Previously: Tegra in Automotive

- Infotainment / Navigation
- Digital Instrument Cluster
- Passenger Entertainment
TEGRA K1
with Kepler GPU

GPU:
- CUDA - Data-Parallel Processing with Kepler architecture (sm_32)
- 192 CORES (1 SM): > 200 GFLOPS

CPU:
- 4 x A15 ARM cores w/ NEON

Memory:
- Shared CPU/GPU:
- 12GB/S BANDWIDTH

GPU USE for COMPUTER VISION
TEGRA K1

GPU:
CUDA - Data-Parallel Processing with Kepler architecture (sm_32)
192 CORES (1 SM): > 200 GFLOPS

CPU:
- 4 x A15 ARM cores w/ NEON

Memory:
- Shared CPU/GPU: 12GB/S BANDWIDTH

Misc:
- Video Codec Engine (incl. Motion Estimation Engine)
- several other hardware units (e.g. Video preprocessing)

MAJOR PERFORMANCE IMPROVEMENTS FOR MOBILE APPLICATIONS
Hence, Tegra K1’s added goal:

Driver Assistance

and, ultimately: autonomous driving!
PATH TO AUTONOMOUS DRIVING CAR

Routine Driver Assistance
- 1 camera CV
- 4 camera visual

Piloted Drive & Autonomous Parking
- 4-6 camera CV
- 4 camera visual

Autonomous Drive
- 6-8 camera CV
- 4 camera visual

Today 2016 2020
PATH TO AUTONOMOUS DRIVING CAR

Driver Assistance

- 1 camera CV
- 4 camera visual

Piloted Drive & Autonomous Parking

- 4-6 camera CV
- 4 camera visual

Autonomous Drive

- 6-8 camera CV
- 4 camera visual
Previous Use: Driver Assistance

- Pedestrian Detection
- Blind Spot Monitoring
- Lane Departure Warning
- Collision Avoidance (*)
- Traffic Sign Recognition
- Adaptive Cruise Control

Limitation due to one camera processed: Car had little notion of 3D surroundings (exception: CollAvoidance *).

A map of 3D surroundings is only in driver’s head...
Autonomous Parking & Piloted Drive: Goals

- Car must navigate itself (albeit at low speeds)
- Needs notion of
  - **Own 3D motion and position** (beyond GPS & wheels)
  - **Obstacles** (Curbs, other cars, pedestrians,...)
  - **Free space** (for Parking)

**Computer Vision: 3D Mapping!**
“STRUCTURE FROM MOTION”

• On-the-fly 3D scene from camera @ 30 fps

• Reconstructs the “Structure” (3D point cloud) from “Motion” (moving features in video)

• Provides input for high-level processing (e.g. lane and curb detectors, parking cars)

• Note: Assumes mostly static scene outside!
STRUCTURE FROM MOTION

Feature Detection

Feature Tracking

Camera Position & Motion

Triangulation (3D Scene)
EXAMPLE ADAS APPLICATION

- Structure From Motion

Static Scene, Moving Camera

3D Point Cloud & Camera Motion (R,t)
STRUCTURE FROM MOTION

- **Input**
  - Frames from Calibrated Camera

- **Pipeline Overview**
  - Harris Corner Detection
  - Image Pyramid Creation
  - Multi-Scale Lucas-Kanade Sparse Optical Flow
  - RANSAC
  - Triangulation

- **GPU Implementation**
  - CUDA Kernels
  - Are now available as part of Vision Works!
FEATURE DETECTION

- TASK
  - Find a sparse set of points that can be tracked reliably
  - The points should cover the images somewhat evenly

- IMPLEMENTATION
  - Harris Filter to find corners in the image
  - 3x3 Filter Size, Scharr Derivates
    \[ A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix} \]
  - Image is split into 16x16 Blocks
  - At most one corner returned per Block
HARRIS CORNER

- **Result**
  - Frame Size: 1280x800
  - 646 Corners Found
  - ROI via Mask Image

- **Performance**
  - 715 us
FEATURE TRACKING / SPARSE OPTICAL FLOW

- **TASK**
  - Find the locations of corresponding corners between two frames

- **Implementation**
  - Multi-Scale Lucas-Kanade Optical Flow
  - 6x6 Window Size, Scharr Derivatives (Symmetric Window!)
  - 6 Pyramid Levels

\[
\begin{bmatrix}
V_x \\
V_y
\end{bmatrix} = \begin{bmatrix}
\sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\
\sum_i I_y(q_i)I_x(q_i) & \sum_i I_y(q_i)^2
\end{bmatrix}^{-1} \begin{bmatrix}
-\sum_i I_x(q_i)I_t(q_i) \\
-\sum_i I_y(q_i)I_t(q_i)
\end{bmatrix}
\]
MULTI-SCALE

ORIGINAL

1/2  1/4

LK  Upscale & LK  Upscale & LK
MULTI-SCALE LUCAS-KANADE FLOW

- **Result**
  - 609 Tracks

- **Pyramid**
  - 445 us
  - 5+1 Levels

- **LK**
  - 1080 us
  - 6x10 Iterations
OUTLIER FILTERING

- **TASK**
  - Filter Feature Tracks that are not in correspondence with a Camera moving in a static 3D World

- **IMPLEMENTATION**
  - RANSAC: Random Sample Consensus
  - 7 Tracks are sufficient to compute the Camera Motion
  - Generate many Hypothesis from random samples
  - Inlier if Angular Reprojection Error is below Threshold (Oliensis, PAMI, 2002)
  - Sample with highest # of Inlier is Winner
RANSAC + CAMERA MOTION?

- **Result**
  - 545 Tracks
  - Rotation Matrix
  - Translation Vector

- **Performance**
  - 54 Iterations (max 30% Outlier)
  - 975 us
CAMERA CALIBRATION

- **TASK**
  - Find the relation between Pixel location and Optical ray

- **IMPLEMENTATION**
  - Lens with large viewing angle best for ADAS
  - Need Camera Model that can handle those
  - We use the model proposed by Scaramuzza and his MATLAB toolbox to calibrate our Camera (https://sites.google.com/site/scarabotix/ocamcalib-toolbox)
TRIANGULATION

- TASK
  - Find 3D Points for Tracks given Camera Calibration and Motion

- IMPLEMENTATION
  - Compute Rays for Feature Locations from Camera Calibration
  - Least-Squares Solution (Rays will not intersect in practice)

\[
\hat{\beta} = (X^T X)^{-1} X^T y.
\]
3D POINT CLOUD

- **Result**
  - 446 3D Points

- **Performance**
  - 70 us
## TIMING SUMMARY

<table>
<thead>
<tr>
<th>Function</th>
<th>Total GPU Time in Microseconds (10^-6 sec)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harris Corner</td>
<td>715 us</td>
<td>Feature Detection</td>
</tr>
<tr>
<td>Down Sampling</td>
<td>445 us</td>
<td>Pyramid Creation</td>
</tr>
<tr>
<td>Lucas-Kanade Optical Flow</td>
<td>1080 us</td>
<td>Feature Tracking</td>
</tr>
<tr>
<td>RANSAC</td>
<td>975 us</td>
<td>Outlier Filtering + Relative Camera Motion</td>
</tr>
<tr>
<td>Triangulation</td>
<td>70 us</td>
<td>3D Point Cloud</td>
</tr>
<tr>
<td>TOTAL</td>
<td>3285 us (3.3 ms)</td>
<td>SfM Time per Frame and Camera</td>
</tr>
</tbody>
</table>
THANKS TO AUDI FOR PROVIDING THE DATASET