ELTECar: Real-Time Sensor Data Recording and Processing for an Autonomous Vehicle

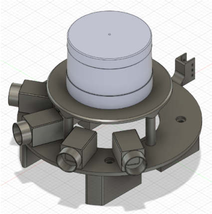
Eötvös Loránd University, Faculty of Informatics
Department of Algorithm and Applications
Geometric Computer Vision Group

GPU Day 2024.

Motivation

- Accumulating data for years
- **Goals**
  - Making data public for research work
  - Use for real-time methods
- **Our Solution:** System with Client-Server architecture
  - Server: Place data into shared memory at specified time intervals
  - Our Client: Visualising and processing the incoming data (but can be any other application for other purposes)

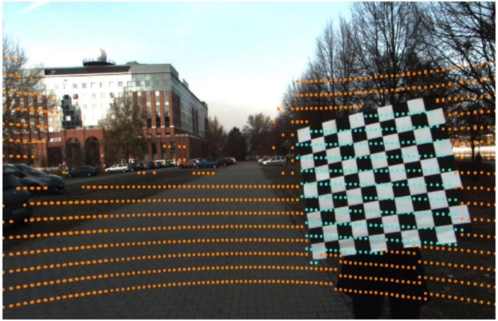
About the Dataset



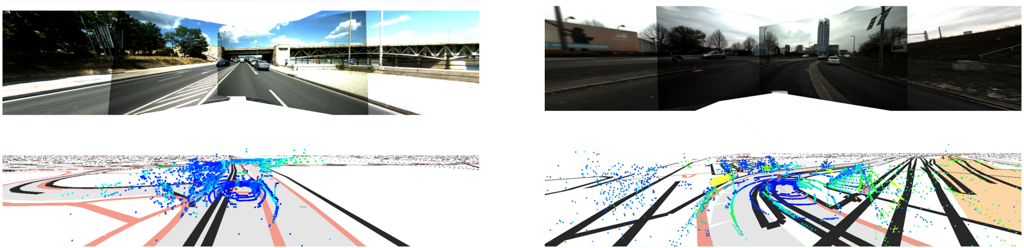
- Sensors
  - Pin-hole cameras
  - LiDAR
  - RTK-GPS
- **Hardware synchronised in time** - 4 SPS
- ~800 minutes of footage recorded in Budapest with this sensor configuration

Sensor calibration

- Intrinsic parameters
  - ∀ cameras
- Extrinsic parameters
  - LiDAR-Camera → ∀ camera
  - Camera-Camera → ∀ neighboring pairs
- **Method:** Chessboard-based calibration



DATA VISUALIZATION - OVERVIEW



- OpenGL + SDL2
  - Easier portability between Windows and Linux
  - Replace the shared memory processing class
- Features
  - Cameras: panoramic video
  - LiDAR: 3D visualisation of point cloud + coloring schemes
  - GPS: draw environment around the vehicle

Panoramic video

- Stitching images for better clarity
- **Method:** plane-plane homography
  - $!\exists$ projective transformation between image pairs if the translation is $\mathbf{0}$
  - Calculated between every neighboring image pairs

Existence of homography

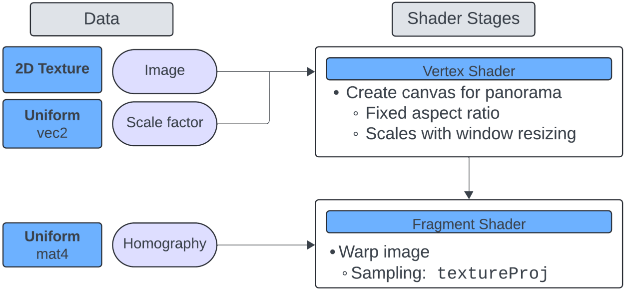
- In our case, translation between the focal point of cameras is negligible $\Longrightarrow$ assume only orientation is different
- The origin is selected as the focal point of one camera
  - Origin is fixed to the focal point of cam_1 in case of the $cam_1 \rightarrow cam_2$ homography
- Camera projection matrices: $\mathbf{P_1} = \mathbf{K_1}\left[\mathbf{R_1}|\mathbf{0}\right]$ and $\mathbf{P_2} = \mathbf{K_2}\left[\mathbf{R_2}|\mathbf{0}\right]$
- Transformation between two corresponding points in different images

$$\mathbf{u_2} = \mathbf{K_2 R_2 R_1^T K_1^{-1} u_1}$$

  - $\mathbf{u_{1,2}}$ are 2D points represented in homogeneous coordinates
  - $!\exists \mathbf{H} = \mathbf{K_2 R_2 R_1^T K_1^{-1}}$

Pipeline

- **Keypoint detection:** SURF detector
- **Keypoint matching:** k-NN, FLANN-based matcher
- **Homography estimation**
  - Initial algebraic estimation from 4 point corrspondences
  - Non-linear numeric optimization (Levenbenberg-Marquardt)
  - Noise sensitiveness → robustified with RANSAC



Rotation interpolation 1

- Rotation around the common axis of cameras
- Minimize post-projection distortion
- **Method:** extracting rotational transformations from homography
  - Angle between the two center cameras
  - Break down the transformation : axis + *angle*

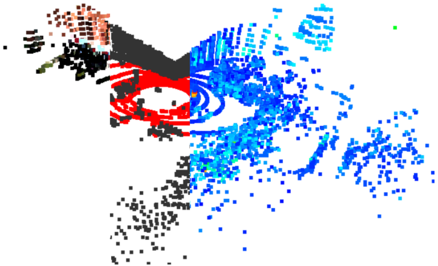
Rotation interpolation 2

- Homography: $\mathbf{H} = \mathbf{K}_j \mathbf{R}_{ij} \mathbf{K}_i^{-1}$
- Point registration problem [1]: $\mathbf{H}\mathbf{K}_i = \mathbf{R}_{ji} \mathbf{K}_j^T$
- $\mathbf{R}_{ij}$ orthogonal $\Longrightarrow \mathbf{R}_{ij}^T = \mathbf{R}_{ji}$

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. Least squares fitting of two 3-D point sets. PAMI, 9(5):698–700, 1987.

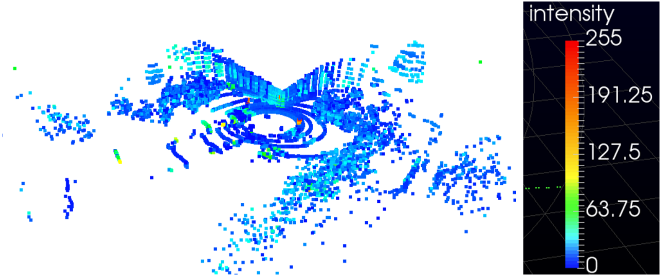
Point cloud coloring

- Points are particles → one draw call/frame
- Multiple color schemes, to help interpretability
  - Based on reflection intensity
  - Based on environment
  - Points of the ground plane

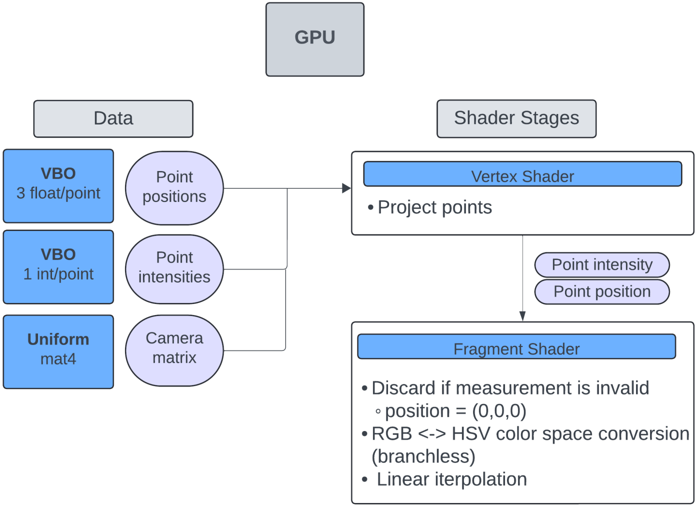


1 Reflection intensity

- Intensity of reflected light
- Traffic lights and signs easily detectable
- Coming from the sensor: integer between 0 – 255
- Coloring scheme of VeloView
  - **Why:** Official application of Velodyne LiDARs
  - 3 colors: blue → yellow → red
  - Linear interpolation in HSV color space on the GPU

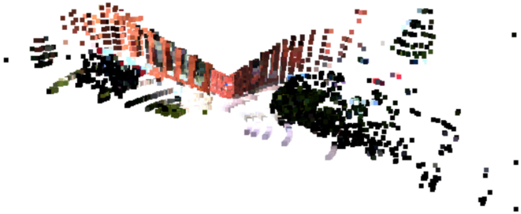


Pipeline

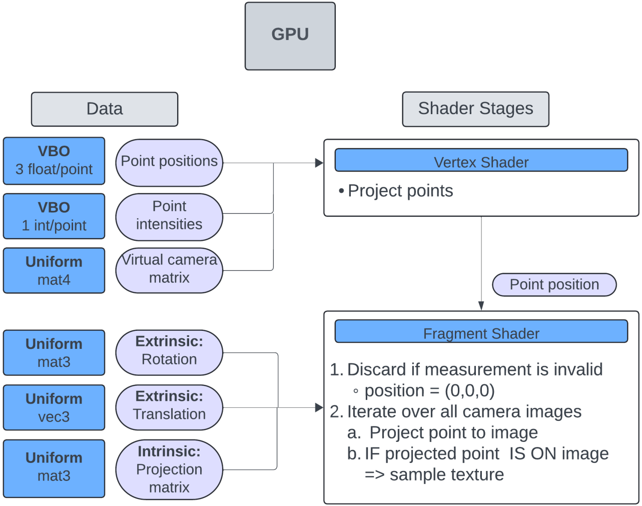

GPU
Data
VBO
3 float/point
Point positions
VBO
1 int/point
Point intensities
Uniform
mat4
Camera matrix
Shader Stages
Vertex Shader
• Project points
Point intensity
Point position
Fragment Shader
• Discard if measurement is invalid
◦ position = (0,0,0)
• RGB <-> HSV color space conversion (branchless)
• Linear iterpolation

2 Color of the environment

- Real color of points
- Color information from images
- **Method:** projecting cloud points to images with calibration data
  - LiDAR-Camera extrinsic parameters
  - Intrinsic parameters of cameras

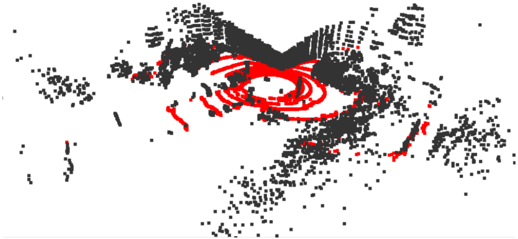


Pipeline


GPU
Data
Shader Stages
VBO
3 float/point
Point positions
VBO
1 int/point
Point intensities
Uniform
mat4
Virtual camera matrix
Uniform
mat3
Extrinsic:
Rotation
Uniform
vec3
Extrinsic:
Translation
Uniform
mat3
Intrinsic:
Projection matrix
Vertex Shader
• Project points
Point position
Fragment Shader
1. Discard if measurement is invalid
◦ position = (0,0,0)
2. Iterate over all camera images
a. Project point to image
b. IF projected point IS ON image
=> sample texture

3 Ground plane detection

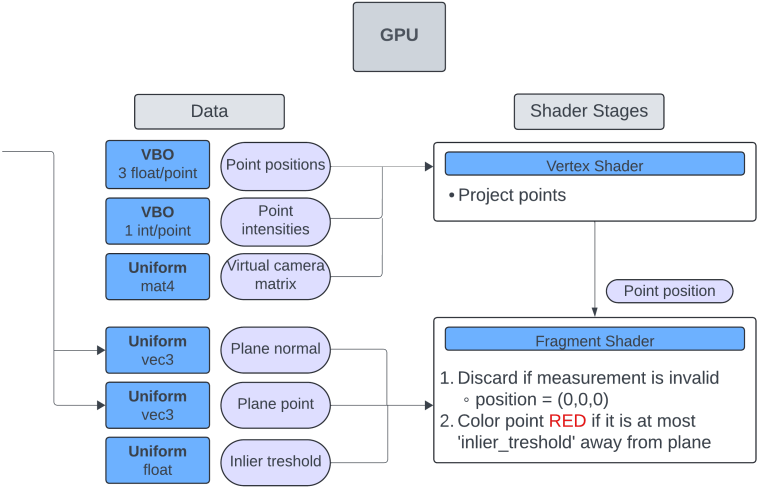
- Color the points of the ground plane
- **Plane fitting:** Principal Component Analysis (PCA)
  - Minimizing geometrical error
  - Optimal in the least squares sense
- Sensitive to noise $\rightarrow$ robustified with RANSAC



PIPELINE

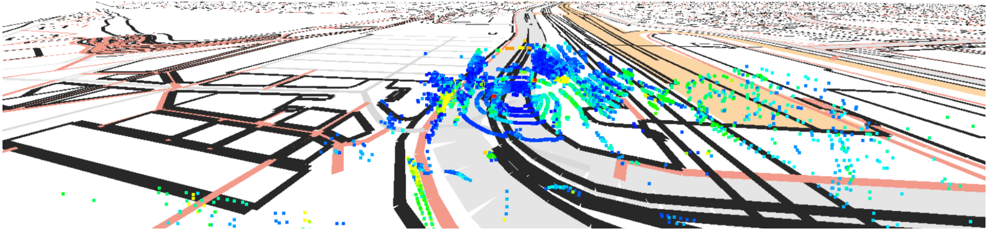
CPU

- **Plane fitting**
  - linear overdetermined homogeneous system
  - *RANSAC*: select model with maximal inliers



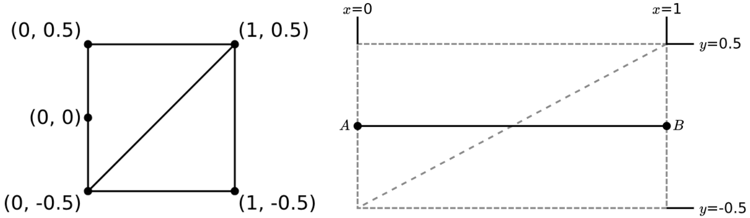
Map data

- Display roads on the ground plane
- Map data from OpenStreetMap [1]
- Map fitting
  - Coordinate system is fixed to the car $\Longrightarrow$ rotation is needed
  - Heading: from 2 sequential GPS measurements *currently*



[1] https://www.openstreetmap.org

LINE DRAWING

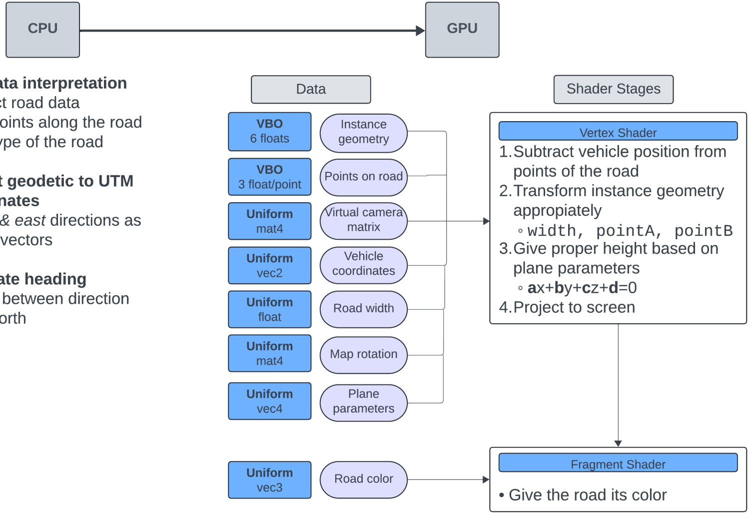


- Draw roads with instancing [1] $\Longrightarrow$ **lot fewer draw calls**
  - Line segments are instances
  - One road/draw call
- Color and width depend on road type
  - *highway, footway, ...*
- **Done in shader**

[1] https://wwwtyro.net/2019/11/18/instanced-lines.html

Pipeline

- **Map data interpretation**
  - extract road data
    => points along the road
    => type of the road
- **Project geodetic to UTM coordinates**
  - *north & east* directions as basis vectors
- **Calculate heading**
  - angle between direction and north



Future plans

- Move the project to GitLab
- Filter position and rotation
  - Kalman filter
  - Sensor fusion
  - Ackermann steering model
- Fish eye optics
- Object detection
  - special cases: parking space detection
- Record new weather conditions
  - We currently have snow and
  - **rainbow** recorded!

Conclusion

- Make the dataset + real-time system public
  - Sensor data
  - Corresponding calibration data
  - **Now available: https://cv.inf.elte.hu**
- Real-time data visualization system accelerated by the GPU
  - Panoramic video
  - Colored 3D point cloud
  - Drawing roads around the vehicle

Thank you for your attention!

http://cv.inf.elte.hu

geometriccomputervisiongro6255@Youtube