

OpenCV & CUDA

Presented by:

Ángel R. Aranda Campos

Francisco J. Hernández López.

Jorge F. Madrigal Díaz

{arac, fcoj23, pacomd}@cimat.mx

OpenCV

Outline

- OpenCV 2.X
- Install
- OpenCV modules
- Drawing Primitives
- Basic Structures
- Image management
 - Pixel access
 - Browse a Picture
- Matrix Operation
- Histograms
- Homographies y Geometric transforms
- Video

OpenCV 2.X

- Library of algorithms released under BSD license.
- Interfaces with C++, C, Python and soon JAVA.
- Can be compiled on Windows, Linux, Android and Mac.
- Has more than 2500 optimized algorithms.
- Support by a big community of users and developers.
- Multiple uses like visual inspection, robotic, etc.

OpenCV Installation

- <http://opencv.willowgarage.com/wiki/>
- <http://www.cmake.org/>

OpenCV Overview: > 500 functions
opencv.willowgarage.com

Robot support

General Image Processing Functions

Image Pyramids

Geometric descriptors

Segmentation

Camera calibration, Stereo, 3D

Features

Utilities and Data Structures

Transforms

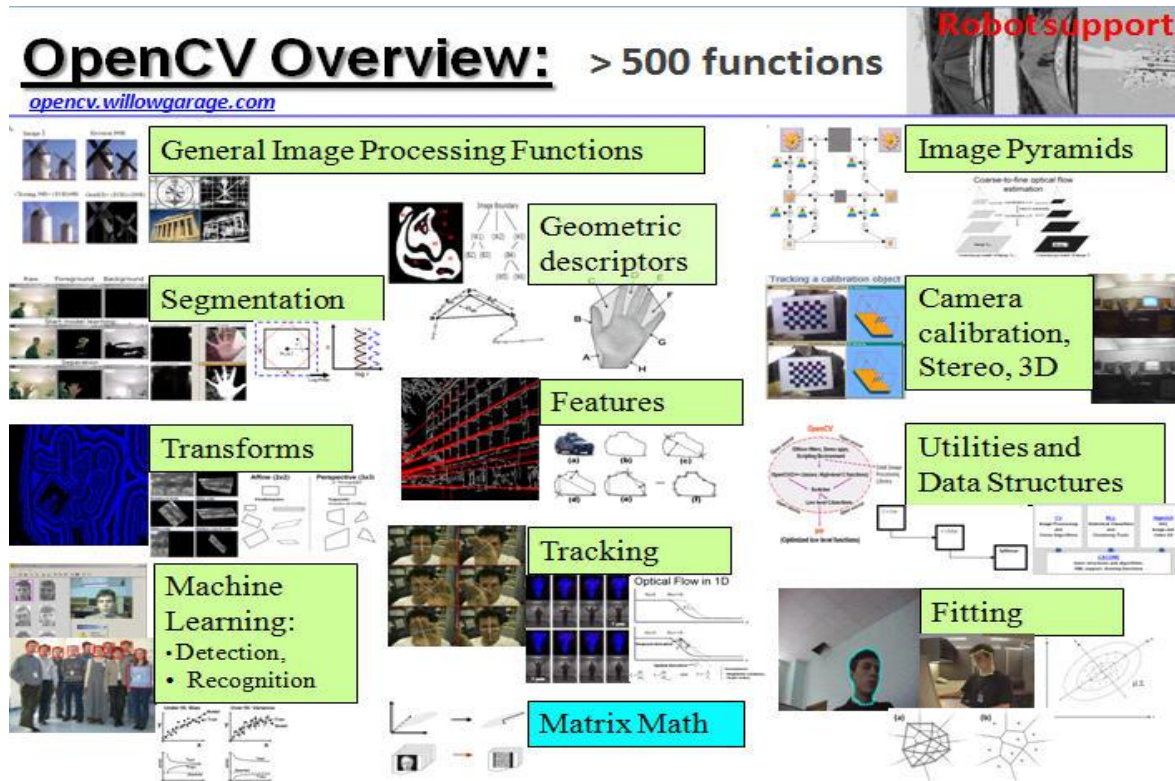
Tracking

Machine Learning:

- Detection,
- Recognition

Fitting

Matrix Math



OpenCV modules

- **Imgproc:** Main functions for image processing.
- **Highgui:** Reading and writing of images and videos, also functions for GUI.
- **Features2d:** Detectors of interest points, descriptors.
- **Calib3d:** Camera calibration, geometry of two views and stereo functions.
- **Video:** Estimation of motion, tracking and background subtraction.

OpenCV modules

- **Objdetect:** Object detection functions (e.g. people).
- **Ml:** Machine learning functions.
- **Flann:** Computational geometry algorithms.
- **Contrib:** Miscellaneous contributions
- **Legacy:** Deprecated code
- **Gpu:** And more recently, GPUs functions (CUDA)

Drawing Primitives

- Line

```
void line(Mat& img, Point pt1, Point pt2, const Scalar& color,  
          int thickness=1, int lineType=8, int shift=0);
```

- Circle

```
void circle(Mat& img, Point center, int radius,  
            const Scalar& color, int thickness=1,  
            int lineType=8, int shift=0);
```

- Rectangle

```
void rectangle(Mat& img, Point pt1, Point pt2,  
               const Scalar& color, int thickness=1,  
               int lineType=8, int shift=0);
```

- etc

Basic Structures

- **cv :: Mat** and **cv :: Mat_**
 - Basic management of matrices

```
1 // make a 7x7 complex matrix filled with 1+3j.  
  Mat M(7,7,CV_32FC2,Scalar(1,3));  
3 // and now turn M to a 100x60  
  // 15-channel 8-bit matrix.  
5 // The old content will be deallocated  
  M.create(100,60,CV_8UC(15));
```

Basic Structures

- **cv :: Mat** and **cv :: Mat_**
 - Ensures correct memory release and implements reference counting and superficial copies, avoiding unnecessary memory creation.

```
// create a 100x100 8-bit matrix
2 Mat M(100,100,CV_8U);
// this will be compiled fine.
4 // No any data conversion will be done.
  Mat_<float>& M1 = (Mat_<float>&)M;
6 // the program is likely to crash
  // at the statement below
8 M1(99,99) = 1.f;
```

Basic Structures

- **cv::vector**
 - Similar to the classic std::vector
 - It is more recommendable for opencv objects
- **Example:**

```
cv::vector<cv::Mat> vec_mat;
```

Image management

- Class **cv::Mat** is responsible for managing the image and replaces the structure **IplImage** (versions < 2.0).

```
IplImage* img = 0;  
img=cvLoadImage("Image1.jpg");
```

```
Mat image;  
image = imread("Image1.jpg", CV_LOAD_IMAGE_COLOR);
```

- We can update our old OpenCV structures to the newest ones.

```
IplImage* iplImage = cvLoadImage("img.jpg");  
2 cv::Mat image4(iplImage, false);
```

Image management

```
1 #include <opencv2/core/core.hpp>
  #include <opencv2/highgui/highgui.hpp>
3
4 int main()
5 {
6
7     // read an image
8     cv::Mat image= cv::imread("img.jpg");
9
10    // create image window named "My Image"
11    cv::namedWindow("My_Image");
12
13    // show the image on window
14    cv::imshow("My_Image", image);
15
16    // wait key for 5000 ms
17    cv::waitKey(5000);
18
19    return 0;
20 }
```

- OpenCV provides functions for reading, showing and saving of images.

Image management

- **cv::Mat memory**
 - Is automatically released by its destructor.
 - Has also a member `release()`.

Image management

```
1 cv::Mat function() {  
    // create image  
3 cv::Mat ima(240,320,CV_8U,cv::Scalar(100));  
    // return it  
5 return ima;  
}
```

- In this case, *function* does not reserve additional memory

```
    // get a gray-level image  
2 cv::Mat gray= function();
```

Image management

- **Pixel access**

- There are different ways to access the pixels within an instance of `cv::Mat`. For example, for grayscale images, we can use the member function “.at<type>” (row,col)

```
1 image.at<uchar>(j , i)= value;
```

- En el caso de imágenes con tres canales

```
image.at<cv::Vec3b>(j , i)[channel]= value;
```

Image management

- **Browse an Image**

- There are several methods for browsing an image completely. Depending on the computation time required, different strategies can be implemented. In general, it makes use of the member function. `ptr <type> (row)`

Image management

- Browse an Image

```
1 // using .ptr and []
  void colorReduce0(cv::Mat &image, int div=64) {
3     int nl= image.rows; // number of lines
      // total number of elements per line
5     int nc= image.cols * image.channels();
      for (int j=0; j<nl; j++) {
7         uchar* data= image.ptr<uchar>(j);
          for (int i=0; i<nc; i++) {
9             // process each pixel
              data[i]= data[i]/div*div + div/2;
11            // end of pixel processing
          } // end of line
13    }
  }
```

Matrix Operations

- OpenCV has several functions for many operations: arithmetic, linear algebra, statistics, etc.. For example:
 - `cv::add`
 - `cv::addWeighted`
 - `cv::cartToPolar`
 - `cv::eigen`

Matrix Operations

- Different ways of doing things of applying matrix operations
- Through the explicit use of functions

```
cv::addWeighted( image1 , alpha , image2 ,  
2 beta , gamma , result );
```

- Overloaded operators

```
cv::Image result = alpha*image1 +  
2 beta*image2 + gamma;
```

Histograms

- CalcHist functions, calcBackProject, compareHist and equalizeHist provide us the functionalities needed to control histograms.

```
1 cv::MatND hist;  
  
3 // Compute histogram  
cv::calcHist(&image,  
5 1, // histogram of 1 image only  
channels, // the channel used  
7 cv::Mat(), // no mask is used  
hist, // the resulting histogram  
9 1, // it is a 1D histogram  
histSize, // number of bins  
11 ranges // pixel value range  
);
```

Histograms

- **CompareHist**

compareHist(InputArray **H1**, InputArray **H2**, int **method**)

- The variable **method** can be
 - CV_COMP_CORREL Correlation
 - CV_COMP_CHISQR Chi-Square
 - CV_COMP_INTERSECT Intersection
 - CV_COMP_BHATTACHARYYA Bhattacharyya distance

Homographies and Geometric transforms

- There are several algorithms for calculating homographies, fundamental matrix or various geometric transformations. In general, these algorithms are based on matchings between a pair of images.
- OpenCV provides a generic class to use different descriptors such as:
 - FAST
 - MSER
 - SIFT
 - SURF
 - BRIEF
 - ORB

Homographies and Geometric transforms

- And it provides a function (`cv::findHomography`) that given a set of matchings between a pair of images, estimates a homography based on two possible algorithms.
 - RANSAC
 - Least-Median Square

Video

- To capture and save videos. OpenCV provides the class:
 - **cv::VideoCapture**. This class has the overload of different operators which make the code more intuitive and readable.
 - **cv::VideoWriter**. This class is used for saving videos.

Video

```
using namespace cv;
2
int main(int , char**)
4 {
    // open the default camera
6    VideoCapture cap(0);
    // check if we succeeded
8    if(!cap.isOpened())
        return -1;
10    Mat edges;
    namedWindow(" edges" ,1);
12    for (;;)
    {
14        Mat frame;
        cap >> frame;
16        cvtColor(frame, edges, CV_BGR2GRAY);
        GaussianBlur(edges, edges, Size(7,7), 1.5,
18        Canny(edges, edges, 0, 30, 3);
        imshow(" edges", edges);
20        if(waitKey(30) >= 0) break;
    }
22    // the camera will be deinitialized automatically
    // in VideoCapture destructor
24    return 0;
}
```

Common Tasks

- Image filtering
- Stereo Matching
- Morphology
- HOG
- Segmentation
- Etc.
- **All Highly Parallelizable**



Questions?



CUDA

Outline

- Parallel Computing
- Motivation
- GPU
- CUDA
- Programming Model
- Installing CUDA
- Examples

Parallel Computing

- Running more than one calculation at the same time or "in parallel", using more than one processor.



OpenMP



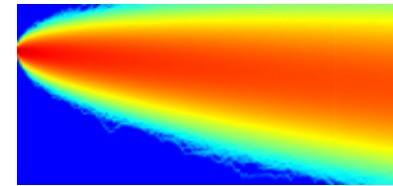
OpenMPI



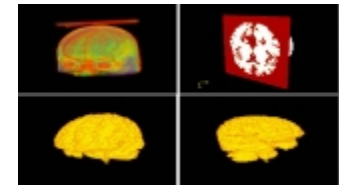
Cg,
CUDA,
OpenCL

Motivation

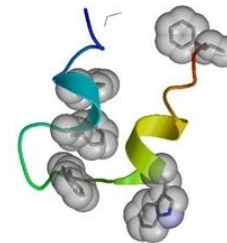
- You can solve problems:
 - Finance.
 - Graphics.
 - Image processing and Video.
 - Linear Algebra.
 - Physics.
 - Chemistry.
 - Biology.
 - Etc....



Differential Eq.



Medial Image Segmentation

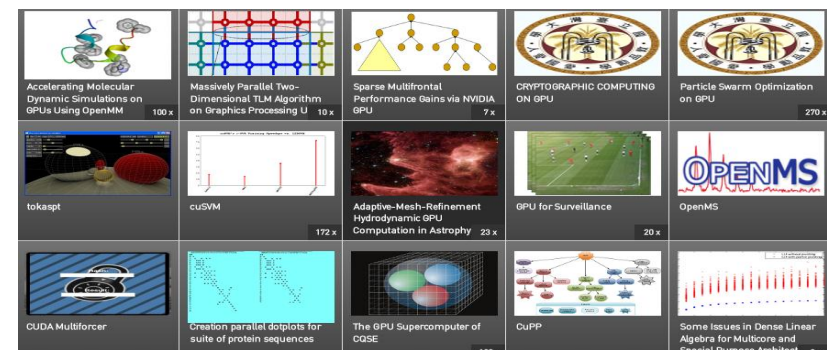


Molecular dynamics



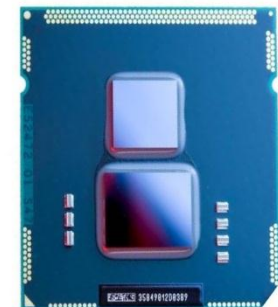
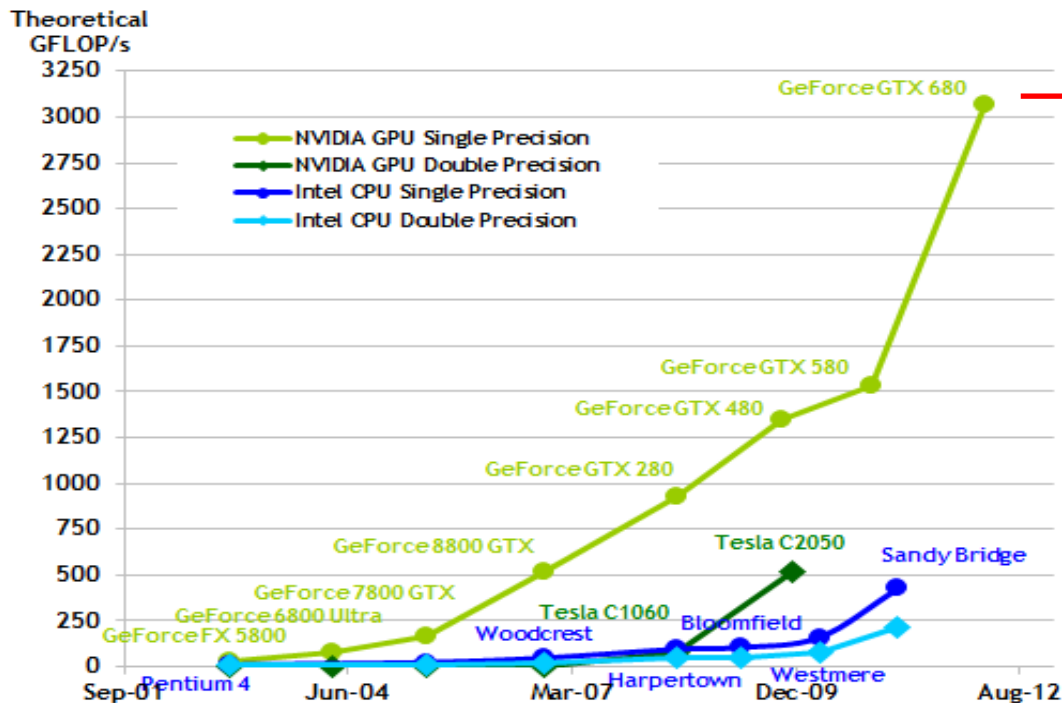
Object detection

CUDA ZONE



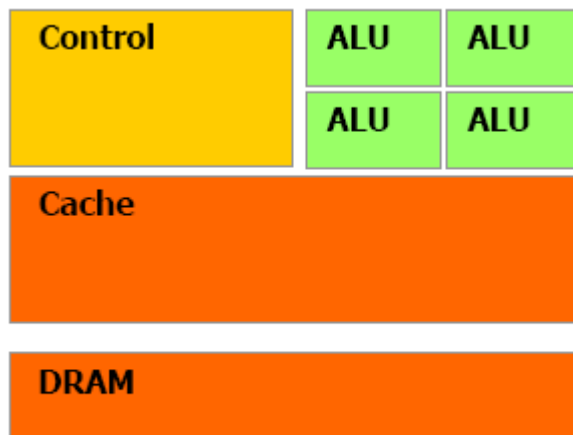
GPU

- Flexible and powerful Processor .
- Handles accuracy of (32/64)-bit in floating point.
- Programmed using high level languages.
- Offers lots of GFLOPS.

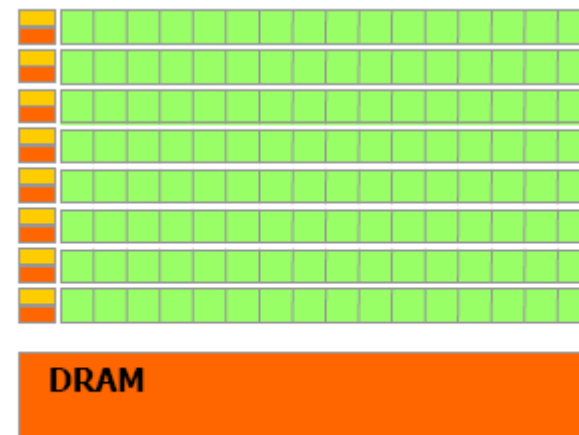


GPU

- Specialized for data parallel computing.
- Uses more transistors to data processing than flow control or data storage.



CPU



GPU

CUDA (Compute Unified Device Architecture)

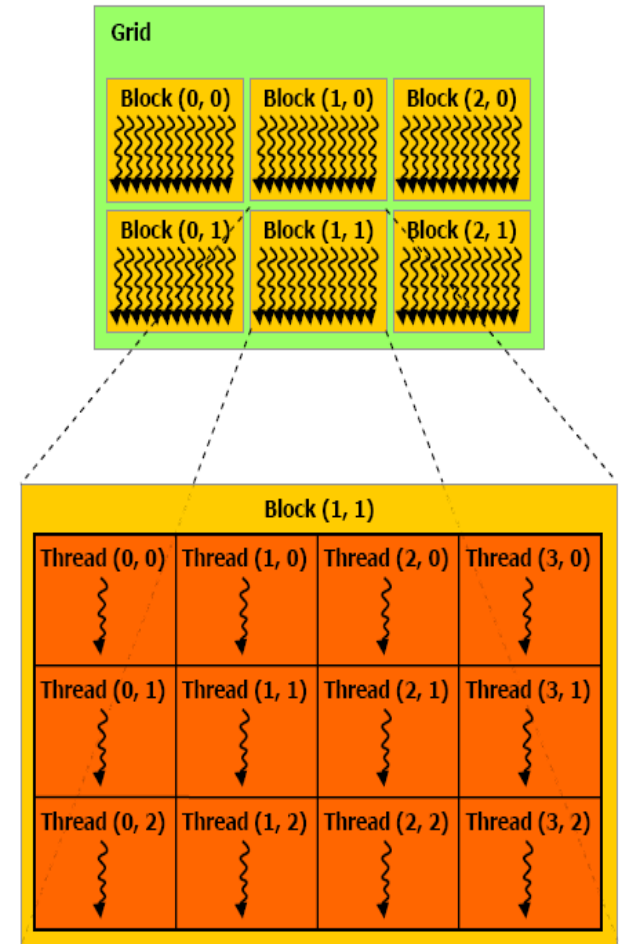
- GPGPU technology (General-purpose computing on graphics processing units) that lets you use the C programming language to execute code on the graphic processing unit (GPU).
- Developed by NVIDIA.
- To use this architecture it is required to have a GeForce 8 series (or Quadro equivalent), and more recently CPUs.

CUDA Features

- Supports the programming language C/C++, Fortran, Matlab, LabView, etc..
- Unification of hardware and software for parallel computing.
- Supports single instruction, multiple data (SIMD).
- Libraries for FFT (Fast Fourier Transform), BLAS (Basic Linear Algebra Subroutines), NPP, TRUSTH, CULA, etc.
- Works internally with OpenGL and DirectX.
- Supports operative systems:
 - Windows, Linux and Mac OS.

Programming Model

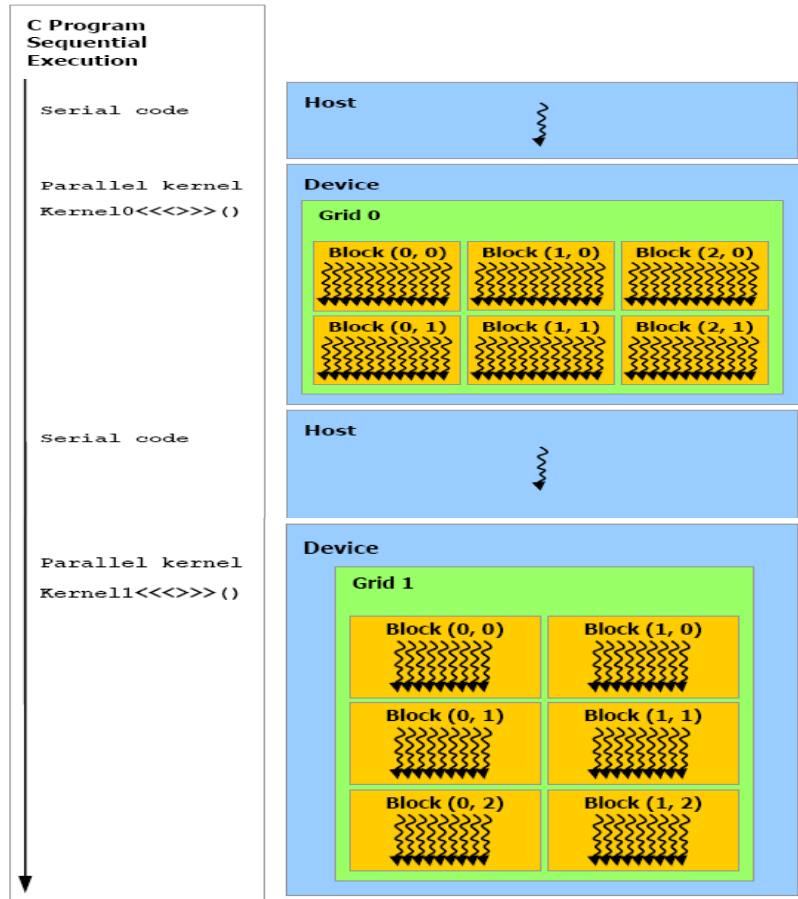
- A program that is compiled to run on a graphics card is called the *Kernel*.
- The set of threads that execute a kernel is organized as a **grid** of thread blocks.
- A thread block is a set of threads that can cooperate together:
 - Easy access to shared memory.
 - Synchronously.
 - With a thread identifier ID.
 - Blocks can be arranged for 1, 2 or 3 dimensions.
- A grid of thread blocks:
 - It has a limited number of threads in a block.
 - The blocks are identified by an ID.
 - Arrangements can be of 1 or 2 dimensions.



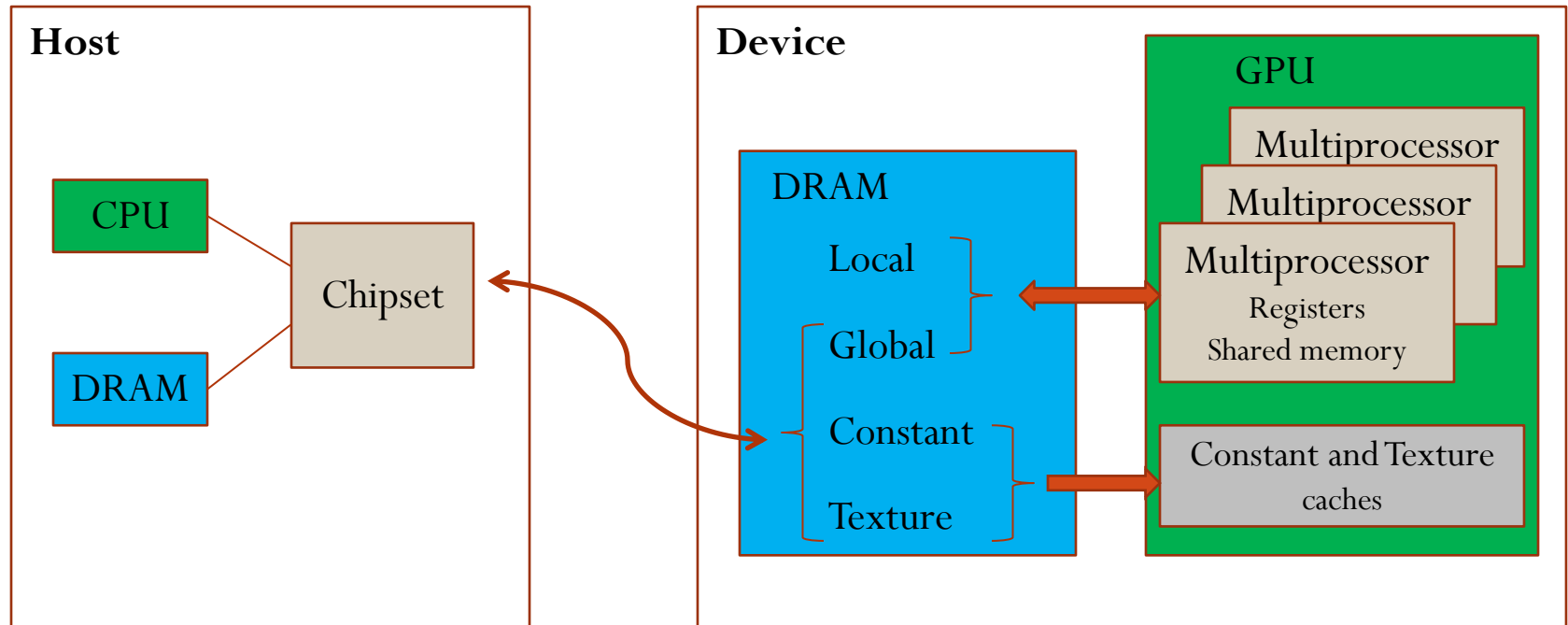
Programming Model

- Running on the Host and Device.

Host = CPU
Device = GPU
Kernel = Set of instructions that runs in the device



Memory model



Memory management

- Allocate and free memory
 - **cudaMalloc** ((void**) devPtr, size_t size)
 - **cudaFree** (void *devPtr)

Memory management

- Copy memory.
 - **cudaMemcpy**(void *dst, const void *src, size_t count, enum cudaMemcpyKind **kind**)

Kind:

- cudaMemcpyHostToHost
- cudaMemcpyHostToDevice
- cudaMemcpyDeviceToHost
- cudaMemcpyDeviceToDevice

Qualifiers for a function

- **__device__**
 - Runs on the device.
 - Called only from the device.
- **__global__**
 - Runs on the device
 - Called only from the host.

Qualifiers for a variable

- **__device__**
 - Resides in global memory space.
 - Has the lifetime of an application.
 - Lives accessible from all threads within the grid, and from the host through the library at runtime.
- **Others:**
 - **__constant__** (Optionally used with **__device__**)
 - Resides in constant memory space.
 - Has the lifetime of an application.
 - Lives accessible from all threads within the grid, and from the host through the library at runtime.
 - **__shared__** (Optionally used with **__device__**)
 - Lives in shared memory space of a thread block.
 - Has the lifetime of a block.
 - Only accessible from the threads that are within the block.

Kernel functions calls

- Example function

```
__global__ void NameFunc(float *parameter, ...);
```

it must be called as follows:

```
NameFunc <<< Dg, Db, Ns, St >>> (parameter1, ...);
```

- **Dg**: Type *dim3*, dimension and size of the grid.
- **Db**: Type *dim3*, dimension and size of each block.
- **Ns**: Type *size_t*, number of bytes in shared memory.
- **St**: Type *cudaStream_t* that indicates which stream will use the kernel.
(Ns and St are optional).

Automatically Defined Variables

- All `__global__` and `__device__` functions have access to the following variables:
 - **gridDim** (dim3), indicates the dimension of the grid.
 - **blockIdx** (uint3), indicates the index of the bloque within the grid.
 - **blockDim** (dim3), indicates the dimension of the block.
 - **threadIdx** (uint3), indicates the index of the thread within the block.

Example

CPU C

```
void add_matrix_cpu(float *a, float *b, float *c,
    int N)
{
    int i, j, index;
    for (i=0; i<N; i++) {
        for (j=0; j<N; j++) {
            index = i+j*N;
            c[index] = a[index] + b[index];
        }
    }
}

void main() {
    ....
    add_matrix(a, b, c, N);
}
```

CUDA C

```
__global__ void add_matrix_gpu(float *a, float *b,
    float *c, int N)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    int index = i + j * N;
    if (i < N && j < N)
        c[index] = a[index] + b[index];
}

void main() {
    dim3 dimBlock(blocksize, blocksize);
    dim3 dimGrid(N / dimBlock.x, N / dimBlock.y);
    add_matrix_gpu<<<dimGrid,
    dimBlock>>>(a, b, c, N);
}
```

CUDA-Enabled Graphic Cards



http://www.nvidia.com/object/cuda_gpus.html

| Architectures | Capability |
|---------------------|------------|
| 8-200 series | 1.0-1.3 |
| FERMI (400 series) | 2.0-2.1 |
| KEPLER (600 series) | 3.0-3.5 |

GPU Architectures and Capability

CUDA-Enabled Graphic Cards

GeForce 8400 GS, C=1.1



Quadro FX 1700, C=1.1



GeForce 8800 GT, C=1.1



TESLA C1060, C=1.3



GeForce GT 640, C=2.1



GeForce GTX 480,
C=2.0



GTX > GTS > GT > GS

Installing CUDA

- Installing CUDA (Driver, Toolkit y SDK).
- <http://developer.nvidia.com/cuda/cuda-downloads>

CUDA DOWNLOADS

CUDA TOOLKIT 4.2
CURRENT PRODUCTION RELEASE

CUDA 5
RELEASE CANDIDATE
[Learn More...](#)

CUDA 4.2 FOR WINDOWS

| 1 Download Toolkit | 2 Download Drivers (v 301.32 or 301.27) | | | 3 Download SDK |
|--------------------|---|---------------------|----------------|----------------|
| | Win7/Vista Desktop | Win7/Vista Notebook | Win XP Desktop | |
| 64bit | 64bit | 64bit | 64bit | 64bit |
| 32bit | 32bit | 32bit | 32bit | 32bit |

CUDA 4.2 FOR LINUX

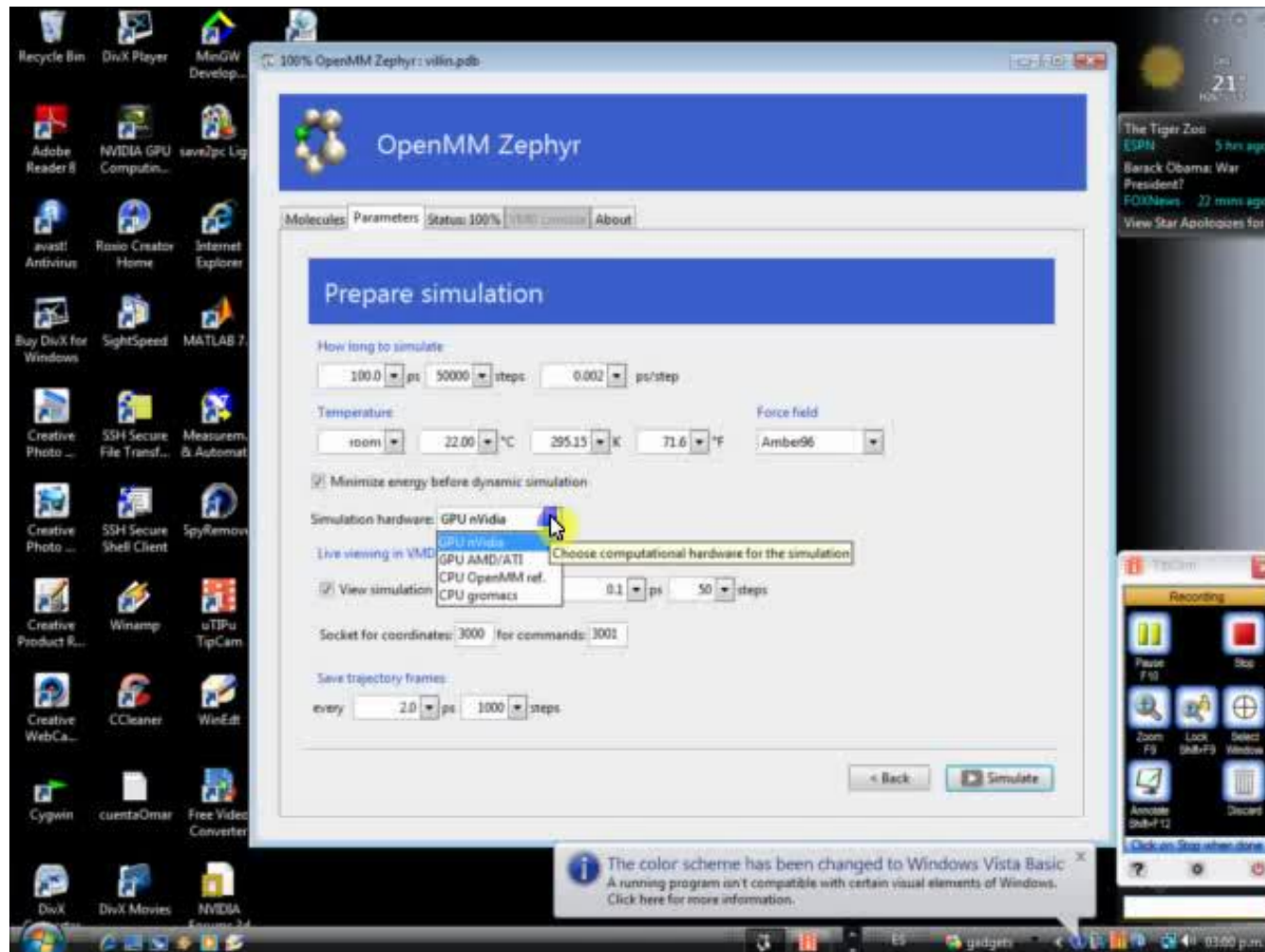
| 1 Download Toolkit | Fedora 14 | Redhat 5.5 | Redhat 6.0 | Ubuntu 11.04 | Ubuntu 10.04 | OpenSUSE 11.2 | SUSE Server 11 SP1 |
|--------------------|-----------|------------|------------|--------------|--------------|---------------|--------------------|
| 64bit | 64bit | 64bit | 64bit | 64bit | 64bit | 64bit | 64bit |
| 32bit | 32bit | 32bit | | 32bit | 32bit | 32bit | 32bit |

| 2 Download Drivers (ver 295.41) | 3 Download SDK |
|---------------------------------|----------------|
| 64bit 32bit | DOWNLOAD |

CUDA 4.2 FOR MAC

| 1 Download Toolkit | 2 Download Drivers (updated 06.11.12) | 3 Download SDK |
|--------------------|---------------------------------------|----------------|
| DOWNLOAD | DOWNLOAD | DOWNLOAD |

Examples

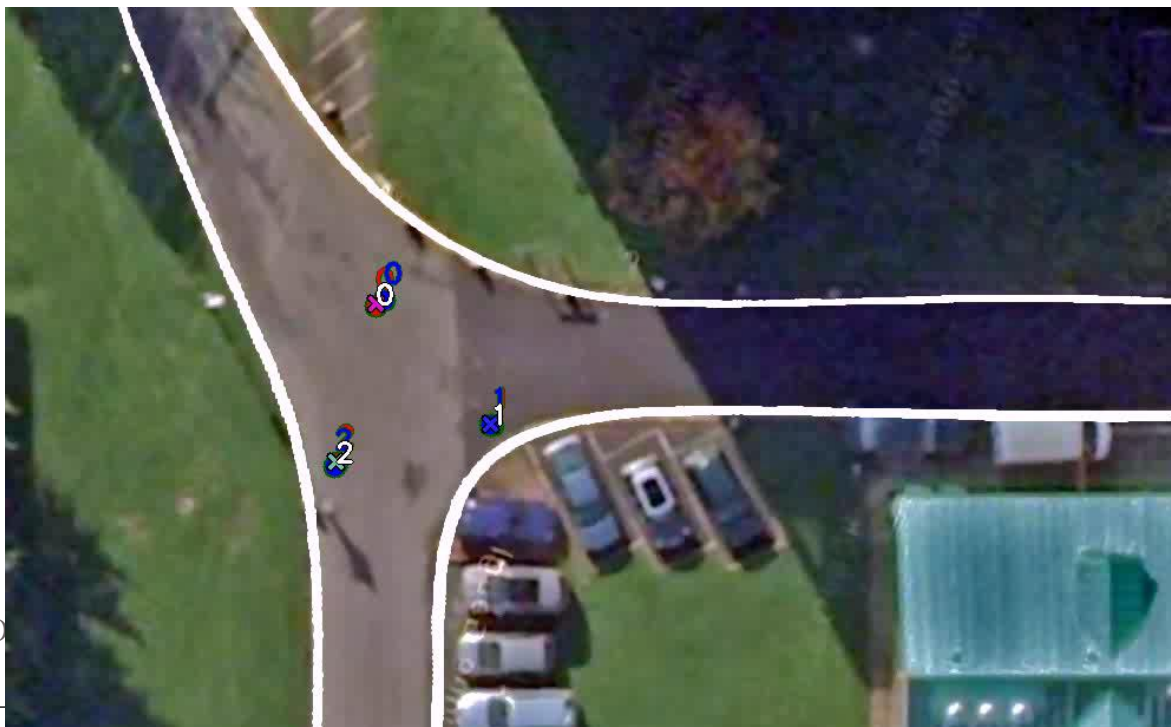


Examples – OpenCV & CUDA

- **GPU-ACCELERATED COMPUTER VISION**

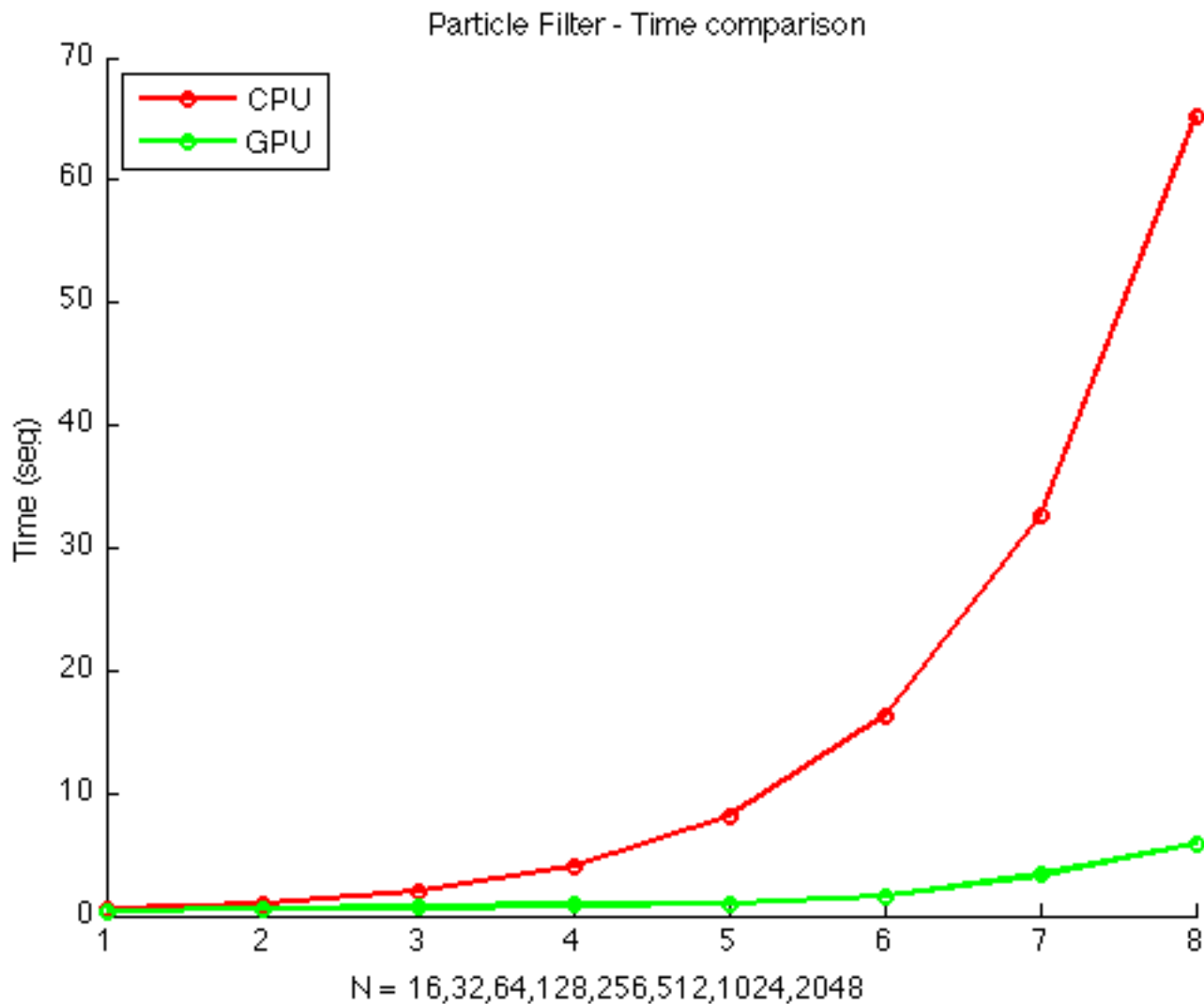


Examples – OpenCV & CUDA



Tracking

Examples – OpenCV & CUDA



Tracking

Examples – OpenCV & CUDA

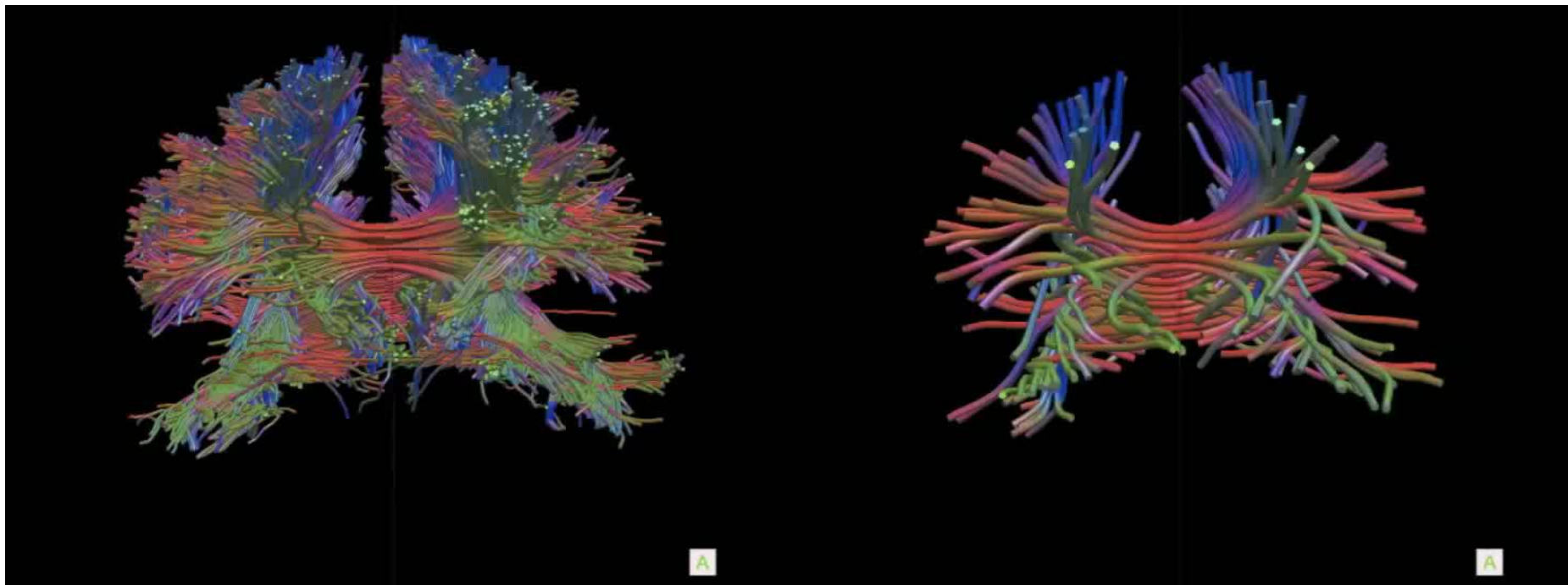


VSreen



Examples – OpenCV & CUDA

Tractography



Tract Estimations from the callosum corpus

Questions?



Thank you!