Persival

ASAM

ASAM OSI® (Open Simulation Interface)

FlatBuffers for OSI 4.0

Clemens Linnhoff & Philipp Rosenberger

Who we are

M. Sc.

Philipp Rosenberger

M. Sc.

Clemens Linnhoff

TECHNISCHE UNIVERSITÄT DARMSTADT

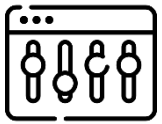
FZD

Persival

Persival in a Nutshell

Requirement Analysis
Real Sensors, Sensor Models, Environment Simulation
PerCollECT, CEPRA, Inspection of parameter space

Sensor Model Customization
High-fidelity customization of models to real sensor systems
Basis: FZD Open-Source Models



Perception Sensor Simulation & Validation

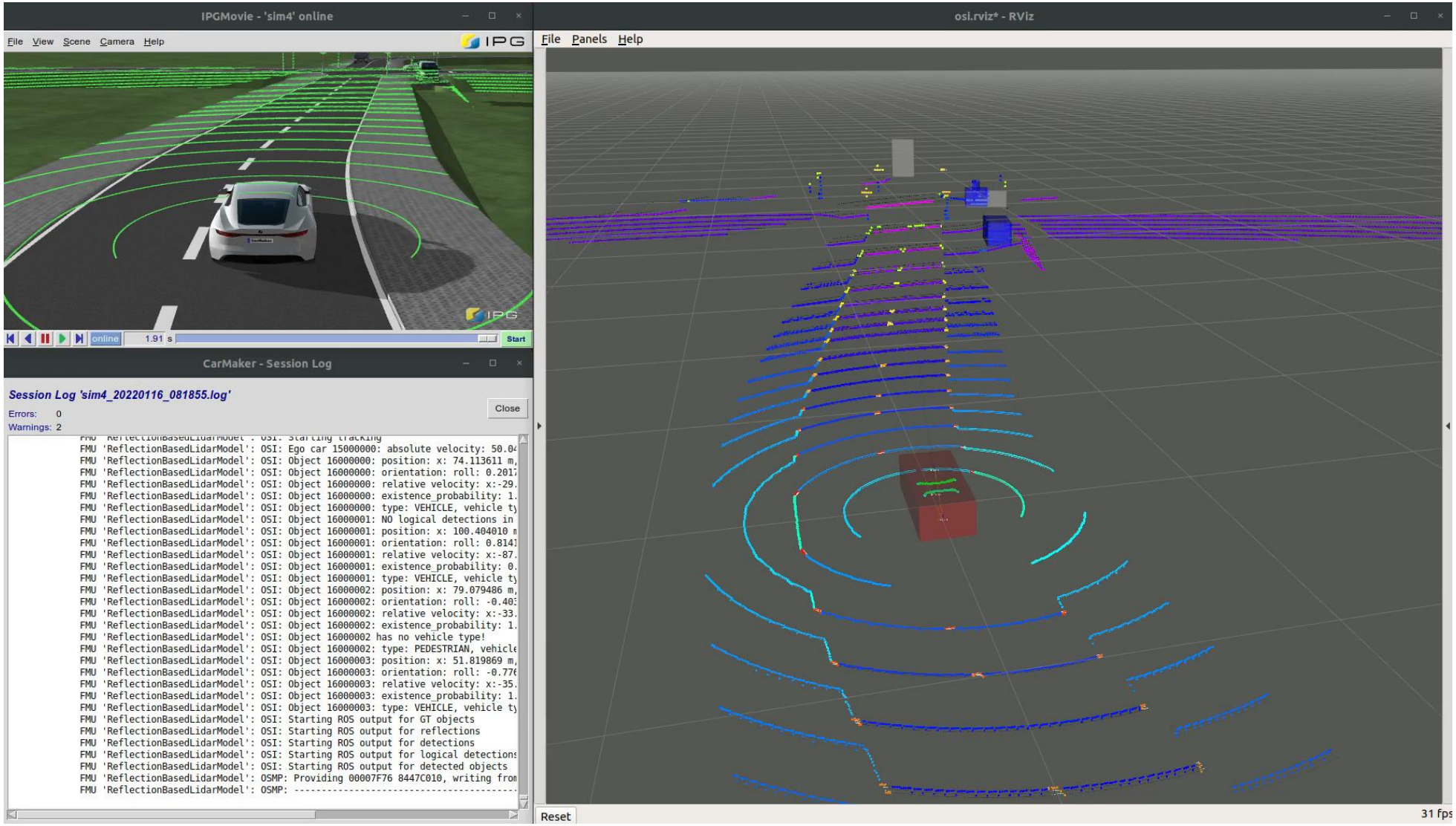
Perception Algorithms
Simulation based development of perception
Ground truth from simulation instead of costly labeling

Sensor Model Validation
Confidence and uncertainty quantification of models
Sample experiments with reference sensors on test track

Standardization of Interfaces

Icon sources: https://www.flaticon.com/authors/freepik

Open-Source Reflection-Based Lidar Object Model



YouTube

Reflections from tool

Detections from model

Ego car

GT objects

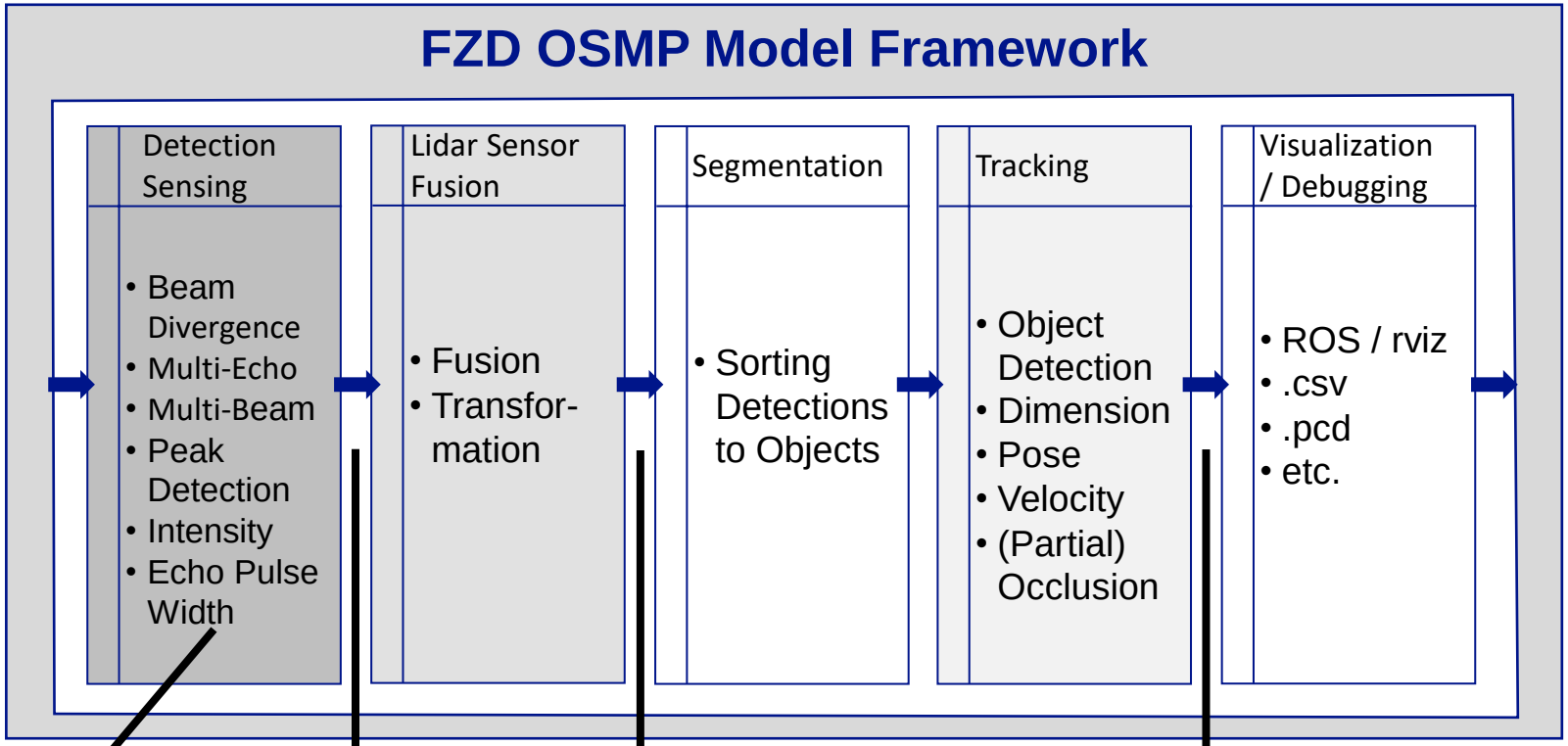
Detected objects

https://gitlab.com/tuda-fzd/perception-sensor-modeling/reflection-based-lidar-object-model

Persival

Open-Source Reflection-Based Lidar Object Model

https://gitlab.com/tuda-fzd/perception-sensor-modeling/modular-osmp-framework

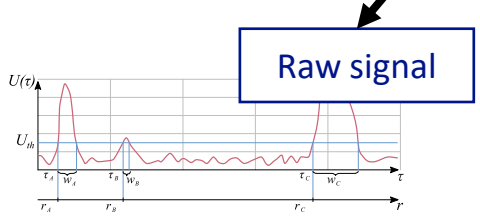


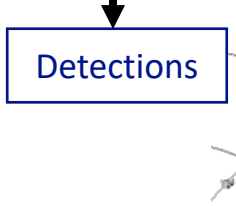
osi3::SensorView

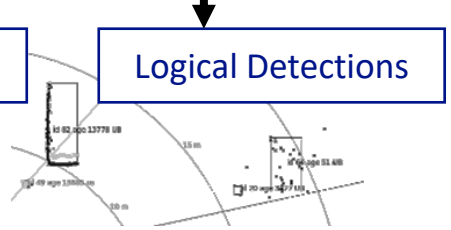
E.g.:
- GroundTruth:: MovingObject / StationaryObject
- LidarSensorView
- RadarSensorView

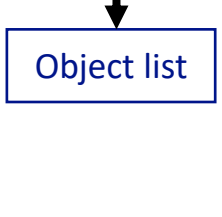
osi3::SensorData

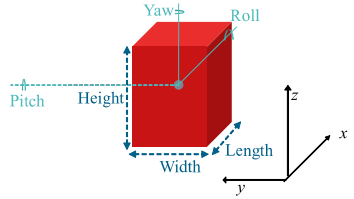
E.g.:
- MovingObject
- StationaryObject
- LidarDetection
- RadarDetection
- LogicalDetectionData







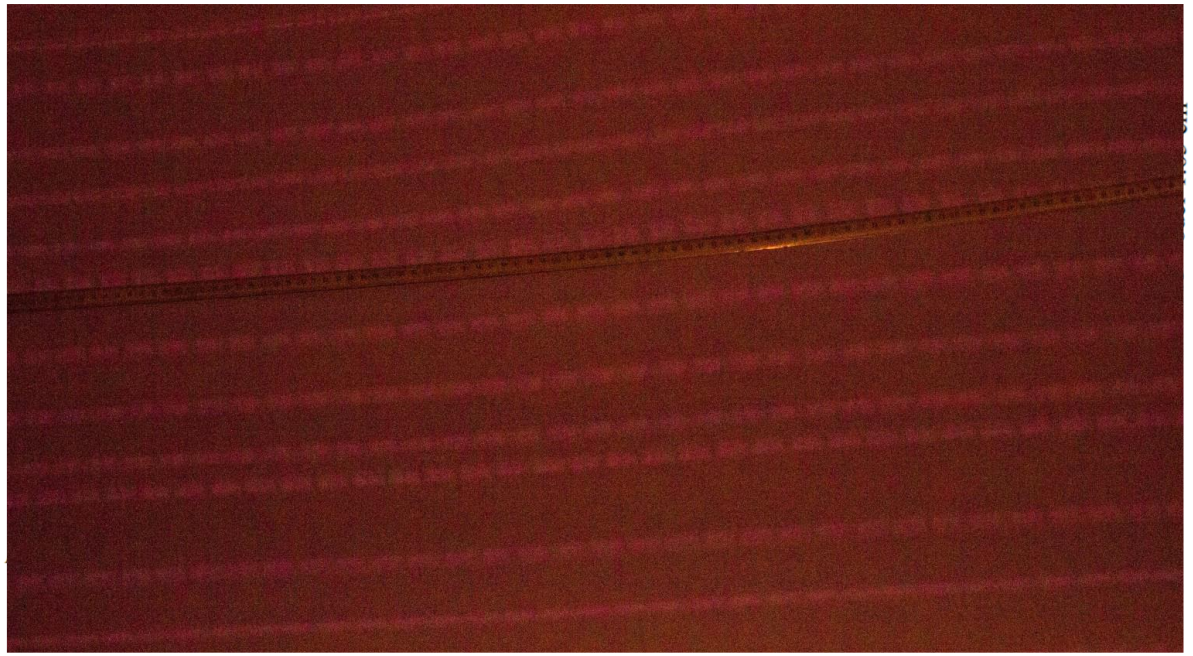




Solution Approaches for better Performance

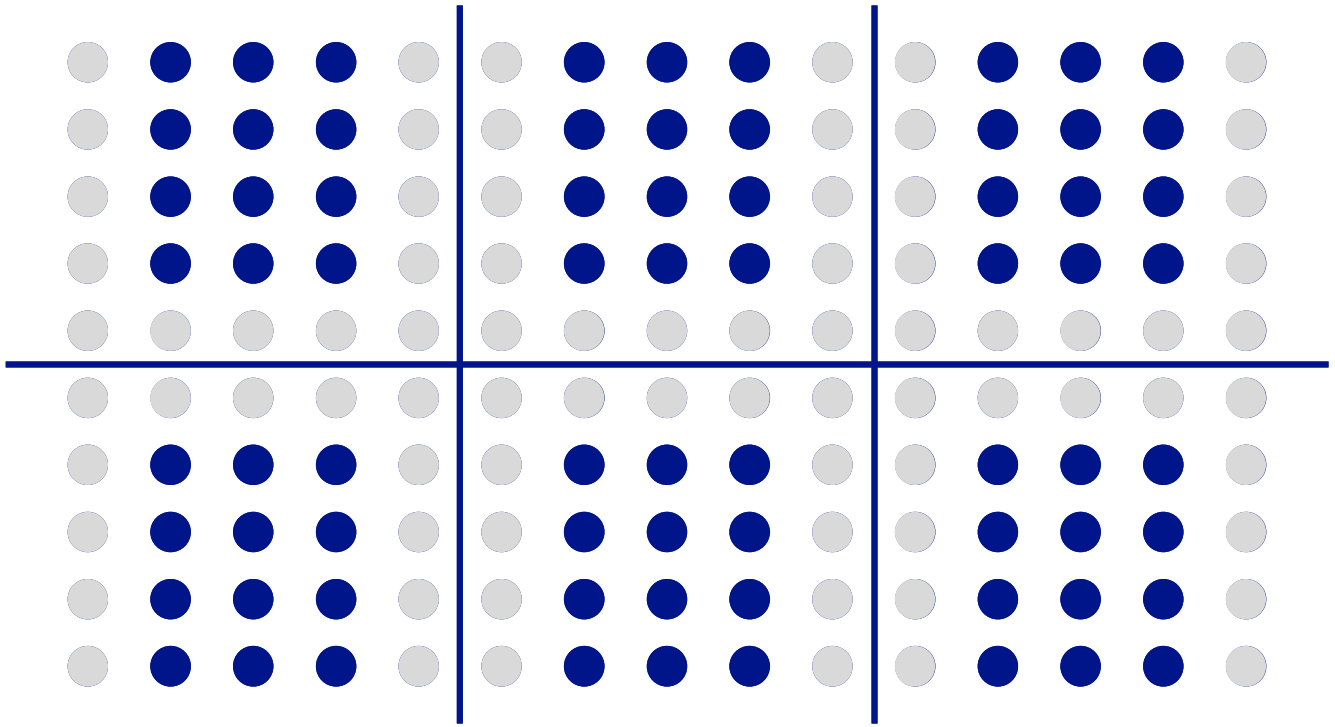
New LidarSensorViewConfiguration (No rays in between actual beams)

J.F. Tamm-Morschel: *Erweiterung eines phänomenologischen Lidar-Sensormodells durch identifizierte physikalische Effekte.* MaTh, Technische Universität Darmstadt, DOI: 10.25534/tuprints-00011499, p. 46



Solution Approaches for better Performance

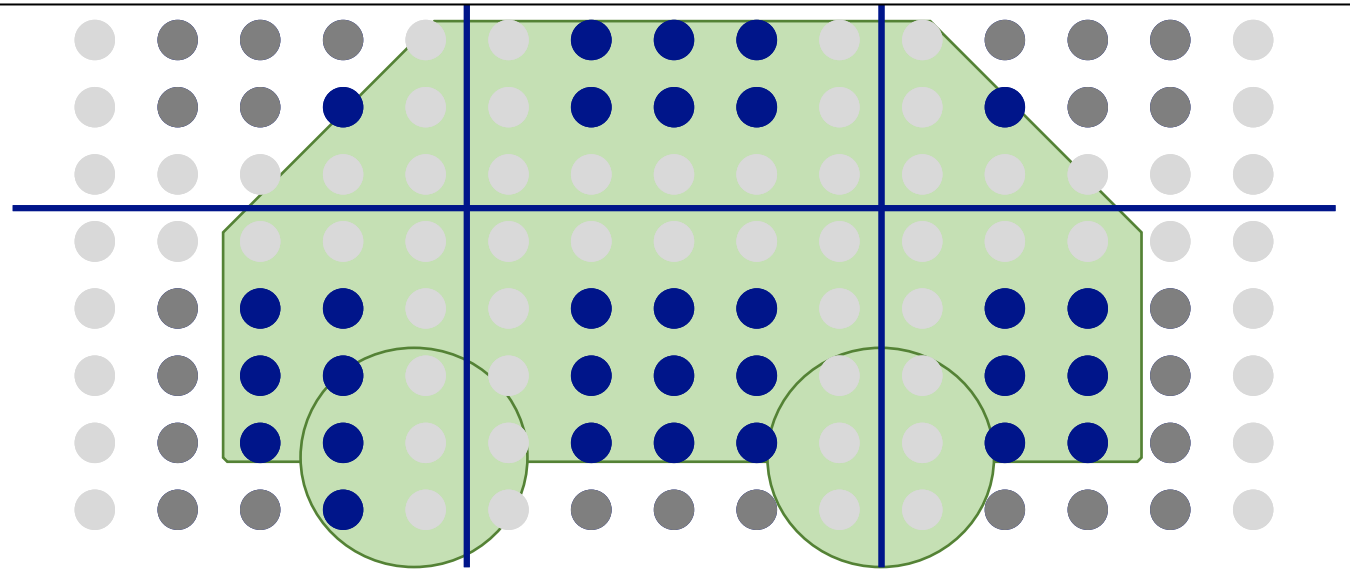
New LidarSensorViewConfiguration (No rays in between actual beams)



Solution Approaches for better Performance

New LidarSensorViewConfiguration (No rays in between actual beams)

New LidarSensorView (Only serialize & transfer hit points)



Solution Approaches for better Performance

New LidarSensorViewConfiguration (No rays in between actual beams)

New LidarSensorView (Only serialize & transfer hit points)

SET Level Project (ongoing)

Less rays (?)

More efficient modeling

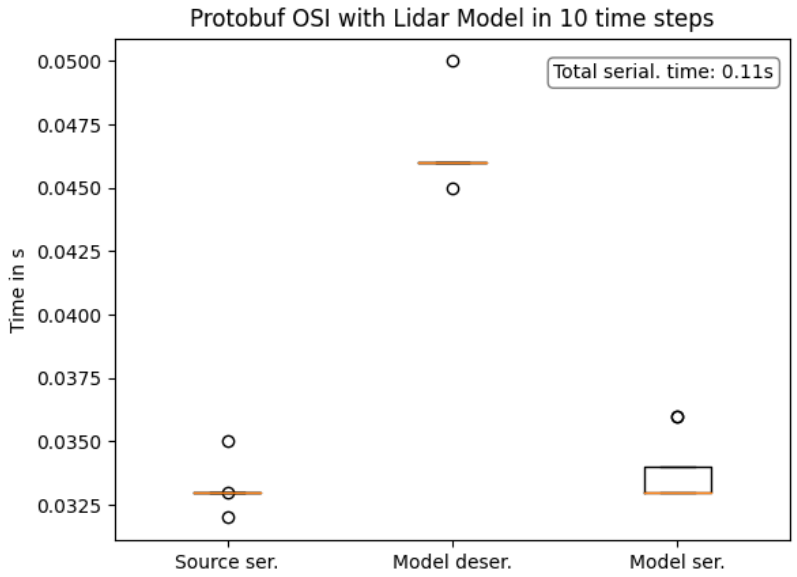
Performance with Protobuf – Lidar Example

Current OSI

- VLP32: 360° azimuth, 40° elevation
- Beam width: 0.18°, beam height: 0.09°, ray discretization: 0.03°
- Because of unsymmetrical FoV, 50° elevation generated
- -> 20.000.000 LidarReflections

- **SETLevel improved OSI**
- VLP32: 360° azimuth, 40° elevation
- 900 azimuth beams, 32 Layers
- Beam width: 0.18°, beam height: 0.09°, ray discretization: 0.03°
- Assume 80 % of rays generate hitpoint
- -> ≈400.000 LidarReflections

Possible Solution:
Use FlatBuffers instead of Protobuf



FlatBuffers in Theory

FlatBuffers is an efficient cross platform serialization library for C++, C#, C, Go, Java, JavaScript, PHP, Python, …

Originally created at Google for game development and other performance-critical applications

Open-Source on GitHub under the Apache license, v2

Source: https://google.github.io/flatbuffers/

FlatBuffers in Theory

- **Access to serialized data without parsing/unpacking**
  - Hierarchical data in a flat binary buffer
  - Can still be accessed directly without parsing/unpacking
  - While also still supporting data structure evolution (forwards/backwards compatibility).
- **Memory efficiency and speed**
  - The only memory needed to access your data is that of the buffer
  - Requires 0 additional allocations (in C++, other languages may vary)
  - Access is close to the speed of raw struct access
- **Flexible**
  - Optional fields for forwards and backwards compatibility
  - A lot of choice in what data to write and what not, and how to design data structures.
- **Tiny code footprint**
  - Small amounts of generated code
  - Just a single small header as the minimum dependency
- **Strongly typed**
  - Errors happen at compile time, so no manually writing repetitive and error prone run-time checks.

Source: https://google.github.io/flatbuffers/

FlatBuffers in Theory

Crash-Course

- **Schema file** that allows for data structures to serialize
  - Scalar (ints/floats of all sizes), string; array of any type; references; unions
  - Fields are optional and have defaults
- **`flatc`** (the FlatBuffer compiler) to generate a C++ header (or C#/Python.. classes)
  - With helper classes to access and construct serialized data
  - This header (say `mydata_generated.h`) only depends on `flatbuffers.h`
  - Use the `FlatBufferBuilder` class to construct a flat binary buffer
  - The generated functions allow you to add objects to this buffer recursively, often as simply as making a single function call
- **Store or send** your buffer somewhere!
- **Read the buffer**
  - Pointer to the root object from the binary buffer
  - From there traverse it with `object->field()`

Source: https://google.github.io/flatbuffers/

FlatBuffers vs. Protocol Buffers in Theory

- Advantages of FlatBuffers compared to protocol buffers

  - FlatBuffers do not need a parsing / unpacking step to a secondary representation before data access, which often needs per-object memory allocation

  - Protocol Buffers code is an order of magnitude bigger

Source: https://google.github.io/flatbuffers/

FlatBuffers vs. Protobuf in Application

Protobuf – Proto File

```
message SensorView
{
    optional InterfaceVersion version = 1;
    optional Timestamp timestamp = 2;
    optional Identifier sensor_id = 3;
    optional MountingPosition mounting_position = 4;
    optional MountingPosition mounting_position_rmse = 5;
    optional HostVehicleData host_vehicle_data = 6;
    optional GroundTruth global_ground_truth = 7;
    optional Identifier host_vehicle_id = 8;
    repeated GenericSensorView generic_sensor_view = 1000;
    repeated RadarSensorView radar_sensor_view = 1001;
    repeated LidarSensorView lidar_sensor_view = 1002;
    repeated CameraSensorView camera_sensor_view = 1003;
    repeated UltrasonicSensorView ultrasonic_sensor_view = 1004;
}
message GenericSensorView
{
   optional GenericSensorViewConfiguration view_configuration = 1;
}
```

...

FlatBuffers – FBS File

```
table SensorView
{
        version:osi3.InterfaceVersion;
        timestamp:osi3.Timestamp;
        sensor_id:osi3.Identifier;
        mounting_position:osi3.MountingPosition;
        mounting_position_rmse:osi3.MountingPosition;
        host_vehicle_data:osi3.HostVehicleData;
        global_ground_truth:osi3.GroundTruth;
        host_vehicle_id:osi3.Identifier;
        generic_sensor_view:[osi3.GenericSensorView];
        radar_sensor_view:[osi3.RadarSensorView];
        lidar_sensor_view:[osi3.LidarSensorView];
        camera_sensor_view:[osi3.CameraSensorView];
        ultrasonic_sensor_view:[osi3.UltrasonicSensorView];
}
table GenericSensorView
{
  view_configuration:osi3.GenericSensorViewConfiguration;
}
```

...

FlatBuffers vs. Protobuf in Application

Protobuf – Generate SensorView

```
osi3::SensorView sensor_view;
sensor_view.mutable_sensor_id()->set_value(1);
osi3::GroundTruth *currentGT = sensor_view.mutable_global_ground_truth();

osi3::MovingObject *veh = currentGT->add_moving_object();
veh->mutable_id()->set_value(10);
veh->mutable_base()->mutable_position()->set_x(50);
veh->mutable_base()->mutable_position()->set_y(2);

sensor_view.SerializeToString(currentBuffer);
encode_pointer_to_integer(currentBuffer->data(),integer_vars[BASEHI_IDX],...);
```

FlatBuffers – Generate SensorView

```
flatbuffers::FlatBufferBuilder builder(1024);
std::vector<flatbuffers::Offset<osi3::MovingObject>> moving_object_vector;
flatbuffers::Offset<osi3::Vector3d> position = osi3::CreateVector3d(builder, 50, 2, 0.0);
osi3::BaseMovingBuilder base_moving_builder(builder);
base_moving_builder.add_position(position);
flatbuffers::Offset<osi3::BaseMoving> base_moving = base_moving_builder.Finish();
osi3::MovingObjectBuilder moving_object_builder(builder);
moving_object_builder.add_base(base_moving);
moving_object_vector.push_back(moving_object_builder.Finish());
auto moving_object_flatvector = builder.CreateVector(moving_object_vector);
osi3::GroundTruthBuilder ground_truth_builder(builder);
ground_truth_builder.add_moving_object(moving_object_flatvector);
auto ground_truth = ground_truth_builder.Finish();
auto sensor_id = osi3::CreateIdentifier(builder, 1);
osi3::SensorViewBuilder sensor_view_builder(builder);
sensor_view_builder.add_sensor_id(sensor_id);
sensor_view_builder.add_global_ground_truth(ground_truth);
auto sensor_view = sensor_view_builder.Finish();
builder.Finish(sensor_view);
auto currentBuffer = builder.GetBufferPointer();
encode_pointer_to_integer(currentBuffer.data(),integer_vars[BASEHI_IDX],...);
```

FlatBuffers vs. Protobuf in Application

Protobuf – Read SensorView

```
int num_moving_obj = sensor_view_in.global_ground_truth().moving_object_size();

bool has_moving_obj = !sensor_view_in.global_ground_truth().moving_object().empty();
bool has_base = sensor_view_in.global_ground_truth().moving_object(0).has_base();
```

Mutable fields can be changed.

FlatBuffers – Read SensorView

```
size_t num_moving_obj = sensor_view_in->global_ground_truth()->moving_object()->size();

bool has_moving_obj = sensor_view_in->global_ground_truth()->moving_object();
bool has_base = sensor_view_in.global_ground_truth().moving_object()->Get(0)->base();
```

Fields are constant. (Mutable option exists, but slower and not recommended)

FlatBuffers vs. Protobuf in Application

- Tables vs. Structs
  - Every Protobuf message is automatically transformed to a Flatbuffers table
  - Messages with only primitive fields can also be transformed to Flatbuffers structs
  - You do not need a builder to create a struct
  - Might be more efficient in some cases
- Considerations
  - FBS files different from Proto (backward compatibility?)
  - Not possible to offer both serialization options in parallel
  - => Strategic decision for the future of OSI & OSMP
  - => The Community is required to try both options to make a decision!

Experiences with FlatBuffers in the Audiance?

Service Provider Tasks

1. Port of OSMPDummySource model to OSI FlatBuffers
2. Port of OSMPDummySensor model to OSI FlatBuffers

- Presentation and discussion of results during ASAM OSI CCB meetings
- Working example as pull request, where possible
- Report about the implementation experience

Service Provider Tasks

1. Port of OSMPDummySource model to OSI FlatBuffers
2. Port of OSMPDummySensor model to OSI FlatBuffers
3. Port of Open-Source reflection-based sensor model to OSI FlatBuffers
4. Creation of a suitable SensorView input source for the reflection-based model for performance analysis

- Presentation and discussion of results during ASAM OSI CCB meetings
- Working example as pull request, where possible
- Report about the implementation experience
- Generation of a test suite with different amounts of objects and lidar reflections for testing

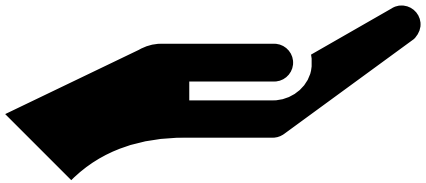
Service Provider Tasks

1. Port of OSMPDummySource model to OSI FlatBuffers
2. Port of OSMPDummySensor model to OSI FlatBuffers
3. Port of Open-Source reflection-based sensor model to OSI FlatBuffers
4. Creation of a suitable SensorView input source for the reflection-based model for performance analysis
5. Co-Simulation platform runner scripts and test evaluation

- Presentation and discussion of results during ASAM OSI CCB meetings
- Working example as pull request, where possible
- Report about the implementation experience
- Generation of a test suite with different amounts of objects and lidar reflections for testing
- Detailed performance evaluation report

Open-Source Through and Through

All results will be Open-Source to enable peer-review & execution!



- New Versions with FlatBuffers as branches in original repositories
- Eclipse OpenMCx as Co-Simulation platform (utilizing SSP)
- Timing export in .json format (as defined in SET Level)
- Python scripts for evaluation
- All performance and coding experience reports publicly available at the end

Project Time Line for Public Releases

- 28.01.2022: Ported OSMPDummySource FlatBuffers model
- 28.01.2022: Ported OSMPDummySensor FlatBuffers model
- 04.03.2022: Suitable SensorView source for reflection-based sensor model
- 04.03.2022: Ported Open-Source reflection-based sensor FlatBuffers model
- Co-Simulation platform runner scripts and test evaluation
  - 18.02.2022: V1 for DummySource and DummySensor
  - 31.03.2022: Final version
- 31.03.2022: Final performance and coding experience reports

Questions? Interested Partners?

Persival

Perception Sensor Simulation & Validation

www.persival.de

Spin-off from

TECHNISCHE UNIVERSITÄT DARMSTADT

FZD