Audi
Vorsprung durch Technik

Technische Universität München

Advanced Driver Assistance System Testing using OptiX

NVIDIA GTC 2012, San Jose

Advanced Driver Assistance System Testing using OptiX

Erwin Roth

Robotics and Embedded Systems, Technischen Universität München

Tugkan Calapoglu, Holger Helmich

Vires Simulationstechnologie GmbH, Bad Aibling

Kilian v. Neumann-Cosel

Automotive Safety Technologies GmbH, Ingolstadt

Marc-Oliver Fischer

Department for Vehicle Safety Sensor Algorithms, AUDI AG, Ingolstadt

Andreas Kern

Audi Electronics Venture GmbH, Ingolstadt

Andreas Heider, Alois Knoll

Robotics and Embedded Systems, Technische Universität München

Agenda

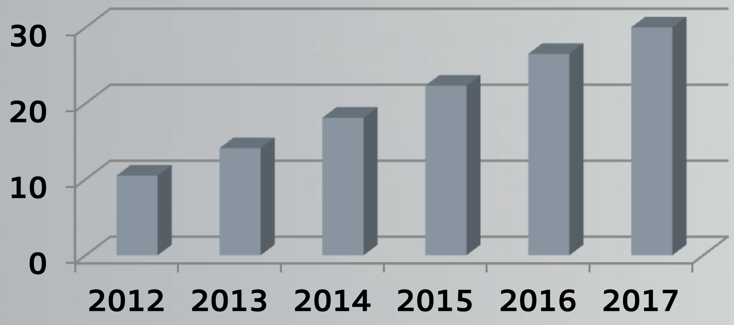
- Motivation, requirements and goals
- Integration of OptiX into the Vires Virtual Test Drive simulation software
- Advanced material and emitter data descriptions
- Example sensor model implementations
- Model validation and verification process
- Summary and Outlook

Motivation

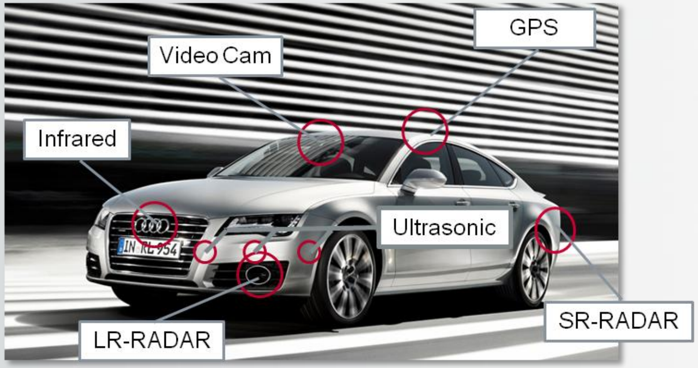
Growing challenges for testing new Advanced Driver Assistance Systems (ADAS)

- Increased number of comfort-, energy-management- and safety-related functions
- Growing dependency of ADAS-functions on multiple perception sensors
- Difficulties to record reproducible sensor data for real world scenarios

Forecast: 300% increase in shipped ADAS units [Mio.]

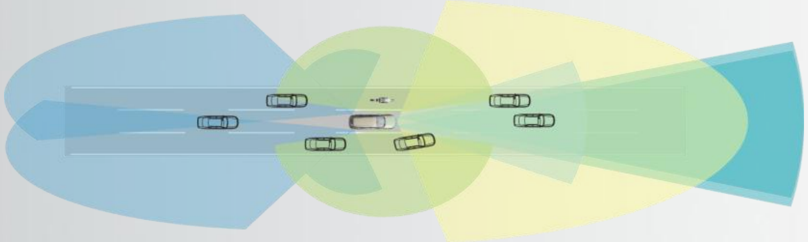


Source: iSuppli



Main Objectives

- Support ADAS testing with computer simulations for **realistic multi-sensor data computation**
- Validated sensor models as parts of an integrated vehicle and environment simulation system



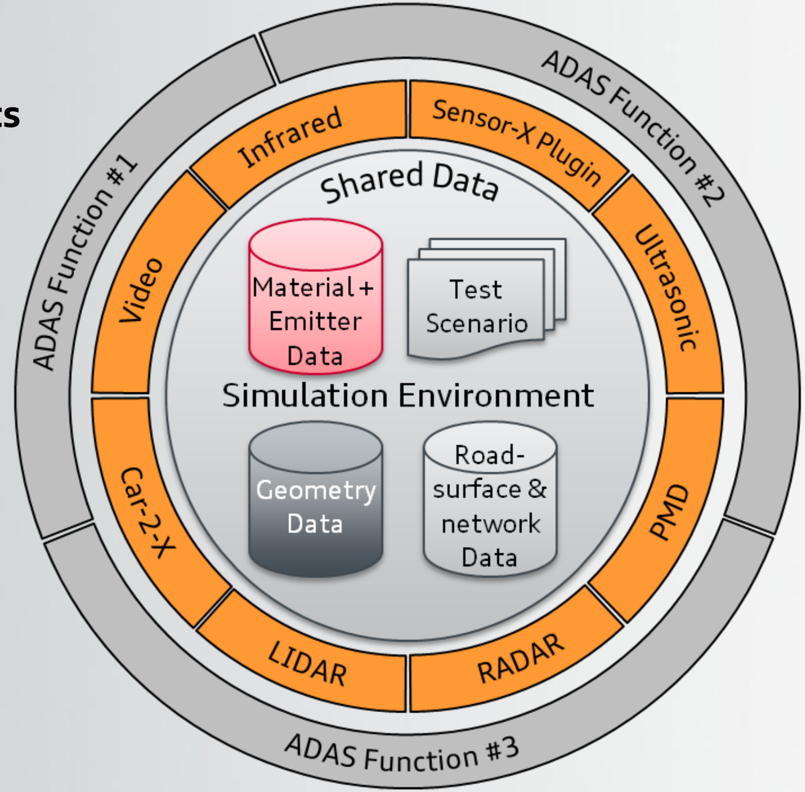
- Enable closed-loop simulations in Hardware- and Software-in-the-loop testbeds
- Reproducibility of test scenarios for a wide range of environment and traffic conditions

> - Early evaluation of new sensor concepts and ADAS functions
> - Increased test space coverage by combining real and virtual test drives

Multi-Sensor Simulation Environment Objectives

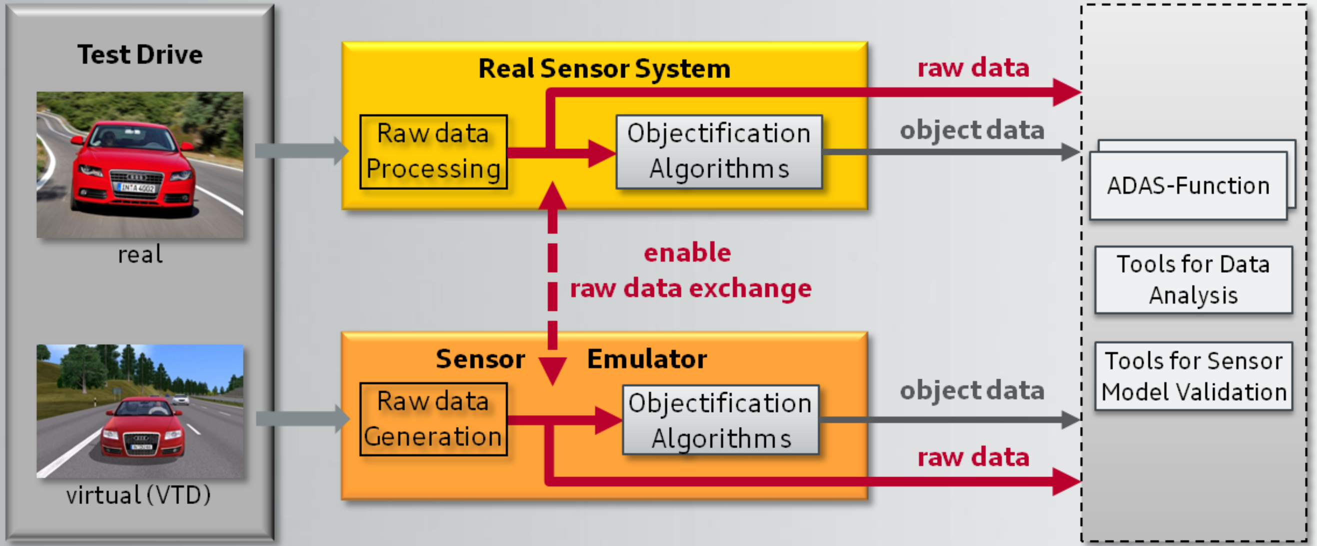
- **Simultaneous execution** of multiple perception sensor emulators with **realistic distortion effects**

- **Share a common simulation infrastructure for sensor data consistency**
  - Scenario description
  - Object, material and emitter (light) data
  - Communication
  - Configuration



Sensor Emulator Requirements - Architecture

- Support the same data formats and interfaces as the real sensor



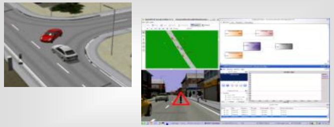
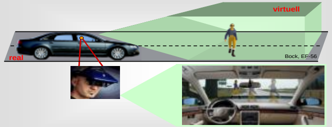
Simplified integration into the existing ADAS development and testing process

Sensor Emulator Requirements – Simulation Variants

- Support various simulation types in the existing ADAS test toolchain

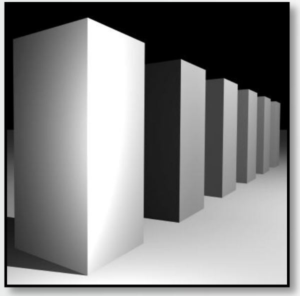
VIRTUAL TEST DRIVE + **ADTF®** (Automotive Data and Time Triggered Framework) + **MATLAB® & Simulink®**

Toolchain for the integrated development and testing of ADAS functions

Software-in-the-loop	Driver-in-the-loop	Vehicle-in-the-loop	Hardware-in-the-loop
			
slower / faster than real-time	soft real-time	hard real-time	hard real-time

Sensor Emulator Requirements – Level of Realism

- Extensible architecture regarding refined models for a higher level of realism



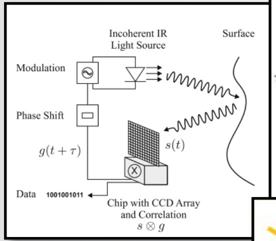
- Configurable approximation accuracy and distortion levels with a single consistent model

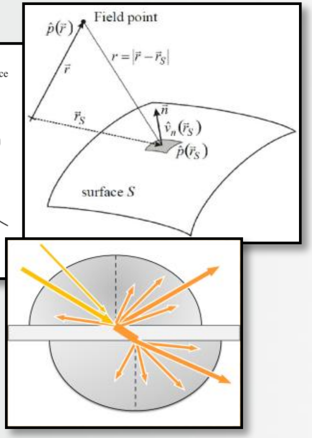
Sensor Emulation Data Accuracy

ideal	‘functional realism’	‘close to reality’
Algo. validation (SIL, faster than real-time)	ADAS system testing (HIL, hard-real-time)	Operational-envelope testing (SIL, non-real-time)

Sensor Emulator Requirements – Physics

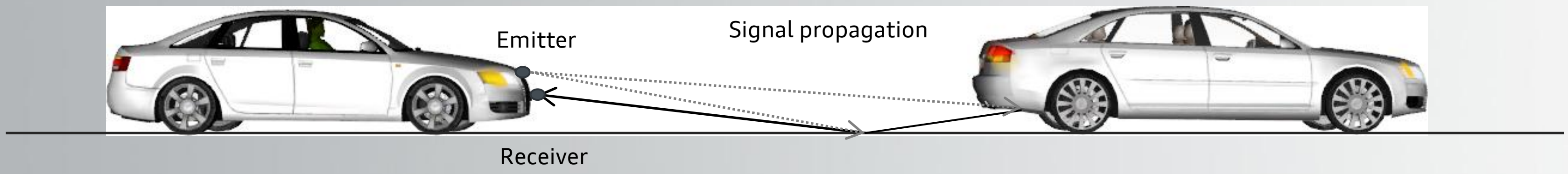
- **Particle-, ray- and wave-based physical measurement methods shall be approximated**





[Source: Keller, Kolb]

- **Physics-oriented modeling of**
  - sensor data acquisition process
  - related systematic and stochastic distortion effects
  - material, surface and emitter properties



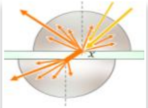
Advanced Material and Emitter Description

Multi-modal sensor simulation requires extended material and light source (emitter) descriptions

- Each material is identified by a unique ID
- Covering also non-visible light spectrum, e.g. 300 – 1000nm
- Holding meta data for material/emitter classification, lookup, …
- Storage of physical properties in form of scalars and textures
- Support for existing measurement techniques and material standards incl. accuracy information
- Material data records must be extensible

Unique Material ID

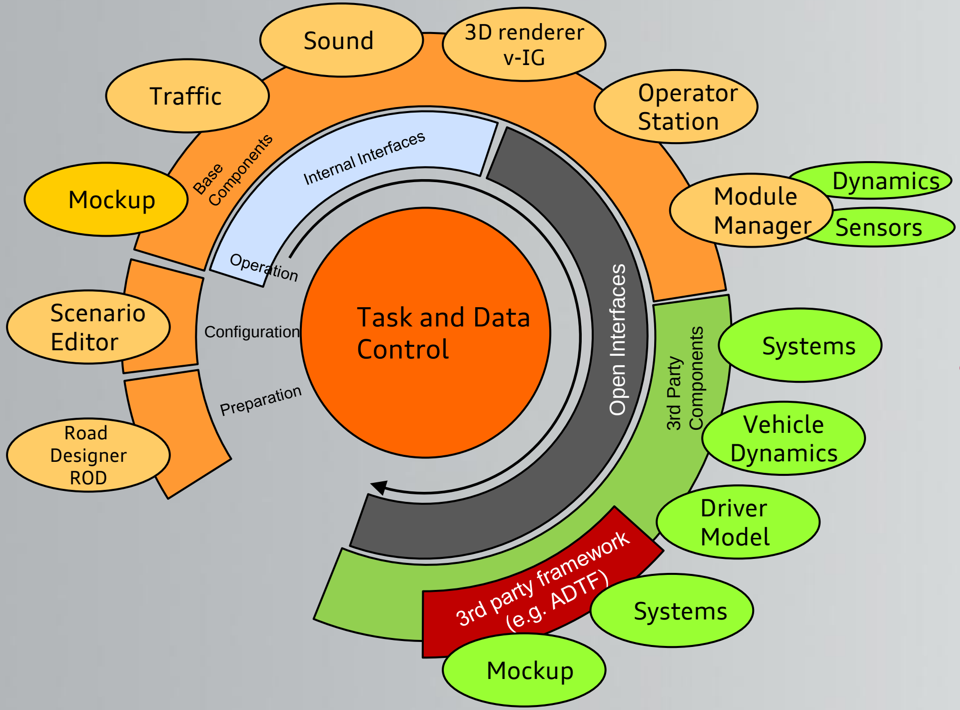
- **Parent Material ID** = Alloy
- **Meta Data**
  - Description = 'Alloy 85'
  - Class: Metal
  - Surface subclass: Coated
  - Measurement standard and accuracy info
- **Physical Parameters** [Si-Units]
  - Elec. Conductivity = 36.59 *10^6 S/m
  - Surface roughness = 120 um
- Diffuse Reflection Spectrum
- OpenGL Fallback Colors
- Software specific shaders / RT programs
- Measured BRDF/BSDF/BTF for visible, infrared, … spectrum



- Extended data

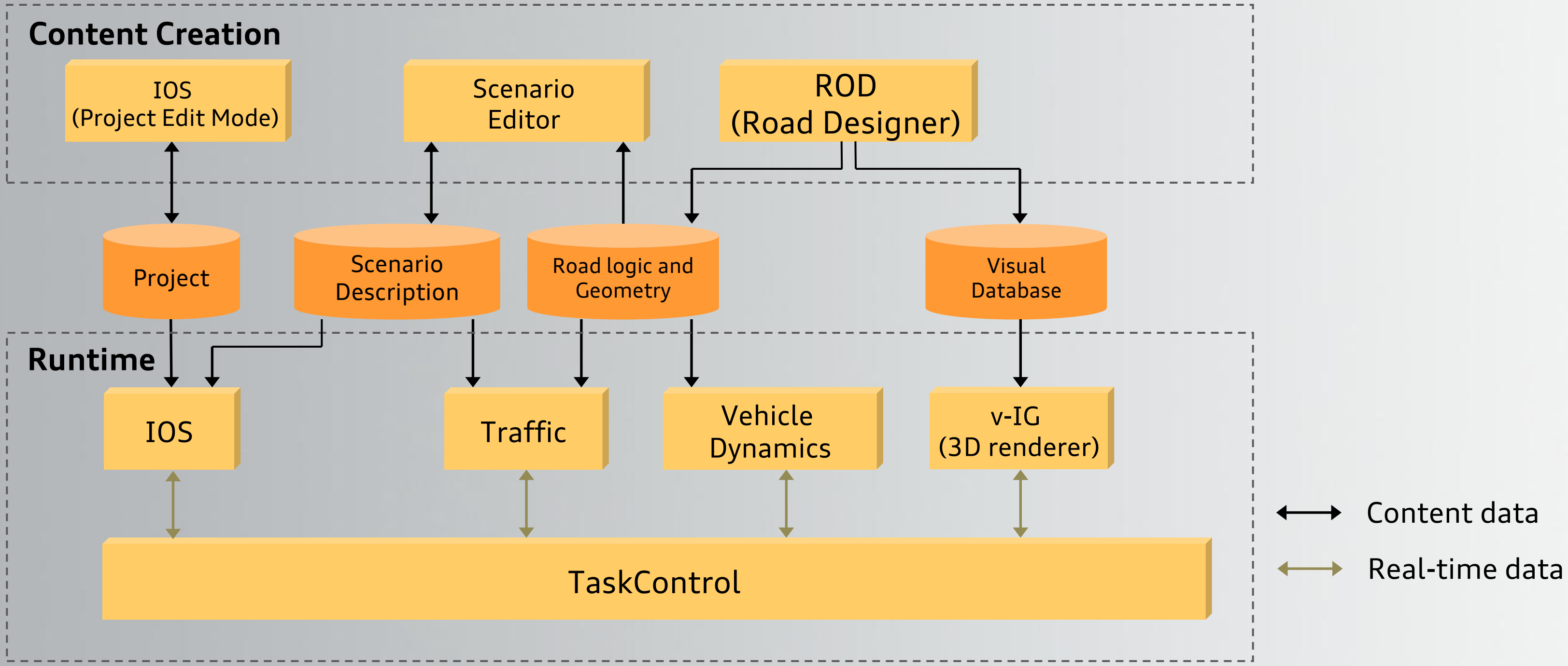
Integration of OptiX and VTD

Virtual Test Drive



- VTD core = simulation environment
  - 3D rendering (image generator)
  - traffic and scenario
  - sound
  - mockup interfaces
  - record/playback
  - event and data handling
  - content creation
  - management of custom modules
- VTD dev = development environment
  - interfaces for
    - run-time data (Run-time Data Bus – RDB)
    - event / control data (Simulation Control Protocol – SCP)
    - sensor development (using Image Generator v-IG)
  - module development via
    - library
    - C++ API

VTD – Content Creation Toolchain



v-IG

- Open Scene Graph (and OpenGL) based 3D renderer
- Part of VTD but also available as a standalone renderer
- Provides an API
- Used in driving, train and flight simulators
- Used in sensor simulation applications

Standard day scene (OpenGL)

Sensor image for hardware-in-the-loop simulator (OpenGL)

HDR night scene with wet road (OpenGL)

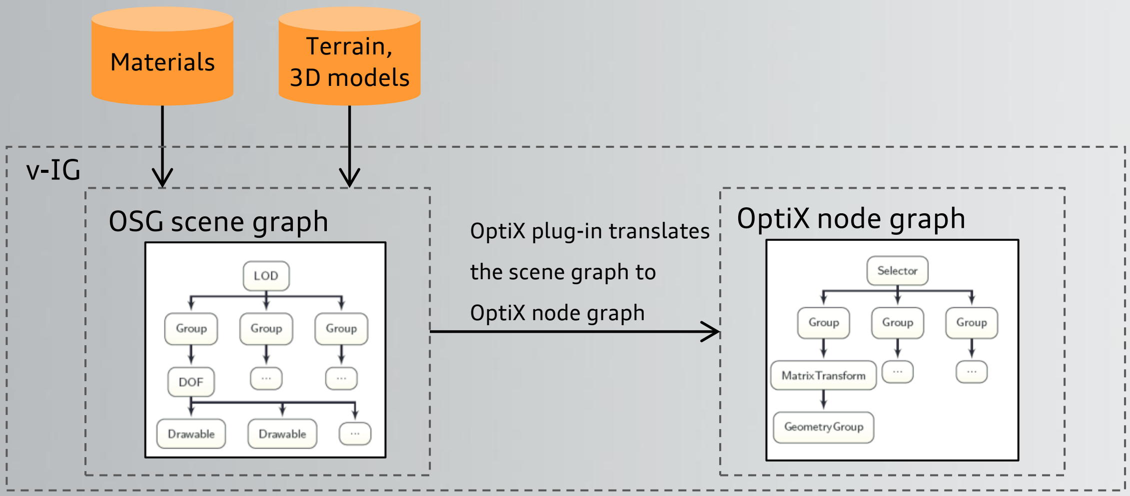
OptiX plug-in

- Conversion of OSG scene to OptiX scene
  - Geometry, Materials
- Synchronizing OSG/OptiX scenes
  - Animations, LOD, Lights
- Post Processing
- Real-time data transfer

- C++ API provided for customization
  - New camera models with Cuda/C++
  - Different buffer formats, multiple output buffers
  - Custom light sources
  - Building post processing pipelines
- Adding/editing materials

Creating the OptiX node graph

- v-IG loads the scene and creates an OSG scene graph
- OptiX plug-in translates the OSG scene graph to an OptiX scene graph
- OptiX specific optimizations during translation
- Some objects (e.g. Vehicles) loaded and deleted in run-time

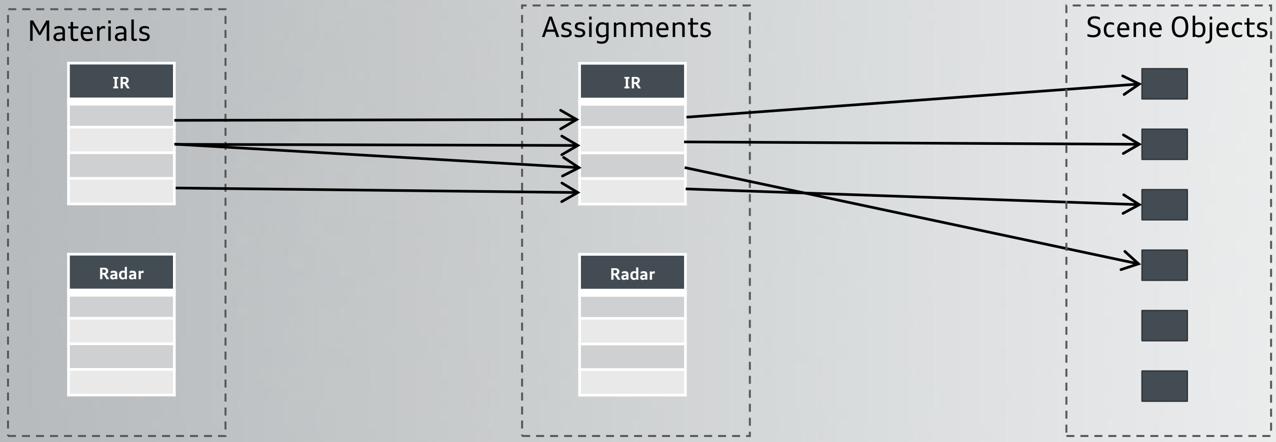


Material management

- XML material definitions
- Grouped according to wavelength and/or sensor type

- Associates materials with objects
- Grouped according to wavelength and/or sensor type

- Identified by textures or ID‘s
- v-IG assigns the materials to OSG scene graph nodes
- OptiX plug-in creates OptiX materials and puts into the material buffer



Vorsprung durch Technik

Material Management

- Material definitions in XML
- Common materials for rasterizer and ray-tracer
  - Shader params will be GLSL uniforms in rasterizer and OptiX variables in ray-tracer
  - Rasterizer loads GLSL programs, OptiX ptx files
- New materials can be derived from existing ones

Sample material decleration

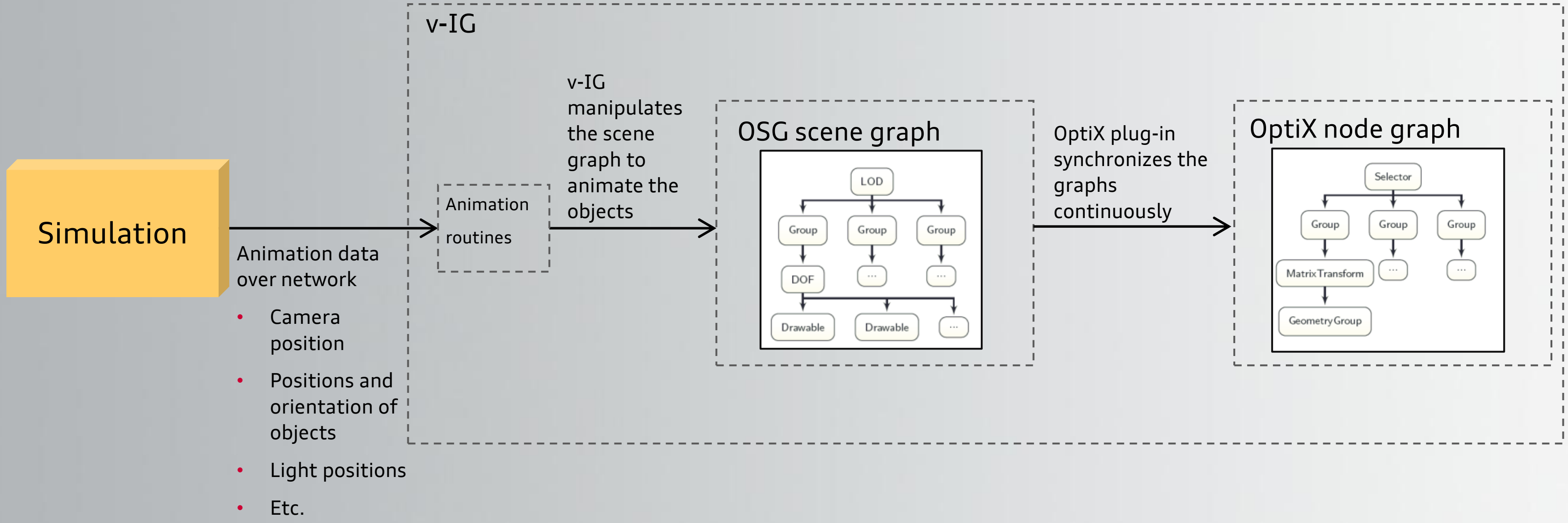
```
<Material name="Audi_PhantomBlack_RT" >
  <GeneralParams ambient="0.01 0.01 0.01 1.0"
                 diffuse="0.02 0.02 0.02 1.0"
                 specular="1.0 1.0 1.0 1.0"
                 emissive="0 0 0 1" shininess="100" />
  <ShaderParams>
    <Param type="vec4" name="u_genericConfig" value="0.2 0.5 1 1" />
  </ShaderParams>

  <FragmentShader file="../data/Shaders/vehicleBodyFrag.glsl" />
  <VertexShader file="../data/Shaders/vehicleBodyVert.glsl" />
  <OptiXHitProgram file="../data/Cuda/vehicleBody.ptx" />

</Material>
```

Copy and override material properties

```
<Material name="Default_Rim_RT" copy="Default_Rim" >
  <OptiXHitProgram file="../data/Cuda/chrom.ptx" />
</Material>
```

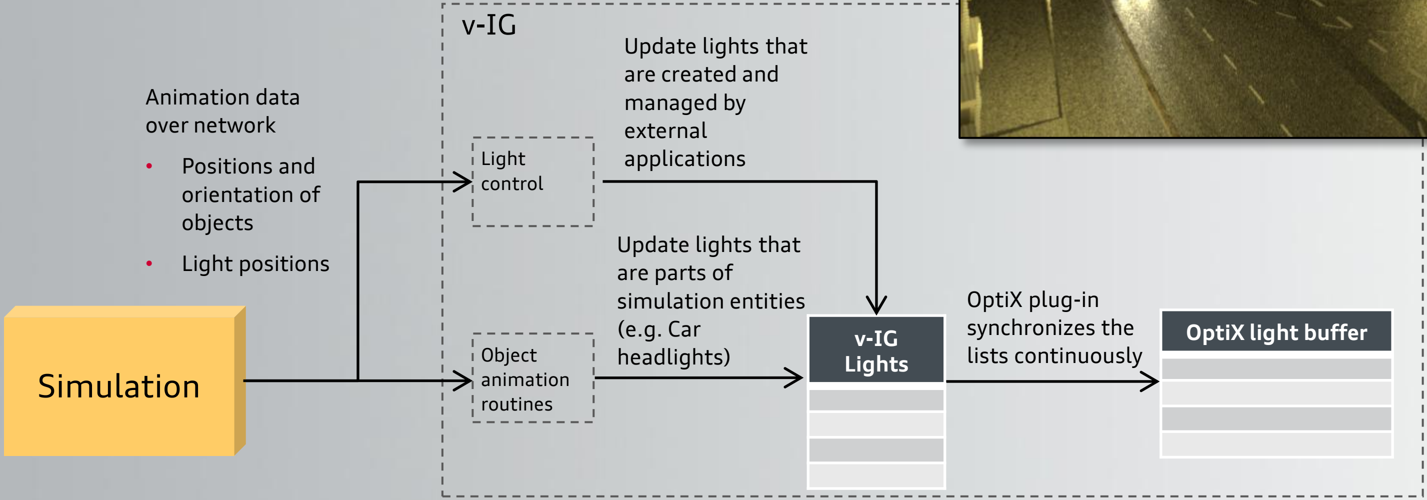
Synchronization of scenes

- v-IG manipulates OSG scene graph nodes (e.g. DOF´s) for animations
- LOD nodes are automatically updated by OSG
- OptiX plug-in monitors function nodes and synchronizes their OptiX counterparts



Lights and emitters

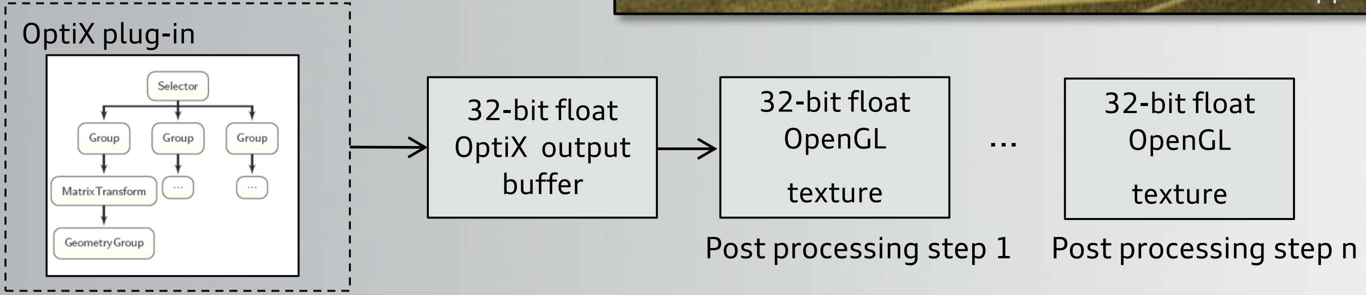
- v-IG creates and manages a list of lights
  - Some lights are generated automatically with the information stored in the terrain and models (car headlights, street lamps)
  - Communication protocol allows for creating and controlling lights externally
- OptiX plug-in synchronizes OptiX light buffer with v-IG lights
- Lights can represent any type of emitter



Post processing

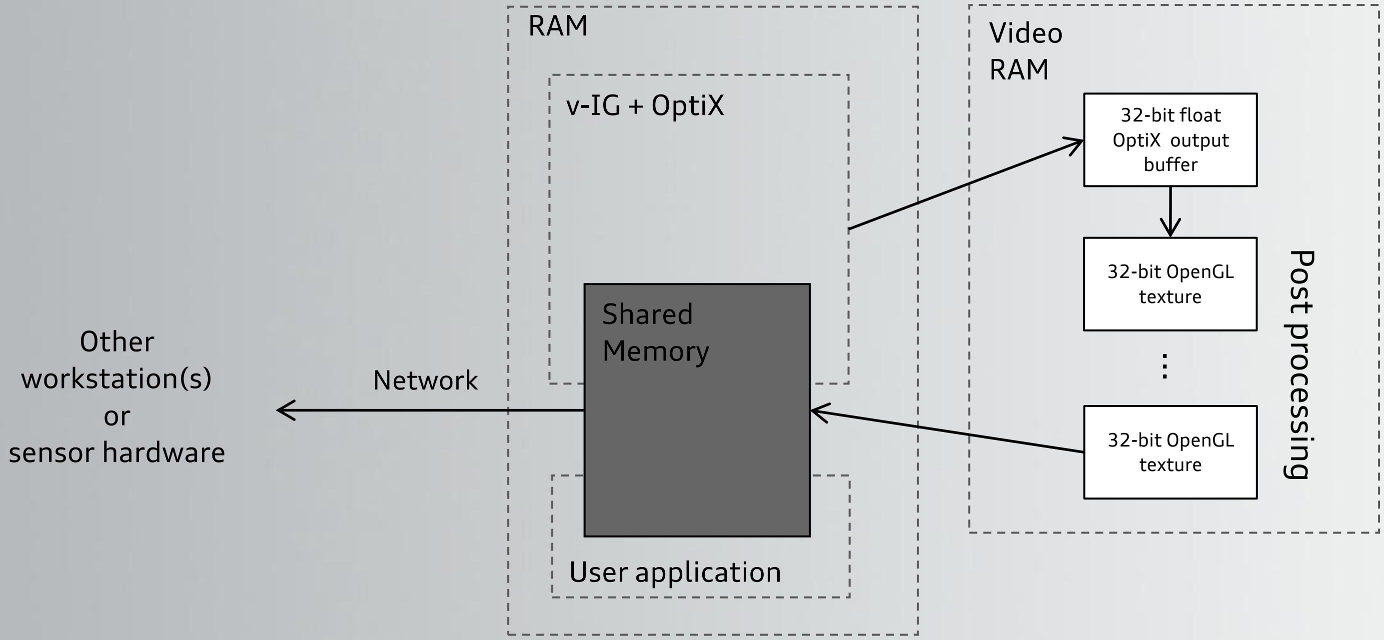
- Rendered buffers can be fed into v-IG post processing pipeline
  - Programmable through API
  - Post processing with GLSL shaders
  - 32 bit floating point
- Motivations
  - Bloom
  - Noise
  - Tone mapping
  - Etc.


Tone mapping and blooming



Real-time data transfer

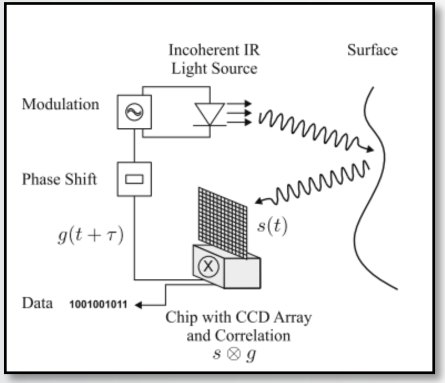
- Rendered buffers are made available to external applications
- Shared memory or network
- Producing data for hardware-in-the-loop and software-in-the-loop simulations



Sensor Model Examples

using OptiX + VTD

Photonic Mixing Device (PMD) Sensor Model



[Source: Keller, Kolb]

- The PMD-sensor uses the Time-of-Flight principle for measuring intensity and depth data of a 3D-scene with modulated infrared (IR) light
- Important systematic and stochastic distortion effects
  - non-ambiguity range
  - extraneous light sensitivity
  - “flying pixels”, motion blur

colorized depth-image

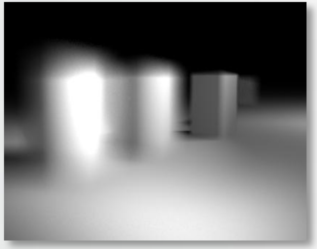
- Three-step sensor data simulation

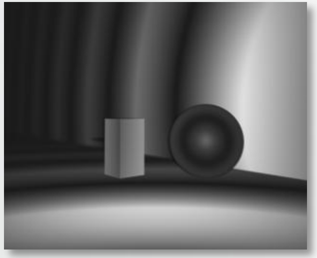
raw sensor data computation (OptiX) → raw data → depth reconstruction (OptiX) → depth data → signal processing / objectification (ADTF) → object data

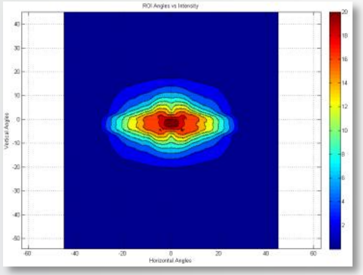
Audi Vorsprung durch Technik

PMD Sensor Model

- **Approximation techniques in the current PMD sensor emulator**
  - Multiple rays per pixel with stratified sampling
  - Simulate the angle-dependent emission characteristics of the modulated IR-light source based on Radiometry measurements
  - Phong reflection model in combination with measured IR-material reflection values



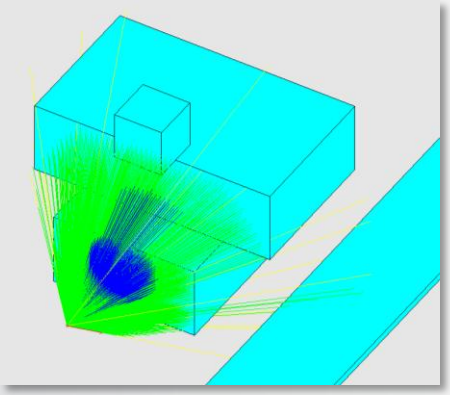




Ultrasonic Sensor Model

- Currently: ideal acoustic wave propagation model

- Modeling requirements
  - Computation of primary-, secondary and cross-echo
  - Efficient computation of up to 20 ultrasonic sensors on a single GPU
  - Consideration of target object material class (e.g. vegetation)



Test scene in MATLAB

16 circularly-arranged ultrasonic sensor „depth maps“ simultaneously rendered with OptiX

Sensor Emulator Validation and Verification Stages

Real Sensor System

Raw data Processing | Objectification Algorithms

- Sensor test bench
- Real test drive
- Issue recordings DB

OptiX + ADTF Sensor Emulator

Raw data Generation | Objectification Algorithms

Real-time capable

- x-in-the-loop variants

MATLAB + ADTF Sensor Emulator

Raw data Generation | Objectification Algorithms

Non-real-time, high fidelity

- MATLAB ray-tracer for raw data generation

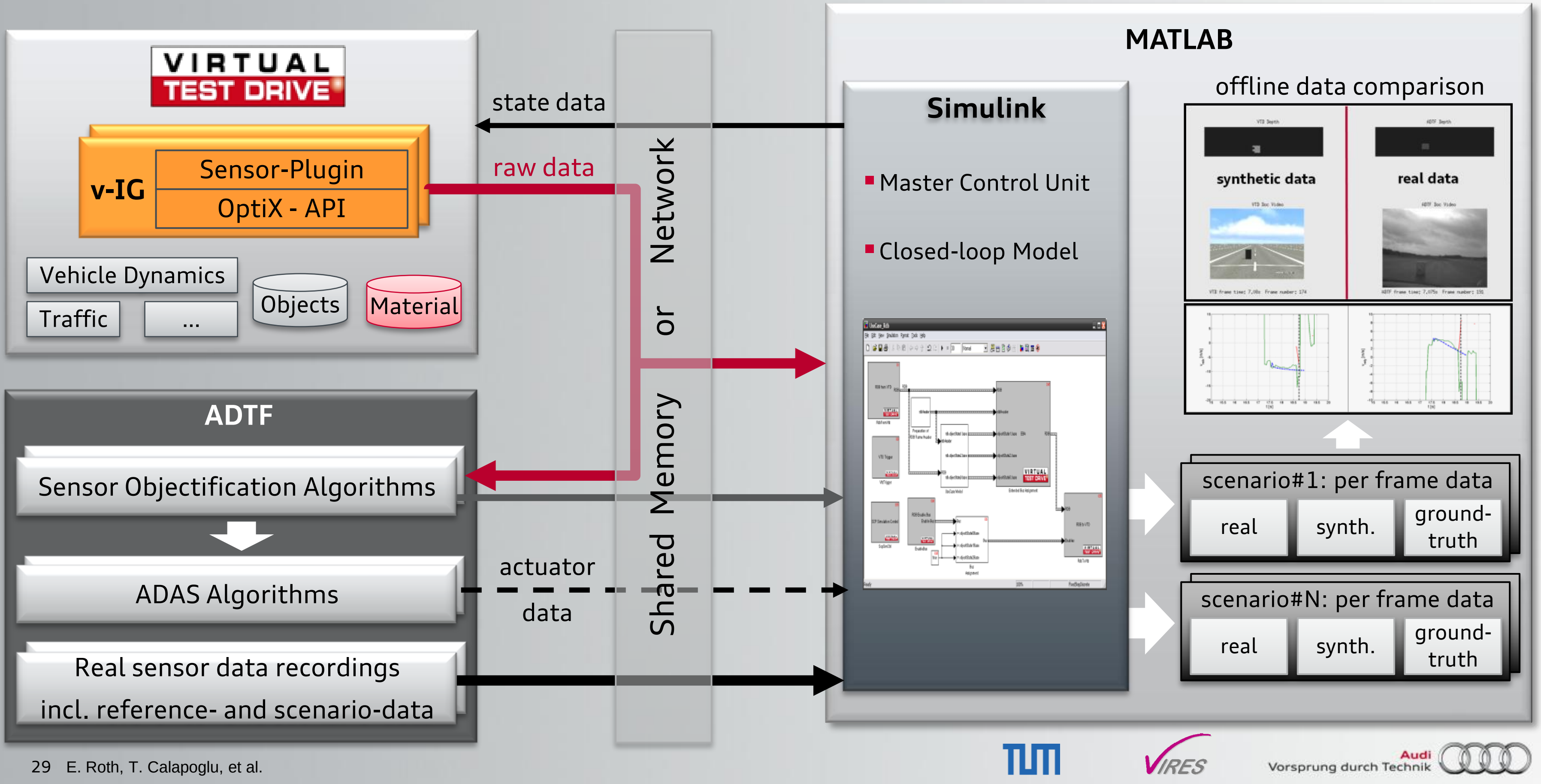
2. Verification
Testing with real sensor data

Comparison based on reference data & virtual scene generation

1. Validation
Prototypical Testing

Comparison based on static + dyn. description of „analytical scenes“ using XML

Sensor Emulator Validation and Verification Toolchain



Summary

- We showed our approach for supporting ADAS algorithm and function testing by using Virtual Test Drive and OptiX for multi-sensor data simulation

- Related requirements and implemented concepts for realistic multi-sensor simulation
  - Physics-oriented sensor modeling using OptiX
  - A common sensor-model simulation infrastructure
  - Advanced material and emitter specifications
  - Validation and verification process

- Ray-tracing with OptiX seems to be a reasonable platform. However, we are just at the beginning ...

AUDI ADAS Demonstrator

Outlook

Challenges to be tackled in the future regarding ...

- **A standard for multi-spectral material and emitter specifications**
  - Simulation software independent description and identification scheme
  - Physical property handling of materials and emitters, e.g. for wavelengths 300 – 1000nm
  - Support for different measurement data formats and standards

- **OptiX**
  - Support for large scenarios (1000x of objects, 100x materials, 10x sensor models, ...)
  - Improved multi-GPU scalability for 60 Hz and higher
  - Improved OptiX debugging, profiling and optimization tools

Thank you very much.

Get in touch with us, if you are also using OptiX for sensor simulation!

Contact:

Erwin Roth, Technische Universität München
erwin.roth@in.tum.de

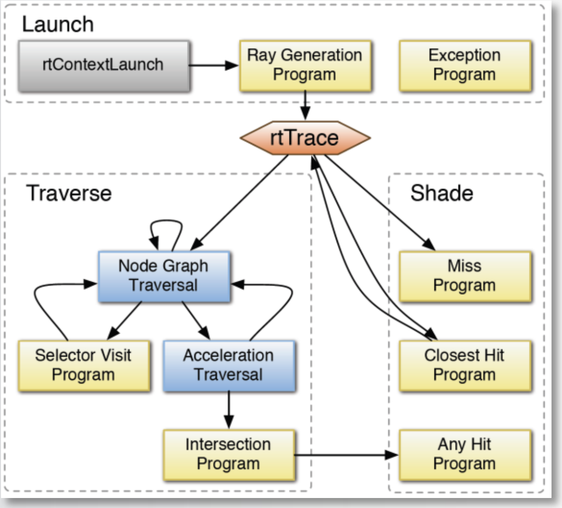
Tugkan Calapoglu, Vires Simulationstechnologie GmbH
tugkan@vires.com

BACKUP

Why was NVIDIA OptiX selected?

- Since Intel's Larrabee was never released ;-)

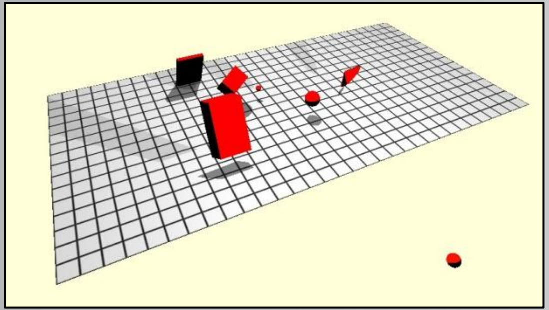
- Programmability and Flexibility of OptiX's Ray Tracing pipeline
  - Customizable Ray Tracing pipeline
  - Focus on mathematical model rather than 3D programming
  - Many core, multi-GPU scalability
  - Availability for different platforms



[Source: NVIDIA]

- AUDI was already using the Virtual Test Drive (VTD) simulation system
  - We decided to extend the OpenSceneGraph-based 3D-renderer of VTD with an OptiX-plugin
  - Allows us to reuse most of the existing rendering and simulation infrastructure

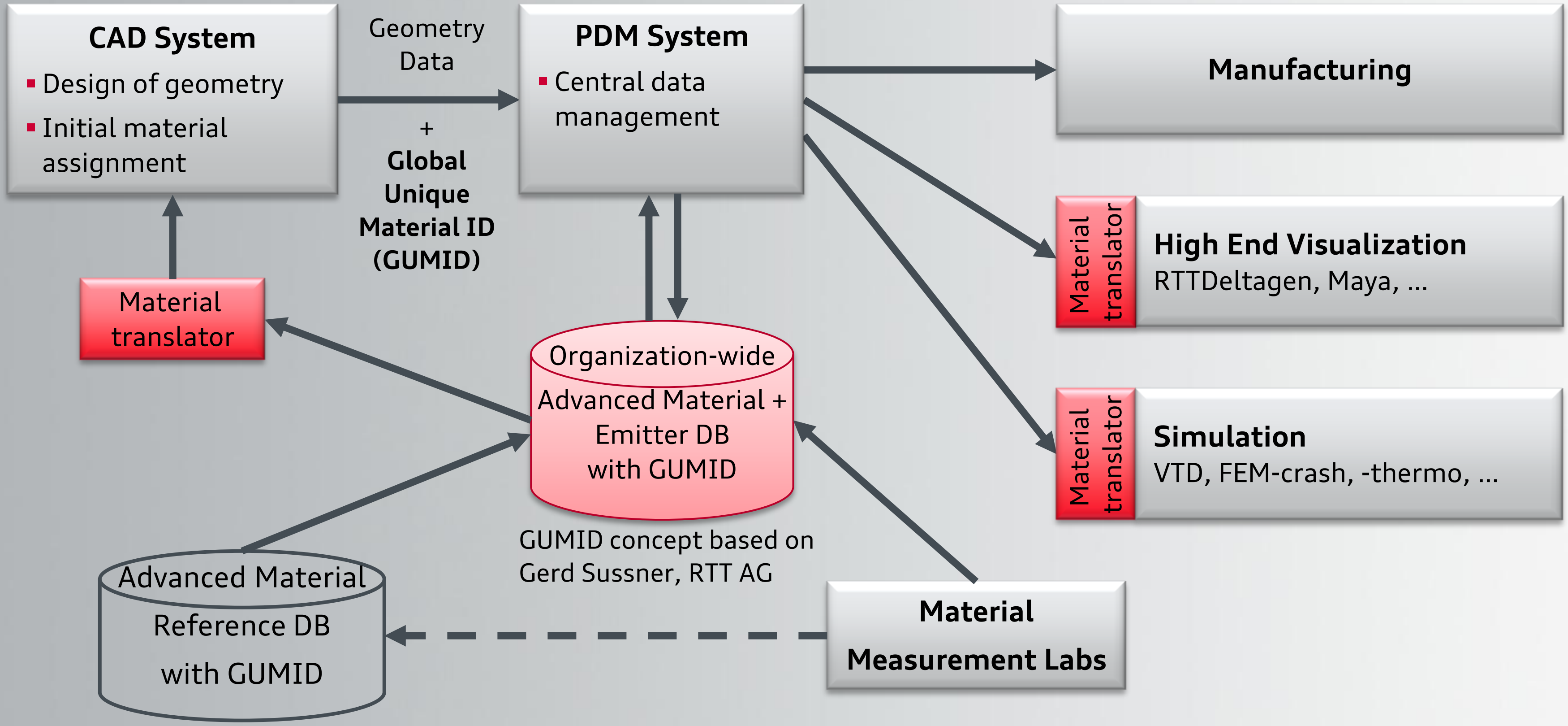
XML-Scene Data Interchange Format



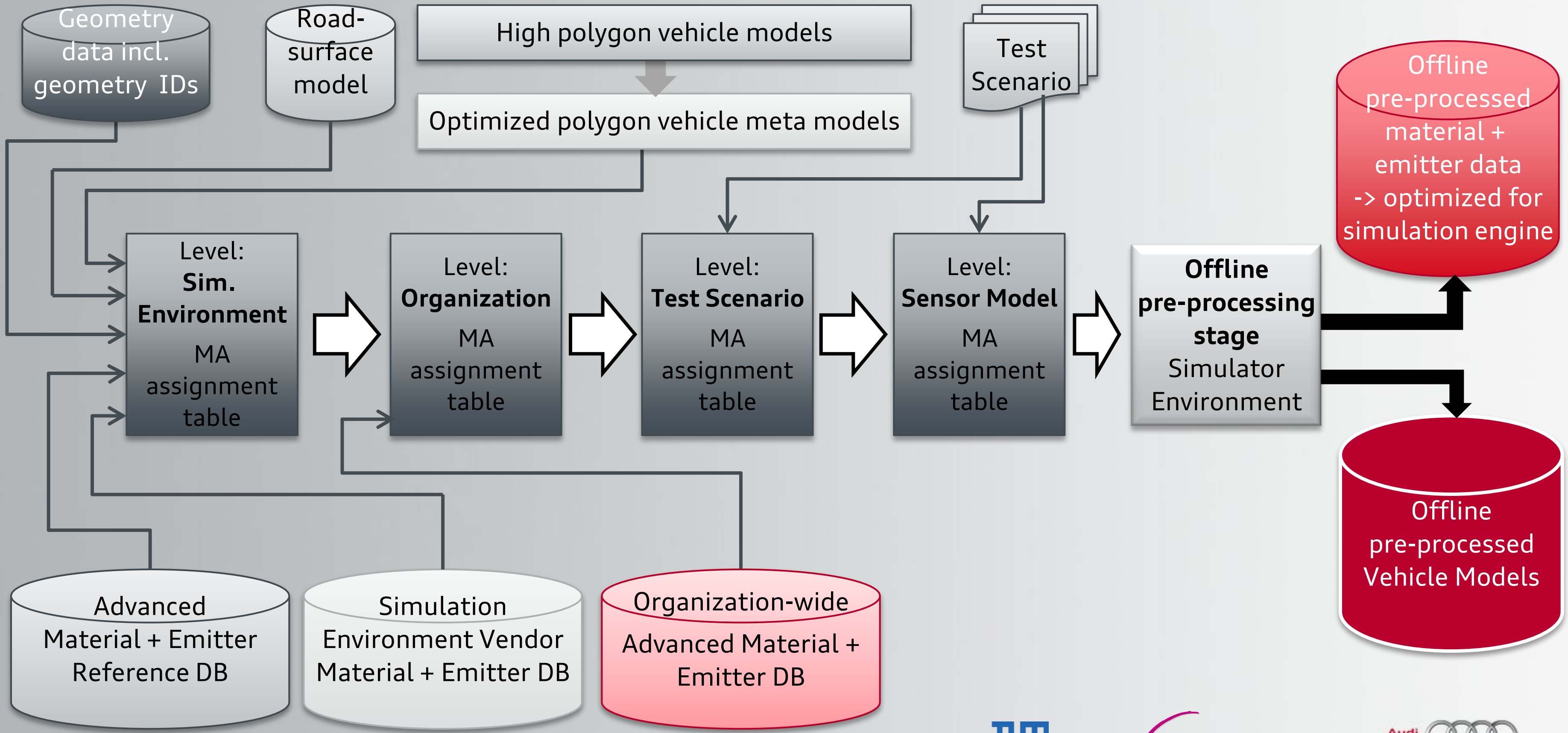
```xml
<OptiXRW>
    <CameraAndLights>
        <Lights />
        <PinholeCamera eye="0 0 0" lookat="0 0 0" up="0 0 0" vfov="0" />
    </CameraAndLights>
    <CommonContextSettings>
        <declareVariables>
            <declareVariable name="output_buffer" type="buffer" width="1024" height=" ... />
            <declareVariable name="eye" type="float3" values="0.000000 0.000000 0.000000" />
            ...
        </declareVariables>
    </CommonContextSettings>
    <Resources>
        <Accelerations>
            <Acceleration id="2" builder="Sbvh" traverser="Bvh" />
            ...
        </Accelerations>
        <TextureSamplers>
            <TexturSampler id="25" filename="" color="" />
            ...
        </TextureSamplers>
        <Materials>
            <Material id="46">
                <declareVariable name="texUnit0Sampler" type="texture" values="47" />
                <declareVariable name="materialSpecular" type="float3" values="0.092235 ... />
                <declareVariable name="materialEmission" type="float3" values="0.000000 ... />
            </Material>
            ...
        </Materials>
        <Geometries>
            <Geometry id="38" PrimitiveCount="84" />
            <Geometry id="93" PrimitiveCount="2165" />
            <Geometry id="111" PrimitiveCount="61558" />
            ...
        </Geometries>
        <GeometryInstances>
            <GeometryInstance id="37" gid="38">
```

- File format for OptiX node graphs
  - XML based format
  - Easily readable and editable
  - C++ library for loading/saving
  - Can be integrated to other software
- Motivations
  - Sensor model validation
  - Debugging

Integrated Material Handling



Multi-Sensor Simulation Material pre-processing Pipeline



Advanced Emitter Description

- **Simulating interference effects on sensors** requires models of ego- and extraneous-emitters
- Examples for emitters:
  - Vehicle headlights, traffic lights, street lamps
  - Emitters of active sensors (RADAR, Infrared light source, …)
  - Car2X-Transmitters
- Related emission characteristics should be stored in physically measurable SI units using Radiometry in order to not cover the visible light spectrum only
- The specific emitter characteristics should be stored in an organization-wide database with a unique emitter ID

Unique Emitter ID

Parent Emitter ID =

Meta Data
- Brief description
- Class: ‚Vehicle Headlight'
- Emission spectrum: min, max
- State flags
- Number of sub-emitters: 2
- Sub-Emitter1
  - Position, Orientation relative to Ref. Coord. Sys.
  - Number of state profiles = 3
  - State-Profile1
    - Emission map Format
    - Emission map data
  - State-Profile2
    - …
- Sub-Emitter2

OpenGL fallback illumination model
- Frustum data
- Phong intensity data
- Attenuation