid_LIBRARY</td>
<td>C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v5.0/lib32/nvcuvid.lib</td>
</tr>
<tr>
<td>WITH_CUDA</td>
<td>checked</td>
</tr>
</tbody>
</table>

Press Configure to update and display new values in red, then press Generate to generate selected build files.
Building OpenCV with GPU support

• Build steps
  – Run CMake GUI and set source and build directories, press Configure and select you compiler to generate project for.
  – Enable WITH_CUDA flag and ensure that CUDA Toolkit is detected correctly by checking all variables with ‘CUDA_’ prefix.
  – Press Configure and Generate to generate a project
    • On Windows, open the Visual Studio solution and click on “Build Solution”.
    • On Linux, run “make” from the build folder.
Installing OpenCV (optional)

• Running OpenCV install scripts is a way to put all headers, libs and binaries to one place for easier use and deployment
  – Set **CMAKE_INSTALL_PREFIX** variable
  – Build **INSTALL** target
Adding OpenCV to your project via CMake

• Create a ‘main.cpp’ file and **CMakeLists.txt** script as below

```cmake
# CMakeLists.txt

# Define CMakeLists.txt
project(sample_project CXX)

# Find OpenCV
find_package(OpenCV REQUIRED)

# Include directories
include_directories(${OpenCV_INCLUDE_DIRS})

# Source files
file(GLOB srcs *.cpp *.h)

# Add executable
add_executable(sample_app ${srcs})

# Target link libraries
target_link_libraries(sample_app ${OpenCV_LIBS})
```

• Use CMake to generate and build the sample. Set **OpenCV_DIR** variable for the project to **build** or **install** OpenCV folder

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenCV_3RDPARTY_LIB_DIR_DBG</td>
<td>E:/builds/openvc/install/share/OpenCV/3rdparty/lib</td>
</tr>
<tr>
<td>OpenCV_3RDPARTY_LIB_DIR_OPT</td>
<td>E:/builds/openvc/install/share/OpenCV/3rdparty/lib</td>
</tr>
<tr>
<td>OpenCV_CONFIG_PATH</td>
<td>E:/builds/openvc/install</td>
</tr>
<tr>
<td>OpenCV_DIR</td>
<td>E:/builds/openvc/install</td>
</tr>
<tr>
<td>OpenCV_LIB_DIR_DBG</td>
<td>E:/builds/openvc/lib</td>
</tr>
<tr>
<td>OpenCV_LIB_DIR_OPT</td>
<td>E:/builds/openvc/install</td>
</tr>
</tbody>
</table>

Set this. Others are detected automatically
Outline

• Getting and building OpenCV with CUDA
• GPU module API
• Overlapping operations
• Using GPU module with your CUDA code
• Questions & Answers
GPU module design considerations

• Key ideas
  – Explicit control of data transfers between CPU and GPU
  – Minimization of the data transfers
  – Completeness
    • Port everything even functions with little speed-up

• Solution
  – Container for GPU memory with upload/download functionality
  – GPU module function take the container as input/output parameters
GpuMat - container for GPU memory

• Class GpuMat – for storing 2D (pitched) data on GPU
  – Interface similar to cv::Mat(), supports reference counting
  – Its data is not continuous, extra padding in the end of each row
  – It contains:
    • data - Pointer data beginning in GPU memory
    • step - distance in bytes is between two consecutive rows
    • cols, rows – fields that contain image size
    • Other fields for internal use only

GpuMat cloud = ... (get 3-channel float data from a file and copy to GPU)

// compute address of Point at (x, y)
cv::Point3f* pixel_ptr = (cv::Point3f*)(cloud.data + cloud.step * y) + x;

// the same, but simplified
cv::Point3f* pixel_ptr = cloud.ptr<cv::Point3f>(y) + x;
Operations with GpuMat

• Allocations (similar to cv::Mat)

```cpp
void GpuMat::GpuMat(const cv::Size& size, int type);
void GpuMat::create(const cv::Size& size, int type);
```

//Type examples:
- CV_8U - grayscale,
- CV_8UC3 - BGR,
- CV_32FC3 - 3D point
- CV_16U - depth

GpuMat image1(Size(1902,1080), CV_8U);
GpuMat image2;
image2.create(Size(1, 1000), CV_32FC3);

• Creating GpuMat header for user allocated data

```cpp
void GpuMat::GpuMat(const cv::Size& size, int type, void* data, size_t step);
thrust::device_vector<float> vector(10000);
GpuMat header(Size(vector.size(), 1), CV_32F, &vector[0], vector.size() * sizeof(float));
cv::gpu::threshold(header, header, value, 1000, THRESH_BINARY);
```
Operations with GpuMat

• Copying between CPU-GPU and GPU-GPU

```cpp
void GpuMat::GpuMat(const cv::Mat& host_data);
void GpuMat::upload(const cv::Mat& host_data);
void GpuMat::download(cv::Mat& host_data);
void GpuMat::copyTo(cv::gpu::GpuMat& other);

//Examples
Mat host_image = cv::imload("file.png"); //load image from file

GpuMat device_image1;
device_image1.upload(host_image1);       //allocate memory and upload to GPU

GpuMat device_image2;
device_image1.copyTo(device_image2);     //allocate memory and GPU-GPU copy

device_image2.download(host_image);      //download data
```
void process() {
    cv::gpu::setDevice(0);
    cv::VideoCapture capture("video.avi");
    cv::Mat templ_h = cv::imread("nvlogo.png");
    cv::gpu::GpuMat templ_d(templ_h); // upload
    cv::gpu::GpuMat image_d, result;
    cv::Mat image_h;

    for(;;) {
        capture >> image_h;
        if(image_h.empty())
            break;

        image_d.upload(image_h);
        cv::gpu::matchTemplate(image_d, templ_d, result, cv::TM_CCORR);

        double max_value;
        cv::Point location;
        cv::gpu::minMaxLoc(result, 0, &max_value, 0, &location);
        if (max_value > threshold)
            drawRectangleAndShowImage(location, ...);
    }
}
OpenCV GPU Module functionality

- **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
Outline

• Getting and building OpenCV with CUDA
• GPU module API
• Overlapping operations
• Using GPU module with your CUDA code
• Questions & Answers
Overlapping operations with CUDA

• Which operations can be run simultaneously?
  – Host and device code execution
  – Data transfer and device code execution
  – Several device code executions
    • Only if GPU is under occupied
    • Must be executed in different streams
    • (won’t talk about this)
Overlapping operations with CUDA

• Advantages
  – Allows to use whole computing power of your PC (CPU + GPU)
  – Allows to partly hide data transfer overheads
    • Result from previous frame is being downloaded
    • Current is being processed on GPU
    • Next frame is being uploaded

Processing time is smaller than without overlapping
Concurrency in OpenCV GPU

• Class CudaMem
  – Allocates page-locked CPU memory
    • Required for asynchronous data transfers,
      • 2x faster transfer than regular memory, but limited by available RAM
  – Interface similar to Mat or GpuMat
  – Convertible to Mat, can pass to any CPU function

```
CudaMem page_locked(Size(1920, 1080), CV_32FC3);  // allocate page locked memory
Mat header = page_locked;                           // no copy, just header

VideoCapture cap("video1080p.avi"); // open video stream
for(;;)
{
    cap >> header; // read video frame to the header which actually points to page-locked buffer
    // now can copy the page_locked to GPU in asynchronous fashion
    ...  
}
```
Concurrency in OpenCV GPU

• **Class Stream** (wrapper for cudaStream_t)
  – Represents asynchronous queue of operations
  – Allows querying/waiting operations to complete
  – Enqueue asynchronous data transfers

```cpp
bool Stream::queryIfComplete();
void Stream::waitForCompletion();

void Stream::enqueueDownload(const GpuMat& src, CudaMem& dst);
void Stream::enqueueUpload(const CudaMem& src, GpuMat& dst);
void Stream::enqueueCopy(const GpuMat& src, GpuMat& dst);

typedef void (*StreamCallback)(Stream& stream, int status, void* userData);
void Stream::enqueueHostCallback(StreamCallback callback, void* userData);
```
Concurrency in OpenCV GPU

• Putting device operations into the queue

```c++
namespace cv { namespace gpu {
  void a_opencv_gpu_function(..., Stream& stream = Stream::Null());
}

// synchronous case
device_image.upload(host_image);               // blocks until upload is done
a_opencv_gpu_function(device_image, output);   // blocks until operation finishes

// asynchronous case
Stream stream;
stream.enqueueUpload(host_image, device_image);        // returns immediately
a_opencv_gpu_function(device_image, output, stream);   // returns immediately
Stream.enqueueDownload(output, output_host);            // returns immediately

// CPU resources are available.
// Let’s compute on CPU here while GPU does own work.
stream.waitForCompletion();
// output_host is ready!
```
Concurrency in OpenCV GPU

• Current limitation:
  – Unsafe to enqueue the same GPU operation multiple times

• The limitation will be removed in OpenCV 3.x
  – (will re-implement without using constant memory although it may lead to slightly worse performance for small number of kernels)
Outline

• Getting and building OpenCV with CUDA
• GPU module API
• Overlapping operations
• Using GPU module with your CUDA code
• Questions & Answers
Writing your own CUDA code

- **GpuMat** (can’t be passed to cu-file due to nvcc compiler issue, this will be fixed in OpenCV 3.0)
  - **data, step, cols, rows** – can just pass to your code
  - Convertible to `PtrStep<T>`, `PtrStepSz<T>` structures

```cpp
// swap_rb.cpp
#include <opencv2/gpu/stream_accessor.hpp>

void swap_rb_caller(const PtrStepSz<uchar3>& src, PtrStep<uchar3> dst, cudaStream_t stream);

void swap_rb(const GpuMat& src, GpuMat& dst, Stream& stream = Stream::Null())
{
    CV_Assert(src.type() == CV_8UC3);
    dst.create(src.size(), src.type());  // create if not allocated yet
    cudaMemcpy_t s = StreamAccessor::getStream(stream);
    swap_rb_caller(src, dst, s);
}
```
Writing your own CUDA code

```cpp
#include <opencv2/core/cuda_devptrs.hpp>

__global__ void swap_rb_kernel(const PtrStepSz<uchar3> src, PteStep<uchar3> dst) {
    int x = threadIdx.x + blockIdx.x * blockDim.x;
    int y = threadIdx.y + blockIdx.y * blockDim.y;

    if (x < src.cols && y < src.rows) {
        uchar3 v = src(y, x); // Reads pixel in GPU memory. Valid! We are on GPU!
        dst(y, x) = make_uchar3(v.z, v.y, v.x);
    }
}

void swap_rb_caller(const PtrStepSz<uchar3>& src, PtrStep<uchar3> dst, cudaStream_t stream) {
    dim3 block(32, 8);
    dim3 grid((src.cols + block.x - 1)/block.x, (src.rows + block.y - 1)/block.y);

    swap_rb_kernel<<<grid, block, 0, stream>>>(src, dst);

    if (stream == 0)
        cudaDeviceSynchronize();
}
```
Writing your own CUDA code

• Device layer functions (the set will be extended in OpenCV 3.0)
  – opencv2/gpu/device/*.hpp
  – useful utility code to call from your kernels

```cpp
#define HIST_SIZE 396
#include <opencv2/gpu/device/block.hpp>
__global__ void histogramm_kernel_pass1(const PtrStepSz<float3> cloud, PtrStep<float> output) {
    __shared__ float smem[HIST_SIZE];
    Block::fill(smem, smem + HIST_SIZE, 0);
    __syncthreads();
    ... compute histogram using atomics on smem ...
    __syncthreads();
    float sum = Block::reduce_n(smem, HIST_SIZE, plus<float<?>>);
    __syncthreads();
    float* output_row = output(blockIdx.y * BLOCK_SIZE + blockIdx.x);
    Block::transform(smem, smem + HIST_SIZE, output_row, divide_by(sum));
}
```
Writing your own CUDA code

- Device layer functions (the set will be extended in OpenCV 3.0)
  - opencv2/gpu/device/*.hpp
  - useful utility code to call from your kernels

```cpp
#define HIST_SIZE 396
#include <opencv2/gpu/device/block.hpp>
__global__ void histogramm_kernel_pass1(const PtrStepSz<float3> cloud, PtrStep<float> output)
{
    __shared__ float smem[HIST_SIZE];
    Block::fill(smem, smem + HIST_SIZE, 0);
    __syncthreads();
    // ... compute histogram using atomics on smem ...
    __syncthreads();
    float sum = Block::reduce_n(smem, HIST_SIZE, plus<float>());
    __syncthreads();
    float* output_row = output(blockIdx.y * BLOCK_SIZE + blockIdx.x);
    Block::transform(smem, smem + HIST_SIZE, output_row, divide_by(sum));
}
```

```cpp
template <typename T, class BinOp>
static __device__ __forceinline__
void reduce_n(T* data, unsigned int n, BinOp op)
{
    int ftid = threadIdx.z * blockDim.x * blockDim.y;
    int sft = blockDim.x * blockDim.y * blockDim.z;
    if (sft < n)
    {
        for (unsigned int i = sft + ftid; i < n; i += sft)
            data[ftid] = op(data[ftid], data[i]);
        __syncthreads();
        n = sft;
    }
    while (n > 1)
    {
        unsigned int half = n/2;
        if (ftid < half)
            data[ftid] = op(data[ftid], data[n - ftid - 1]);
        __syncthreads();
        n = n - half;
    }
}
```
OpenCV resources

• Main site:
  – http://opencv.org/

• Online documentation (with tutorials)
  – http://docs.opencv.org/

• Question & Answers site
  – http://answers.opencv.org/questions/

• Books
  – http://opencv.org/books.html

• Bug tracker
  – http://code.opencv.org/projects/opencv/issues

“Learning OpenCV” 2nd ed. by G.Bradski, A.Kaehler with GPU module chapter coming soon!
Questions?

www.opencv.org
Upcoming GTC Express Webinars

July 31 - NMath Premium: GPU-Accelerated Math Libraries for .NET
August 7 - Accelerating High Performance Computing with GPUDirect RDMA
August 13 - GPUs in the Film Visual Effects Pipeline
August 14 - Beyond Real-time Video Surveillance Analytics with GPUs

Register at www.gputechconf.com/gtcexpress
GTC 2014 Call for Submissions

Looking for submissions in the fields of

- Science and research
- Professional graphics
- Mobile computing
- Automotive applications
- Game development
- Cloud computing

Submit at www.gputechconf.com
GeoInt Accelerator
- Platform to start developing with GPUs
- GPU access, GPU-accelerated apps and libraries
- Register to learn more: http://goo.gl/Eui6k6

Webinar Feedback
Submit your feedback for a chance to win Tesla K20 GPU*
https://www.surveymonkey.com/s/OpenCV_July30
* Offer is valid until August 13th 2013

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
- OpenCV.org: http://answers.opencv.org/questions/