NVIDIA VISIONWORKS TOOLKIT
Frank Brill, Elif Albuz
VISIONWORKS MOTIVATION

Advanced Silicon + Simplify vision programming
Fully optimized and accelerated
Modular and Extensible = Widespread vision processing in embedded, mobile and automotive devices and applications
OUR APPROACH

- Open Platform & Easy to Program
- Multi-Camera based complex systems
- Ecosystem Leverage
VISIONWORKS

Power Efficient Computer Vision Powered with CUDA

Supported on Tegra K1 Linux and Android

ACCELERATING
• Advanced Driver Assistance
• Computational Photography
  • Augmented Reality
  • Robotics
  • Deep Learning
  and more...

Version 0.10 is available for registered partners!
VISIONWORKS SOFTWARE STACK

Application Code

Vision Pipeline Samples
- Object Detection
- SLAM

VisionWorks Primitives
- Classifier
- Corner Detection
- 3rd Party

3rd Party Pipelines

VisionWorks Framework
- OpenVX

CUDA Libraries

Tegra K1
TEGRA K1
192-core Super Chip
TEGRA K1: A MAJOR LEAP FORWARD FOR MOBILE & EMBEDDED APPLICATIONS

KEPLER GPU, 192 CORES,
>300GFLOPS
CUDA
12GB/S BANDWIDTH
VIDEO IMAGE COMPOSITOR (VIC)
TEGRA K1 DEVELOPMENT PLATFORMS

**JETSON TK1**
gigE, usb3.0, HDMI
running Linux4Tegra

**JETSON X3 (TK1 PRO)**
gigE, usb3.0, HDMI, CANBUS
running Vibrante Linux
AUTOMOTIVE GRADE

Coming to Android K1 Devices soon..
TEGRA K1 CUDA DEVELOPMENT

**CUDA-Aware Editor**
- Automated CPU to GPU code refactoring
- Semantic highlighting of CUDA code
- Integrated code samples & docs

**Nsight Debugger**
- Simultaneously debug of CPU and GPU
- Inspect variables across CUDA threads
- Use breakpoints & single-step debugging

**Nsight Profiler**
- Quickly identifies performance issues
- Integrated expert system
- Source line correlation

Cross platform development
Native memcheck, GDB, nvprof
CUDA LIBRARIES

VisionWorks

NPP

OpenCV

CUBLAS

CUFFT

CUDA Math Lib
OPENVX - POWER EFFICIENT VISION ACCELERATION

- Khronos, open, cross-vendor vision API
  - Focus on mobile and embedded systems
- Foundational API for vision acceleration
  - Useful for middleware or by applications
  - Enables diverse efficient implementations
- Complementary to OpenCV
  - Which is great for prototyping
OPENVX GRAPHS - THE KEY TO EFFICIENCY

- Directed graphs for processing power and efficiency
  - Each Node can be implemented in software or accelerated hardware
  - Nodes may be fused to eliminate memory transfers
  - Processing can be tiled to keep data entirely in local memory/cache

- EGLStreams route data from camera and to application

- Can extend with “VisionWorks” nodes using CUDA

Example OpenVX Graph
## OpenVX and CUDA are Complementary

<table>
<thead>
<tr>
<th>Use Case</th>
<th>GPGPU Programming</th>
<th>Domain targeted Vision processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Language-based</td>
<td>Library-based - no separate compiler required</td>
</tr>
<tr>
<td>Target Hardware</td>
<td>‘Exposed’ architected memory model – programmer manages memory</td>
<td>Abstracted node and memory model - diverse implementations can be optimized for power and performance</td>
</tr>
<tr>
<td>Precision</td>
<td>Full IEEE floating point mandated</td>
<td>Minimal floating point requirements – optimized for vision operators</td>
</tr>
<tr>
<td>Ease of Use</td>
<td>General-purpose math and other libraries</td>
<td>Fully implemented vision operators and framework ‘out of the box’</td>
</tr>
</tbody>
</table>

It is possible to use CUDA to build OpenVX Nodes
OpenVX and OpenCV are Complementary

<table>
<thead>
<tr>
<th>Governance</th>
<th>VisionWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community driven open source with no formal specification</td>
<td>Defined and implemented by NVIDIA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Portability</th>
<th>VisionWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>APIs can vary depending on processor</td>
<td>Tegra K1 mobile platforms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Scope</th>
<th>VisionWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very wide</td>
<td>Tight focus on hardware accelerated functions for mobile vision</td>
</tr>
<tr>
<td>1000s of imaging and vision functions</td>
<td>Use external camera API</td>
</tr>
<tr>
<td>Multiple camera APIs/interfaces</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Efficiency</th>
<th>VisionWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory-based architecture</td>
<td>Graph-based execution</td>
</tr>
<tr>
<td>Each operation reads and writes memory</td>
<td>Optimizable computation, data transfer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use Case</th>
<th>VisionWorks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rapid experimentation</td>
<td>Production development &amp; deployment</td>
</tr>
</tbody>
</table>
VISIONWORKS SOFTWARE STACK

Application Code

Vision Pipeline Samples
- Object Detection
- SLAM

VisionWorks Primitives
- Classifier
- Corner Detection

3rd Party Pipelines

VisionWorks Framework

CUDA Libraries

Tegra K1
<table>
<thead>
<tr>
<th>VisionWorks Primitives - Jan 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sobel</td>
</tr>
<tr>
<td>Convolve</td>
</tr>
<tr>
<td>Bilateral Filter</td>
</tr>
<tr>
<td>Integral Image</td>
</tr>
<tr>
<td>Integral Histogram</td>
</tr>
<tr>
<td>Corner Harris</td>
</tr>
<tr>
<td>Corner FAST</td>
</tr>
<tr>
<td>Image Pyramid</td>
</tr>
<tr>
<td>Optical Flow PyrLK</td>
</tr>
<tr>
<td>Optical Flow Farneback</td>
</tr>
<tr>
<td>Warp Perspective</td>
</tr>
<tr>
<td>Hough Lines</td>
</tr>
<tr>
<td>Fast NLM Denoising</td>
</tr>
<tr>
<td>Stereo Block Matching</td>
</tr>
<tr>
<td>IME (Iterative Motion Estimation)</td>
</tr>
<tr>
<td>Soft Cascade Detector</td>
</tr>
<tr>
<td>Object Tracker</td>
</tr>
<tr>
<td>TLD Object Tracker</td>
</tr>
<tr>
<td>SLAM</td>
</tr>
<tr>
<td>Path Estimator</td>
</tr>
<tr>
<td>MedianFlow Estimator</td>
</tr>
</tbody>
</table>
## OPENVX-BASED PRIMITIVES

| Absolute Difference | Accumulate | Accumulate Squared | Accumulate Weighted | Arithmetic Addition | Arithmetic Subtraction | Bitwise And | Bitwise Exclusive Or | Bitwise Inclusive Or | Bitwise Not | Box Filter | Canny Edge Detector | Channel Combine | Channel Extract | Color Convert | Convert Bit depth | Dilate Image | Equalize Histogram | Erode Image | Gaussian Filter | Histogram | Image Pyramid | Magnitude | Mean and Standard Deviation | Median Filter | Min, Max Location | Optical Flow Pyramid (LK) | Phase | Pixel-wise Multiplication | Remap | Scale Image | TableLookup | Thresholding | Warp Affine |
VISIONWORKS PIPELINES (V0.10)

- Structure From Motion/SLAM
- Pedestrian Detection
- Vehicle detection
- Object tracking
- Dense optical flow
- Active Shape Model
- Denoising
STRUCTURE FROM MOTION (SFM) BENCHMARKING

Speedup (x) GPU vs ARM code on T124*

- Image Pyramid: 8.8
- FastCorner Detection: 21.05
- Harris Corner Detection: 84.04
- Optical Flow: 21.25
FEATURE TRACKING EXAMPLE

- OpenCV Framesource
- Find VisionWorks
- Harris corners
- Track VisionWorks
- Pyramidal Lucas-Kanade
- Display OpenCV
  imshow
typedef struct tracker_t {

    // Context and image size
    vx_context context;
    vx_uint32 width;
    vx_uint32 height;

    // Parameters for PyrLK
    vx_size pyr_levels;
    vx_float32 harris_k;
    vx_float32 harris_thresh;
    vx_uint32 lk_num_iters;
    vx_size lk_win_size;

    // Where to run
    vx_target cuda_target;

    // Graph and nodes
    vx_graph graph;
    vx_node cvt_color_node;
    vx_node ch_extract_node;
    vx_node pyr_node;
    vx_node opt_flow_node;
    vx_node feature_track_node;

    // Delays for differential processing
    vx_delay pyr_delay;
    vx_delay pts_delay;

    // The currently tracked list of features
    vx_list curr_features;
} tracker_t;
int main(int argc, const char* argv[]) {
    tracker_t trk;

    // Open input video source file
    const std::string videoFile = "cars.avi";
    cv::VideoCapture cap;
    if (!cap.open(videoFile)) {
        std::cerr << "Can't open video " << videoFile << std::endl;
        return -1;
    }

    // Create OpenVX context
    trk.context = vxCreateContext();

    // Create OpenVX Image to hold frames from video source
    trk.width = cap.get(CV_CAP_PROP_FRAME_WIDTH);
    trk.height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
    vx_image frame = vxCreateImage(trk.context, trk.width, trk.height, FOURCC_RGB);
    vx_rectangle frame_rect = vxGetValidRegionImage(frame);

    // Read first frame from video sequence
    cv::vx::grab(cap, frame, frame_rect);

    // Set the parameters
    trk.pyr_levels = 6;
    trk.harris_k = 0.04;
    trk.harris_thresh = 100.0;
    trk.lk_num_iters = 5;
    trk.lk_win_size = 10;

    // Initialize tracking
    createTrackerGraph(frame, trk);
    processFirstFrame(frame, trk);

    // RUN MAIN PROCESSING LOOP
    for (;;) {
        // Grab next frame
        if (cv::vx::grab(cap, frame, frame_rect) != VX_SUCCESS) {
            // re-open video file
            cap.open(videoFile);
            cv::vx::grab(cap, frame, frame_rect);
        }

        // Update frame queue
        vxAgeDelay(trk.pyr_delay);
        vxAgeDelay(trk.pts_delay);

        // Process graph
        vxSetParameterByIndex(trk.cvt_color_node, 0, (vx_reference)frame);
        vxProcessGraph(trk.graph);

        // Map OpenVX Image to OpenCV Mat, draw arrows & state
        // and show the resulting image
        vx_imagepatch_addressing_t addr;
        cv::Mat cv_frame;
        cv::vx::accessImagePatch(frame, frame_rect, 0, &addr, cv_frame);
        drawArrows(cv_frame, getPrevFeatures(trk), trk.curr_features);
        cv::imshow("Feature Tracker Demo", cv_frame);

        cv::vx::commitImagePatch(frame, 0, 0, &addr, cv_frame);
    }
// See if it's time to stop
const int key = cv::waitKey(5) & 0xff;
if (key == 27) { // escape
  break;
}

// Release all objects
vxReleaseDelay(&trk.pyr_delay);
vxReleaseDelay(&trk.pts_delay);
vxReleaseList(&trk.curr_features);
vxReleaseGraph(&trk.graph);
vxReleaseNode(&trk.cvt_color_node);
vxReleaseNode(&trk.ch_extract_node);
vxReleaseNode(&trk.pyr_node);
vxReleaseNode(&trk.opt_flow_node);
vxReleaseNode(&trk.feature_track_node);
vxReleaseTarget(&trk.cuda_target);
vxReleaseContext(&trk.context);
vxReleaseRectangle(&frame_rect);
vxReleaseImage(&frame);
return 0;
void processFirstFrame(vx_image frameRGB, tracker_t &trk) {

vx_image frameYUV = vxCreateImage(trk.context, trk.width, trk.height, FOURCC_IYUV);
vx_image frameGray = vxCreateImage(trk.context, trk.width, trk.height, FOURCC_U8);

vxuColorConvert(trk.context, frameRGB, frameYUV);
vxuChannelExtract(trk.context, frameYUV, VX_CHANNEL_Y, frameGray);
vxuPyramid(trk.context, frameGray, (vx_pyramid)vxGetReferenceFromDelay(trk.pyr_delay, 1), 0);
vxnvuHarrisTrack(frameGray, (vx_list)vxGetReferenceFromDelay(trk.pts_delay, 1), 0, 0,
    trk.harris_k, trk.harris_thresh);
vxReleaseImage(&frameYUV);
vxReleaseImage(&frameGray);
}
void createTrackerGraph(vx_image frameRGB, tracker_t &trk) {
    trk.graph = vxCreateGraph(trk.context);
    trk.cuda_target = vxGetTargetByName(trk.context, "nvidia.cuda");

    // Create color convert node
    vx_image frameYUV = vxCreateVirtualImage(trk.graph, trk.width, trk.height, FOURCC_IYUV);
    trk.cvt_color_node = vxColorConvertNode(trk.graph, frameRGB, frameYUV);

    // Create channel extract node
    vx_image frameGray = vxCreateVirtualImage(trk.graph, trk.width, trk.height, FOURCC_U8);
    trk.ch_extract_node = vxChannelExtractNode(trk.graph, frameYUV, VX_CHANNEL_Y, frameGray);

    // Create pyramid node
    vx_pyramid pyr_sample = vxCreatePyramid(trk.context, trk.pyr_levels, VX_SCALE_PYRAMID_HALF, trk.width, trk.height, FOURCC_U8);
    trk.pyr_delay = vxCreateDelay(trk.context, (vx_reference)pyr_sample, 2);
    vxReleasePyramid(&pyr_sample);
    trk.pyr_node = vxPyramidNode(trk.graph, frameGray, VX_pyramid)vxGetReferenceFromDelay(trk.pyr_delay, 1), 0);
    vxAssociateDelayWithNode(trk.pyr_delay, 1, trk.pyr_node, 1);
    vxAssignNode(trk.pyr_node, trk.cuda_target);
FEATURE TRACKING: GRAPH CREATION (2)

// Set up parameters for OpticalFlowPyrLK node

```c
vx_list pts_sample = vxCreateList(trk.context, VX_TYPE_KEYPOINT, 1000);
trk.pts_delay = vxCreateDelay(trk.context, (vx_reference)pts_sample, 2);
vxReleaseList(&pts_sample);

trk.curr_features = vxCreateList(trk.context, VX_TYPE_KEYPOINT, 1000);

vx_uint32 lk_epsilon = UINT_MAX;
vx_scalar s lk_epsilon = vxCreateScalar(trk.context, VX_TYPE_UINT32, &lk_epsilon);
vx_scalar s lk_num_iters = vxCreateScalar(trk.context, VX_TYPE_UINT32, &trk.lk_num_iters);
vx_bool lk_use_init_est = vx_false_e;
vx_scalar s lk_use_init_est = vxCreateScalar(trk.context, VX_TYPE_BOOL, &lk_use_init_est);
```

// Create OpticalFlowPyrLK node

```c
trk.opt_flow_node = 
vxOpticalFlowPyrLKNode(trk.graph, 
   (vx_pyramid)vxGetReferenceFromDelay(trk.pyr_delay, 0), 
   (vx_pyramid)vxGetReferenceFromDelay(trk.pyr_delay, 1), 
   (vx_list)vxGetReferenceFromDelay(trk.pts_delay, 0), 
   (vx_list)vxGetReferenceFromDelay(trk.pts_delay, 1), 
   trk.curr_features, VX_TERM_CRITERIA_ITERATIONS, s lk_epsilon, 
   s lk_num_iters, s lk_use_init_est, trk.lk_win_size);
```

```c
vxAssociateDelayWithNode(trk.pyr_delay, 0, trk.opt_flow_node, 0);
vxAssociateDelayWithNode(trk.pyr_delay, 1, trk.opt_flow_node, 1);
vxAssociateDelayWithNode(trk.pts_delay, 0, trk.opt_flow_node, 2);
vxAssociateDelayWithNode(trk.pts_delay, 0, trk.opt_flow_node, 3);
vxAssignNode(trk.opt_flow_node, trk.cuda_target);
```
// Create HarrisTrack node
trk.feature_track_node = vxnvHarrisTrackNode(trk.graph, frameGray, (vx_list)vxGetReferenceFromDelay(trk.pts_delay, 1), 0,
    trk.curr_features, trk.harris_k, trk.harris_thresh);
vxAssociateDelayWithNode(trk.pts_delay, 1, trk.feature_track_node, 1);
vxAssignNode(trk.feature_track_node, trk.cuda_target);

// Verify the graph is legal, possibly optimize
vxVerifyGraph(trk.graph);

// Clean up
vxReleaseScalar(&s_lk_epsilon);
vxReleaseScalar(&s_lk_num_iters);
vxReleaseScalar(&s_lk_use_init_est);
vxReleaseImage(&frameYUV);
vxReleaseImage(&frameGray);
}

FEATURE TRACKING: GRAPH CREATION (3)
void processFirstFrame(vx_image frameRGB, tracker_t &trk) {
    vx_image frameYUV = vxCreateImage(trk.context, trk.width, trk.height, FOURCC_IYUV);
    vx_image frameGray = vxCreateImage(trk.context, trk.width, trk.height, FOURCC_U8);
    vxuColorConvert(trk.context, frameRGB, frameYUV);
    vxuChannelExtract(trk.context, frameYUV, VX_CHANNEL_Y, frameGray);
    vxuPyramid(trk.context, frameGray, (vx_pyramid)vxGetReferenceFromDelay(trk.pyr_delay, 1), 0);
    vxnvuHarrisTrack(frameGray, (vx_list)vxGetReferenceFromDelay(trk.pts_delay, 1), 0, 0,
                     trk.harris_k, trk.harris_thres);
    vxReleaseImage(&frameYUV);
    vxReleaseImage(&frameGray);
}

vx_list getPrevFeatures(tracker_t &trk) {
    return (vx_list)vxGetReferenceFromDelay(trk.pts_delay, 0);
}
int main(int argc, const char* argv[]) {
tracker_t trk;

// Open input video source file
const std::string videoFile = "cars.avi";
cv::VideoCapture cap;
if (!cap.open(videoFile)) {
    std::cerr << "Can't open video " << videoFile << std::endl;
    return -1;
}

// Create OpenVX context
trk.context = vxCreateContext();

// Create OpenVX Image to hold frames from video source
trk.width = cap.get(CV_CAP_PROP_FRAME_WIDTH);
trk.height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
vx_image frame = vxCreateImage(trk.context, trk.width, trk.height, FOURCC_RGB);

vx_rectangle frame_rect = vxGetValidRegionImage(frame);

// Read first frame from video sequence
cv::vx::grab(cap, frame, frame_rect);

// Initialize tracking
createTrackerGraph(frame, trk);
processFirstFrame(frame, trk);

// RUN MAIN PROCESSING LOOP
for (;;) {
    // Grab next frame
    if (cv::vx::grab(cap, frame, frame_rect) != VX_SUCCESS) {
        // Reopen video file
        cap.open(videoFile);
        cv::vx::grab(cap, frame, frame_rect);
    }

    // Update frame queue
    vxAgeDelay(trk.pyr_delay);
    vxAgeDelay(trk.pts_delay);

    // Process graph
    vxSetParameterByIndex(trk.cvt_color_node, 0, (vx_reference)frame);
    vxProcessGraph(trk.graph);

    // Map OpenVX Image to OpenCV Mat, draw arrows & state
    // and show the resulting image
    vx_imagepatch_addressing_t addr;
    cv::Mat cv_frame;
    cv::vx::accessImagePatch(frame, frame_rect, 0, &addr, cv_frame);
    drawArrows(cv_frame, getPrevFeatures(trk), trk.curr_features);
    cv::imshow("Feature Tracker Demo", cv_frame);
    cv::vx::commitImagePatch(frame, 0, 0, &addr, cv_frame);
}

// Set the parameters
trk.pyr_levels = 6;
trk.harris_k = 0.04;
trk.harris_thresh = 100.0;
trk.lk_num_iters = 5;
trk.lk_win_size = 10;
}
static void drawArrow(cv::Mat& frame, cv::Point p, cv::Point q, 
            cv::Scalar line_color) {
    int line_thickness = 1;
    const double angle = std::atan2((double) p.y - q.y, (double) p.x - q.x);
    const double hypotenuse = std::sqrt((double)(p.y - q.y)*(p.y - q.y) +
             (double)(p.x - q.x)*(p.x - q.x));
    if (hypotenuse < 3.0 || hypotenuse > 50.0)
        return;
    // Here we lengthen the arrow by a factor of three.
    q.x = (int) (p.x - hypotenuse * cos(angle));
    q.y = (int) (p.y - hypotenuse * sin(angle));
    // Now we draw the main line of the arrow.
    cv::line(frame, p, q, line_color, line_thickness);
    // Now draw the tips of the arrow. I do some scaling so that the
    // tips look proportional to the main line of the arrow.
    const double tips_length = 9.0 * hypotenuse / 50.0 + 5.0;
    p.x = (int) (q.x + tips_length * cos(angle + CV_PI / 6));
    p.y = (int) (q.y + tips_length * sin(angle + CV_PI / 6));
    cv::line(frame, p, q, line_color, line_thickness);
    p.x = (int) (q.x + tips_length * cos(angle - CV_PI / 6));
    p.y = (int) (q.y + tips_length * sin(angle - CV_PI / 6));
    cv::line(frame, p, q, line_color, line_thickness);
}

static void drawArrows(cv::Mat& frame, vx_list old_points, 
            vx_list new_points, 
            cv::Scalar line_color = CV_RGB(0, 0, 255)) {
    vx_iterator old_points_iter = 0;
    vx_keypoint old_kp_ref = (vx_keypoint) vxGetListItem(old_points, &old_points_iter, VX_LIST_FRONT);
    vx_iterator new_points_iter = 0;
    vx_keypoint new_kp_ref = (vx_keypoint) vxGetListItem(new_points, &new_points_iter, VX_LIST_FRONT);

    while (old_kp_ref != 0 && new_kp_ref != 0) {
        vx_keypoint_t* old_kp = NULL;
        vx_keypoint_t* new_kp = NULL;
        vxAccessKeypoint(old_kp_ref, &old_kp);
        vxAccessKeypoint(new_kp_ref, &new_kp);

        if (new_kp->tracking_status) {
            cv::Point p(old_kp->x, old_kp->y);
            cv::Point q(new_kp->x, new_kp->y);
            drawArrow(frame, p, q, line_color);
        }
        vxCommitKeypoint(old_kp_ref, NULL);
        vxCommitKeypoint(new_kp_ref, NULL);
        old_kp_ref = (vx_keypoint) vxGetListItem(old_points, &old_points_iter, VX_LIST_NEXT);
        new_kp_ref = (vx_keypoint) vxGetListItem(new_points, &new_points_iter, VX_LIST_NEXT);
    }
}
VISIONWORKS - LOOKING FORWARD

- Enable multi-camera applications
- Depth sensor fusion
- 3D world interpretation
- Additional performance optimization
- Conformance with OpenVX once specification finalized
Thank you
FEATURE TRACKING VIDEO