IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Vehicular Networks Simulation with Realistic Physics

ESTEBAN EGEA-LOPEZ[1], FERNANDO LOSILLA[1], JUAN PASCUAL-GARCIA[1] AND JOSE MARIA MOLINA-GARCIA-PARDO[1]

[1]Department of Information Technologies and Communications, Universidad Politécnica de Cartagena (UPCT), Cartagena, 30202, Spain, (e-mail: {esteban.egea, fernando.losilla, juan.pascual, josemaria.molina}@upct.es)

Corresponding author: Esteban Egea-Lopez (e-mail: esteban.egea@ upct.es).

This work was supported by the Spain Ministerio de Economía, Industria y Competitividad under grants TEC2016-76465-C2-1-R and TEC2016-78028-C3-2-P.

ABSTRACT Evaluation of cooperative automated driving applications requires the capability of simulating the vehicle and traffic dynamics as well as the communications with a level of accuracy that most of current tools still lack. In this paper we explore the use of game engines in hybrid traffic-network simulators. We describe and validate a novel framework based on this approach: Veneris. Our framework is made of a traffic simulator, implemented on top of the Unity game engine, which includes a realistic vehicle model and a set of driving and lane change behaviours adapted to a 3D environment that reproduce real-world traffic dynamics; a ray-launching propagation simulator on graphics-processing-unit (GPU), called Opal, and a set of modules which enable bidirectional coupling with the OMNET++ network simulator. The more relevant and novel mechanisms of Veneris are introduced, but further implementation details can be checked on the source code provided in our repository. We discuss the validation tests we have performed and show how it provides accurate results in three key areas: the fidelity of the vehicle dynamics, the recreation of realistic traffic flows and the accuracy of the propagation simulation. In addition, general results of the expected performance are provided.

INDEX TERMS Game engine, GPU, radio propagation, ray tracing, simulation, traffic, vehicular networks.

I. INTRODUCTION

Connected vehicles extend the capabilities of multiple advanced driver-assistance systems and automated vehicles by enabling the possibility to perform cooperative actions (Cooperative Automated Driving, CAD) or increase the awareness of the vehicle sensor systems [1]. CAD can improve safety and efficiency, by introducing Cooperative Adaptive Cruise Control (CACC) applications [2], including not only platoon driving, but also cooperative collision avoidance [3] and others. CAD, however, requires further research before being deployed [1], [2] and simulation is a key tool for its desig and evaluation, because of both the safety-critical features of the applications and the cost and resources required for real test and validation.

Evaluation of CAD applications requires the capability of simulating both the vehicle and traffic dynamics as well as the communications. The level of realistic detail of the simulation determines the kind of application that can be tested. On the one hand, tools for simulation of highly realistic driver behaviour, vehicle dynamics and sensors [4], called *nanoscopic vehicle simulators*, are available but, due to their cost, are used mainly by the automotive industry and very specialised research groups and are limited by the number of simulated vehicles that can handle [5]. On the other hand, there are a number of *traffic* [6] and *network* [7] simulation tools currently available for the general research community. After being focused on their respective domains, a number of new frameworks have combined them into so called *hybrid* simulators [8] in the last decade.

However, most of them still lack the level of detail to perform accurate simulations of many CAD applications, as discussed in several similar prior works [5], [9], [10]. In particular, *microscopic traffic simulators*, which use car-following models, do not reproduce realistic vehicle dynamics. Consider as a representative example the simulation of safety and emergency applications: microscopic simulators are not directly suitable to model accidents because they use mobility models that are specifically designed to avoid vehicle crashes. Therefore, those models have to be at least modified properly, which is actually a subtle task, as dis-

cussed in our previous work [11], and makes difficult setting up controlled experiments.

Moreover, in the emergency case, the issue is not just to be able to hack a car-following model to allow the occurrence of collisions but to reproduce realistically the dynamics of a vehicle in a collision or during an evasive maneuver. Another characteristic example is CACC; in this case, more realistic longitudinal dynamics have been introduced in some tools [10], but the lateral dynamics should also be taken into account in many applications of interest, such as cooperative merging, or more obviously, if one is interested in the safety of cooperative lane changing, for instance.

Highly accurate physical behavior can be incorporated with the use of physics engines, which are simulation tools already used in robotics and other fields [12]. However, direct integration of dedicated physics engines in current traffic tools is not trivial and possibly not even advisable, due to the different goals of both approaches. Recent solutions include extending microscopic simulators with more physically realistic but still simplified models [10] or using the physics engine indirectly via an interface provided by *robot simulators*, coupling network and robot simulator in another type of hybrid simulator [9]. The result is, in the first case, only a limited increase in physical accuracy and, in the second case, the loss of the capability of reproducing realistic traffic patterns, unless the car-following, lane-changing and other features of traffic simulators are reimplemented for the robot simulator.

There is an alternative, not yet fully explored and evaluated. Game engines [13], [14], mainly focused on rendering, also include powerful physics engines. Although they have been traditionally proprietary software, many have been made available to the public in the last years, and have become increasingly used in several areas of research. Combining network simulators and game engines provides an alternative type of hybrid simulator for CAD research. This solution is similar to the robot-network hybrid simulation previously discussed [9] and in fact suffers the same problem: the need to reimplement traffic models. But the use of game engines brings additional possibilities and advantages: their capabilities as general-purpose three-dimensional (3D) simulations [13], with rich interactivity between the components and, in particular, with the users, supporting seamlessly human-in-the-loop simulations; as well as very flexible development interfaces, intuitive graphical design and multi-platform support.

The 3D representational capability gives game engines a clear advantage over other alternatives with respect to the other area where the network aspect of CAD simulations can be improved: radio propagation. Communications have a key influence on the stability of CACC [15] and other CAD applications, which require accurate radio modeling. Indeed, the simulation of propagation mechanisms with high fidelity requires as a previous step the capability to build 3D models of the scenario [16], [17], but network simulators do not usually provide it. Moreover, the most accurate methods, those based on ray tracing [16], [17], have been usually too computationally costly to be used in large and *dynamic* scenarios [17], and so networks simulators usually only provide stochastic or simplified hybrid methods [17]. However, in recent years the computational power of graphics processing units (GPU) has been leveraged to simulate propagation [16], [18] at a much more reasonable computational cost. This approach also favors game engine over other alternatives, since they integrate in one or another form GPU programming capabilities. That is, game engines can be used to seamlessly bring 3D models and GPU offloading to network simulators.

In this paper we explore this alternative, that is, the use of game engines in hybrid traffic-network simulators. We describe and validate a novel framework based on this approach: Veneris, short for Vehicular Networks Simulator with Realistic Physics. Our framework is made of (1) a traffic simulator, implemented on top of the Unity game engine [14], which includes a realistic vehicle model and a set of driving and lane change behaviours that reproduce the traffic dynamics; (2) a ray-launching GPU based propagation simulator, called Opal, and (3) a set of modules which enable bidirectional coupling with the widely used OMNET++ network simulator [7]. The more relevant features of the framework are described and discussed in relation to the related work. In particular, we introduce a novel duplicate ray filtering algorithm for ray-launching methods; a lane change selection mechanism based on cost-functions and adapt the IDM [19] and MOBIL [20] models to a 3D environment.

We provide the research community with a flexible tool for realistic testing of CAD applications. The source code of Veneris is freely available at our repository[1], where all the implementations details can be checked. In addition, the scenarios used for the validation can be downloaded and executed.

In Section II, we further motivate our work and discuss the aforementioned issues and related works. In Section III we describe the framework components, and discuss the more relevant features and novel mechanisms introduced. The key aspects of Veneris, that is, the fidelity of the vehicle dynamics, the recreation of realistic traffic flows and the accuracy of the propagation simulation are validated in Section IV. Conclusions are summarized in Section V.

II. RELATED WORK

High fidelity tools with software, driver, vehicle and hardware in-the-loop [4] are used by the automotive industry. Due to their proprietary nature and high cost, in this work we focus our discussion on freely available tools for the general CAD research community.

Hybrid traffic-network simulators interact bidirectionally by exchanging messages. Widely used tools such as Veins, iTETRIS or VSimRTI, have been discussed in [8]. All of them use a car-following based microscopic traffic simulator, either SUMO [6] or VISSIM [8]. Extending them with real-

[1] http://pcacribia.upct.es/veneris

istic physics, although possible in a limited way as in [10], is complex, and in fact may disable their main advantage: these tools use a simple unidimensional (1D) representation of vehicles and scenarios which allow them to cope with thousands of vehicles simultaneously. Introducing a 3D physically constrained representation of vehicles and scenarios would also require an almost complete rewriting of the code. Extensions of current tools include Plexe [10], which extends both Veins and SUMO with CACC controllers and a more realistic engine model, though limited to longitudinal dynamics.

Physics engines have been used for vehicle dynamics in [9], where authors couple the network simulator ns-3 and the robot simulator Webots in a new hybrid simulator. However, the robot simulator does not include adequate driving and lane changing behaviours and provides only limited capabilities for scenario building. Therefore, this approach is useful to study CAD applications only in simple scenarios. When using a physics engine on a small scale we obtain more physically correct results in the simulation, but it is not clear if they are still representative on a larger scale. To answer that question, unlike in [9], in this work we not only couple the network and vehicle simulator but also implement widely used car-following [19] and lane changing models [20], adapting them for 3D environments. We also test and validate our implementation by comparing our results with real traffic data in Section IV-B. The validation of synthetic mobility traces is difficult [21], due the lack of available real traces [22]. We have followed a methodology similar to the one proposed in [23] based on publicly available data, provided by the navigation and directions services of Google [24].

Game engines are increasingly been used in diverse research fields [13], mainly in artificial intelligence and computer vision [25], due to their flexibility to build interactive 3D environments. Among the available engines, we have used here Unity [14], a popular game engine, due to its moderate learning curve and multiplatform support. It has been used in similar studies, as in [26], where authors characterize the channel of vehicular communications. However, they use directly the ray tracing capabilities of the Unity engine, which use the CPU, remarkably affecting the performance and the scalability. On the contrary, with Opal, we offload the ray tracing to the GPU, using NVIDIA Optix [27], an optimized library for parallel ray tracing on the GPU, while using the CPU for traffic simulation.

Ray launching methods for propagation simulation go back several decades, but recent advances including the use of GPU are discussed in [16]. Such techniques are one of the most accurate tools to estimate the propagation phenomena, but their computational cost has been traditionally a barrier for their use, so network simulators use mainly stochastic models [8]. There are hybrid models, such as GEMV^2 [17], where a simplified representation of the objects, just the outlines of buildings, is used together with stochastic methods. The model is very specific, aimed at vehicular network simulations, but the results reproduce real propagation accurately at a reasonable cost. Our approach is general and provides similar results with better performance. The most similar work to Opal is found in [18], where authors also use NVIDIA Optix to perform the ray tracing. Although similar in several aspects, we use a different implementation and introduce a novel ray duplication filtering mechanism, described in Sect. III-D. Unlike Opal yet, they include diffraction but it is not addressed how it is integrated with higher layer protocols.

In summary, although realistic vehicle dynamics and propagation features have already been implemented to some extent previously in separated works, with Veneris we leverage game engines to integrate them in a novel way and adapt driving models to a 3D environment, providing a different hybrid framework. In the following sections we further discuss and compare more particular details of Veneris with related works.

III. VENERIS FRAMEWORK

In this section we describe and discuss the modules that make up the Veneris framework. Veneris is made of a set of tools that provide different functionality and can interact with each other. The main components, shown in Fig. 1, are:

- **Veneris simulator**. A set of Unity [14] components that provide a realistic microscopic road network simulation in an interactive 3D environment. The components have been grouped in functional modules. Builder components are used to generated the scenario elements: roads, intersections, traffic lights or buildings. Vehicle components include a model of the dynamics of the vehicle and components which model the behavior of the vehicles on roads, intersections and the interaction with other vehicles. Communication components implement the communication with simulation modules. Managers handle different aspects of the simulation globally.
- **Opal**. A ray-launching based, deterministic RF propagation simulator, implemented in C++ with NVIDIA Optix [27], a library and application framework for high performance ray tracing on the GPU.
- **Veneris OMNET++ modules**. A set of OMNET++ and INET [7] modules which enable bidirectional coupling between the communication network and the traffic simulation.

A. SIMULATION OVERVIEW

To get an overall view of the framework capabilities and operation we describe the steps needed to set up a simulation and the different types that can be run. The different types and the tools involved are depicted in Fig. 2. The first step is to set up a simulation scenario. Scenarios are created from SUMO files (network, trips and polygons), which are usually generated from real-world map data [6], for instance from OpenStreetMap (OSM) [28], using the SUMO web wizard tool. These files are used by the Veneris builders in the Unity editor to generate the 3D representation of the scenario in a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

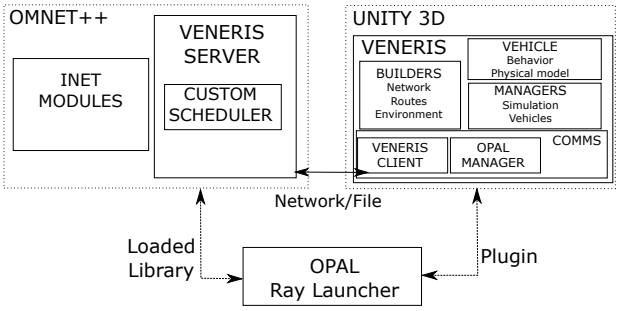


FIGURE 1: Veneris main components . Opal can be used as a standalone application, as a plugin with Veneris or loaded with OMNET++. Veneris and OMNET++ communicate through the network or by files.

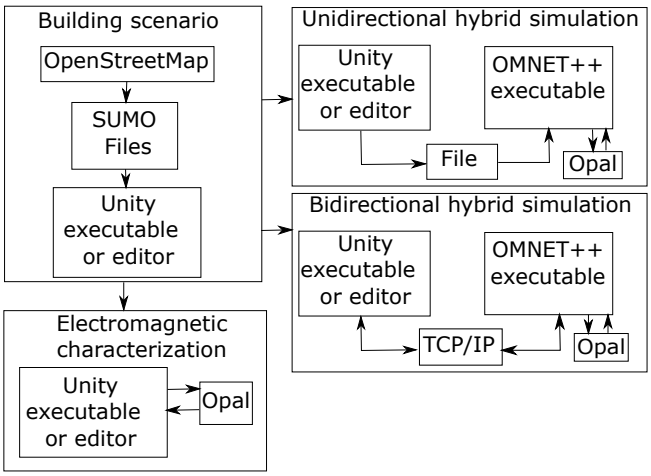


FIGURE 2: Workflow and types of simulations with Veneris and Opal. First, the scenario is built from real-world map data. Then, three alternatives are available: (1) to carry out the electromagnetic characterization of the scenario, just with Opal; (2) to run a unidirectional hybrid simulation, where the traffic simulation output is stored and then fed to the OMNET++ modules; (3) to run a bidirectional simulation, where traffic and network simulation are run simultaneously and interact with each other.

Unity scene. A vehicle manager in charge of inserting vehicles, according to the SUMO trip definitions, is also created. These vehicles use the physical model described later in Sect. III-C. A simulation manager as well as a number of additional components, such as user interface (UI), cameras and statistic recorders, can be added to the scene. At this point, a traffic simulation can be run, either directly in the Unity editor or after building an executable.

The next step is to perform a network simulation together with the traffic simulation. Network simulation is provided by OMNET++. To obtain a hybrid simulation, Veneris interacts with a running OMNET++ simulation by sending messages over a TCP connection. Basic messages include vehicle state and synchronization information. The former one is used to create, destroy and update the state (position mainly) of the vehicle modules in the OMNET++ simulation. The latter one keeps both simulations synchronized: Unity uses a real-time simulation while OMNET++ is event-based. Therefore, Veneris uses a custom scheduler to ensure that the time of OMNET++ events never exceeds the current Unity time. At this point we can perform a hybrid simulation, with all the protocol stacks available for OMNET++ as well as its propagation models.

Finally, we can use Opal to introduce more realistic propagation results. Opal is loaded by OMNET++ modules as an external library, as shown in Fig. 1. When Opal is enabled, Veneris sends to OMNET++ messages with the 3D meshes of the scenario objects as well as the updated transformation matrices of the moving objects, that are used by Opal to create and update the scene graph that is later ray-traced. Every time a packet is transmitted in OMNET++ the propagation is computed by Opal by in the GPU, taking advantage of its parallel computation capabilities, and the resulting received power is passed back to the receivers. The result is a highly realistic simulation of the traffic dynamics and electromagnetic wave propagation.

Alternatively, Opal can be directly used with Unity as a plugin with a provided interface. This mode is useful to compute the electromagnetic characterization of a scenario. For instance, to simulate the coverage of a base station on some city location obtained from real-world data. Another option is to use Opal as a radio medium with OMNET++, without using Veneris, which allow to simulate arbitary protocols and models on a 3D-aware environment. Opal can also be used as a stand-alone application, independently of Unity and OMNET++.

B. VENERIS

Veneris implements a microscopic road network simulation in an interactive 3D environment. It is implemented with the Unity Engine, taking advantage of its rendering capabilities, as well as its internal physics engine, provided by NVIDIA PhysX [29]. The functionality provided can be summarized in four major areas:

Scenario building and management. Veneris simulator leverages the outstanding capabilities of the SUMO simulator for generating road networks and traffic demands. The road network, routes and environment are generated by builder components from SUMO files (network, trips and polygon files respectively). The SUMO network elements (edges, lanes, junctions and connections) are translated into Unity 3D components and grouped in game objects, which also include a mesh renderer and a mesh collider. While the renderer is only for visualization purposes, the collider actually provides the physical interaction with other objects, such as the vehicle wheels. For each road, one or more lane and path components are added, which are used by the microscopic traffic model and other behavior components. Finally, intersection components which include stop lines and path connectors are also inserted to be used by the behavior models.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

In addition, the trips generated by SUMO are imported by a vehicle manager component, which assigns the routes to vehicles and schedules them for insertion.

Vehicle dynamics. A realistic vehicle model has been developed. It is described in detail in Sect. III-C.

Behavior models. They have been implemented as behavior trees and provide the core of the microscopic traffic simulation. Each vehicle, in addition to the vehicle dynamics, has an Agent responsible for controlling it. This Agent can be a user via different input controllers, such as mouse, keyboard or wheel and pedal set, although in general it is an automated model. In the latter case, the agent is made of different submodels:

- *Path tracker*. It makes the vehicle follow a path on a lane. It is implemented with a proportional path tracker, which makes the vehicle steer to a lookahead point at a certain distance on the path.
- *Car-following model*. It describes the action of individual drivers in response to the surrounding traffic. We have implemented a modified version of the Intelligent Driver Model (IDM) [19]. This model sets the acceleration of every vehicle as a function of the actual speed, the net distance gap and the velocity difference with the leading vehicle.
- *Lane-changing model*. We have adopted MOBIL [20], an operational lane-changing model complementary to IDM. We have only implemented strategical lane changes at the moment, that is, vehicles only do the changes required to follow their routes, and use MOBIL to decide when to perform the change. As a consequence, Veneris is not suitable to realistically simulate highways where the discretionary lane changes have a greater influence.
- *Intersection behaviors*. These models control the actions of a vehicle when approaching and crossing intersections.
- *Strategic planner*. This component acts as a coordinator and schedules other behaviours for running, such as intersection behaviours found in the current road edge. In addition, it is responsible for planning strategic lane changes.

Communications. Components which handle the interaction with external frameworks, such as OMNET++, via message passing. Generated messages are efficiently serialized with the Google flatbuffers library [30] and can be sent over the network or saved to a file. Messages are passed every physics update in Unity, every 20 ms by default though it can be configured. Depending on the interaction mode, we consider the following simulation types:

- *Unidirectional*. In this mode, the events in the network simulation cannot alter the events in the traffic simulation. The network simulation basically uses the messages from the traffic simulation to update the positions of vehicles or other entities. Therefore, a network simulation may also be run over a trace file from the traffic simulation. Indeed, the messages can be directly serialized to a file instead of the network, and used to exactly *replay* the traffic simulation with different network configurations. In this case, the simulation time for Veneris can run faster and it just puts the messages on a transfer queue and does not block.
- *Bidirectional*. Sometimes also called bidirectionally coupled, the events in both the network and traffic simulations can alter each other state. OMNET++ also may send messages to Veneris. In this case, both simulations have to run synchronized, and, after sending messages, Unity blocks until a reply from OMNET++ is received.
- *Interactive*. The traffic simulation reacts to external events other than network ones. For instance, when there is a vehicle controlled by a user. This mode can be used with any of the other two. The requirement is to keep the simulation responsive, that is, to keep a framerate high enough to be considered acceptable by the user. For the unidirectional mode, due to our type of simulation, the framerate degrades usually with the load on the CPU, whereas for the bidirectional mode, the framerate is also limited by the performance of the network simulator because of their synchronization.

We next describe briefly some of the more relevant features or novel mechanisms introduced in Veneris. All the implementation details can be checked on the source code provided in our repository.

Rich, interactive 3D environment. Unlike classical traffic simulators, where vehicles are abstract point entities changing their coordinates along lines representing lanes, in Veneris the dynamical elements are 3D rigid bodies subject to physical interactions and constraints, which has a number of implications on the implementation that have been considered. For instance, turns at intersections must have a turning radius large enough to let the vehicle do the maneuver and it has to have a speed slow enough to avoid skidding. In addition, vehicles can react to environment entities and complex behaviors can be seamlessly introduced. In fact, the strength of game engines lie in their ability to recreate rich interactive environments and provide a number of tools, such as mesh colliders and triggers, to facilitate its development. Consider, for instance, a road area with a layer of ice where vehicles can skid: programming such behavior in other tools is remarkably complex, whereas with the Unity engine is just a matter of placing a mesh trigger with the desired shape and programming the action to be executed upon a vehicle enters, stays or leaves the trigger area.

Human in the loop. The possibility of using human-controlled vehicles enables the evaluation of scenarios not usually or easily available for other tools, except for nanoscopic simulations. For instance, for testing the reliability of a CACC to longitudinal and lateral disturbances, a user can perform arbitrary maneuvers, readily executed with a commodity wheel and pedal set, but difficult to program with other tools.

IDM adapted to 3D environment. IDM assumes a 1D con-

figuration where each vehicle follows another one on a line and the acceleration is updated at every time step. We have adapted it to our 3D environment as follows: (1) in our vehicle model the acceleration cannot be directly set, but is the result of the drive torque applied to the wheels. Instead, we use the IDM acceleration function to set the input applied to the acceleration pedal, so strong accelerations/decelerations are mapped to throttle/brake pedal pressure; (2) selecting the leading vehicle in a 3D environment is not as straightforward as choosing the vehicle in front: a driver has to react to the actions of the relevant neighbors. Therefore, the leading vehicle is set to the most relevant (dangerous) neighbor by computing their trajectory intersection, that is, the neighbor that may first collide with the vehicle if speed is kept constant. In addition, to compute the velocity difference, we subtract only the component of the velocity vector of the leader on the unit direction vector of the vehicle.

Cost-based strategic lane change selection. The route generated by the SUMO tools and imported by the vehicle manager only enumerates the connected road segments (edges) that have to be traversed during the trip, but most of the edges are made of several lanes. Therefore, on a strategical level a vehicle has to choose the lane for each edge according to, first, its connectivity to the next segment lanes, and second, some desirable traffic condition, such as the occupancy of the selected lane. The selection algorithm is not trivial and has a remarkable influence on the traffic simulation. Our solution is to use a cost function to select the desired lanes. The cost function depends on some dynamical properties of the lanes, such as the density (vehicles/m), occupancy (number of vehicles), average speed on the lane, etc. Given the cost function, we then select the minimum cost path on the set of potentially usable lanes. To this purpose, a route manager creates a global graph for the scenario where each lane is a node which is linked with a vertex to all the connected lanes in other segments and to all adjacent lanes. On this lane graph, each vehicle runs a A* algorithm constrained on the vehicle route, that is, only nodes (lanes) on the actual route of the vehicle are considered in the search. The procedure works because the route generated by SUMO previously is feasible, that is, there exists actually a path traversing the lanes between the origin and destination. The strategic (mandatory) lane changes are scheduled according to the minimum cost path found and they are later executed using the MOBIL model. This procedure is repeated along the route, to take into account the current traffic conditions in the cost. In our implementation, the strategic planner component keeps a plan for 3 road segments ahead, which is updated every time a new road segment is reached. This cost approach is flexible and allows the combination and testing of multiple alternatives. As an example, we use as cost both the vehicle density as well as the length of the target lane and vehicle speed, to discourage changes to lanes where there is not enough time to stop at the next intersection.

User provided intersection behavior. The behavior of vehicles at intersections is critical for the correct simulation of traffic but there is a wide variety of potential situations to consider. Our approach to cope with this variety is to let the intersection provide its own behavior control. That is, a behaviour is attached to each intersection when the scenario is built, and the strategic planner loads and runs the behavior corresponding to the intersections found in the current road segment. We have implemented basic actions (stop, turn with priority, straight with priority, etc.) and traffic light rules, but users can extend and test Veneris in a seamless way by providing specialized behaviors for particular types of intersections.

Visualization and multi-platform support. The use of a game engine as a simulation platform allows to take advantage of its rendering capabilities for debug and result analysis purposes. A number of visual aids can be added to the simulations to help understand the behaviour of the system and results can be directly rendered over the scene after the simulation is done. Moreover, thanks to the multi-platform support of Unity, simulations can be compiled for different platforms. In particular, with WebGL traffic simulations can directly be run on a web browser, which facilitates the development of interactive scenarios for educational or demonstration purposes. Some examples can be found on the Veneris website.

C. VEHICLE

The vehicle model used in Veneris is a 15-DOF (Degrees of Freedom) model, see Fig. 3, composed of a sprung mass and four unsprung masses for the wheels. The sprung mass comprises the chassis, frame, internal components, passengers and cargo and it has six DOF that define the state of the vehicle body. These include longitudinal, vertical, lateral, yaw, pitch and yaw motion. In addition, each wheel has two DOF, one of them for their vertical motion and the other for the wheel spin. Finally, the steering system has another DOF.

The implementation of the vehicle model follows a modular approach where each of the main physical parts of a vehicle (e.g. brakes, steering system, etc.) is associated with a configurable software component. In this way, the setup of a vehicle can be easily changed and its components can be replaced by others using different models. Besides, components can be extended with new functionality, since the source code of components is available. Fig. 4 shows the main components of the vehicle model, which model the operation of typical real car parts [32]. In addition to the components depicted in the figure, a vehicle controller assembles the components and configures the vehicle in runtime. Another component provides the inputs to the vehicle (pedals, gear shift and steering wheel) from different devices, such as the keyboard or an automated controller.

The powertrain component generates the drive torque that will be delivered to the powered wheels. Inside the powertrain, the engine component calculates the torque it delivers using a torque curve that takes the engine angular speed and the throttle as input parameters. A clutch transfers this torque to the drivetrain component, which couples the engine to the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

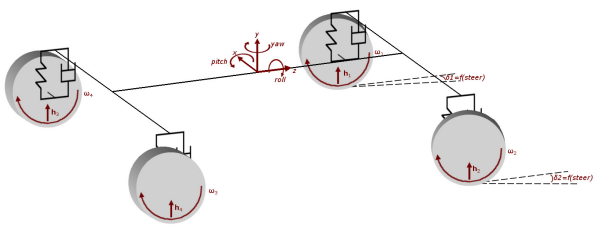


FIGURE 3: 15-DOF vehicle model. The rotation of the front wheels is a function of the steering wheel rotation calculated according to an Ackermann steering geometry.

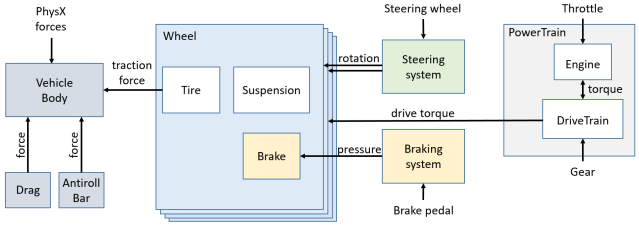


FIGURE 4: Main components of the vehicle model.

drive axle with the powered wheels, converting torque and rotational speeds according to the selected gear. Brakes have been modeled using a hydraulic braking system. The steering system component uses an Ackermann steering geometry. The suspension for each wheel is modeled as a mass-spring-damper system, as was shown in Fig. 3. Each axis of the vehicle has an anti-roll bar to reduce the amount of roll during cornering. The tire model plays a major role in realistic simulation of vehicle dynamics. It determines the wheel traction forces. Experimental results show that the tire force generated by each tire depends on the slip of the tire relative to the road, the normal load on the tire and the friction coefficient of the tire-road interface [33]. In Veneris, an existing component from the NVIDIA PhysX [29] physics engine has been used to simulate the wheels, including the suspension. This tire model uses predefined longitudinal and lateral traction force curves based on wheel slip, tire load and friction. It offers satisfactory accuracy if these curves are adjusted to match the operational conditions of the tire. Finally, the vehicle body is modeled as a rigid body with adjustable center of mass and moment of inertia. The rest of components exert forces on the rigid boy that include traction, suspension and aerodynamic drag forces. These forces are used to calculate linear and angular accelerations along the axes corresponding to the six DOF of the vehicle body and determine the translation and rotation of the vehicle.

D. OPAL

Opal is a 3D ray-launching based, deterministic radio-frequency propagation simulator. With *Ray-launching* methods, also called shooting and bouncing (SBR), [16] electromagnetic waves are simulated by rays launched from the transmitter with a predetermined angular spacing (AS). These rays are propagated along their trajectory until they hit an obstacle where they are reflected, diffracted, transmitted or scattered. Subsequent rays are traced again. The contributions of the different rays that hit a reception sphere on the receiver are added to compute the electric field. The assumed space discretization limits the accuracy of the method compared to methods such as *Ray-tracing*, also called image method [16].

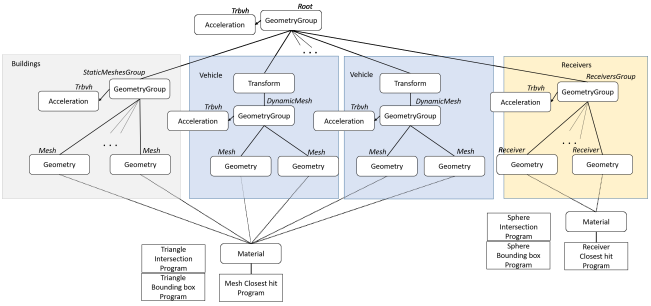


FIGURE 5: Opal scene graph.

Opal has been implemented in C++ with NVIDIA OptiX [27], a general-purpose ray tracing engine for rendering, designed for GPU and other highly parallel architectures. The programming model of OptiX involves a host-based API used to define data structures for ray tracing and a device-based CUDA API, used to generate rays, intersect them with surfaces and process those intersections, by calling user-defined programs. Opal works with both static and moving 3D scene objects, represented as triangle meshes, and multiple transmitter and receivers. Objects, transmitters and receivers can be dynamically added and removed from the scene and subsequent ray launches take into account those changes.

With OptiX a scene is represented as a graph that controls the traversals of rays through the scene [27]. Nodes usually describe geometric objects, transforms and other data objects, such as acceleration structures, based on a bounding volume hierarchy (BVH) [16], that improve the operations on sets of geometrical objects. The structure of the scene created by Opal is shown in Fig. 5. All the static objects in the scene, such as buildings, share an acceleration structure below the root node and are assigned an intersection program, which computes ray-triangle (mesh) intersection, and one or more materials and a closest hit program. The receivers, represented as spheres, are attached just below the root node and also share an acceleration structure. When they move, their sphere coordinates are updated. Finally, the dynamical geometrical objects, such as moving vehicles, which may be made of several meshes, are assigned each one a transform node, updated when the object moves, and their own acceleration structure.

A user program associated to the closest hit for meshes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

performs the operations required when a ray hits the geometry, such as reflections, diffractions and so on. In our case, only reflections are implemented currently, so it computes the reflected ray, the reflection coefficient and updates the ray payload. The closest hit program for receivers filter duplicates and adds the ray contributions to the electric field.

Every mesh added to the scene has its own electrical properties (relative permittivity and conductivity), which are taken into account in the computations. In addition, several parameters, such as the AS, the maximum number of reflections, the individual radius of the receiver spheres and so on, can be configured.

Novel online duplicate ray removal. Multiple counting of received rays is a major problem of the ray-launching approach [31]. Removal of duplicates usually requires setting a receiver sphere radius dependent on the ray unfolded distance and keeping a log of the ray paths to track the sequence of hits and filter them in a post-processing phase, which is not efficient in a parallel environment. We use a novel approach: each ray stores in a *single* integer the sequence of environment elements (mesh faces) it hits, by using a *combined hash* of the face identifiers. Each time a face is hit, the combined hash is updated. Since this hash is guaranteed to be unique for different sequences of faces, including permutations, it is used to remove all the rays following the same sequence except for the closest one to the receiver. Therefore, all hits on a receiver, with hash and distance information, are stored and, after tracing, are filtered directly on the GPU with the Thrust parallel library [35]. It allows us to keep all the computations on the GPU to leverage parallel computing.

Moving objects. Propagation simulation by ray launching has usually been used for characterization of static environments, where the transmitter is kept fixed at a few positions [16], mainly because of a lack of tools for proper representation of dynamical 3D environments and low performance of the tracing process. On the contrary, ray tracing engines such as OptiX have been primarily designed to obtain high performance and quality rendering for interactive video games and, as such, they are optimized for real-time changing scenes. By leveraging them, Opal allows to dynamically add transceivers and other objects and can actually bring real-time changes to propagation simulators with acceptable accuracy and performance. Even though GPU-based propagation simulation has been described in several prior works [16], only authors of [18] evaluate the performance of ray launching with moving vehicles.

E. OMNET++ MODULES

A set of OMNET++ modules have been developed to provide bidirectional coupling between the traffic and network simulator. First, a generic execution server replaces the default OMNET++ scheduler with an external time scheduler which advances simulation time and execute events only when synchronization messages from the external simulator are received. Every message coming from Veneris includes the time as run by the external simulator in the header, so the simulation advances at each received message. Second, a Veneris Server derives from the generic execution server and processes all the currently defined types of messages from Veneris, creating and inserting new vehicle modules in the simulation and updating the state of the active modules. In addition, an Opal radio medium extends the INET framework [7] by enabling Opal ray launching based propagation simulation. In fact, Opal can be used as radio medium for a INET simulation independently of Veneris. The Veneris server is mainly used to build an Opal scene from a Unity scenario and to update the state of vehicles in a hybrid simulation.

IV. VALIDATION

In this section we describe and discuss the validation tests we have performed to the Veneris framework. There are three key areas that need to be tested: the fidelity of the vehicle dynamics, the recreation of realistic traffic flows and the accuracy of the propagation simulation. In addition, general results of the expected performance are provided, although a thorough performance evaluation and comparison is left for a future work.

A. VEHICLE VALIDATION

Two different tests have been performed in order to demonstrate that the vehicle model is able to reproduce the behavior of actual vehicles with enough fidelity. One of the tests is aimed at validating longitudinal vehicle dynamics, while the other deals with lateral dynamics. The VeDYNA vehicle simulation software [34] has been used to compare the results in both cases, replicating an existing test in VeDYNA into Veneris. The same vehicle configuration has been used in the two simulation environments for the equivalent parameters. More specialized parameters only present in VeDYNA have been left to their default values. The validation tests are discussed in the following paragraphs. The first test is an acceleration/deceleration test. At the beginning of the test, the vehicle is in neutral gear and the accelerator pedal is progressively pressed. After one second, when the pedal is fully pressed, the first gear is engaged. As the car accelerates, gears are shifted up just before the engine reaches its maximum revolutions per second. After thirty five seconds, the accelerator pedal is released and the brake pedal is pressed, reaching full pressure after one second and maintaining the pedal pressed afterwards.

Figure 6 shows the longitudinal speed of the vehicle in both simulation environments. It can be seen how the results are very similar during the acceleration phase and nearly identical during the braking phase with a Mean Absolute Error (MAE), $MAE = \frac{1}{N}\sum_1^N |x_i - y_i| = 0.5594$ m/s. The acceleration differences come from the use of different integration steps. Veneris, by default, uses 0.02 s for the integration step, which is not small enough to handle situations with large torque transfers from the engine to the drivetrain. This fact forces a reduction of the maximum torque that can be transferred between them in order to avoid oscillations in the rotational speed of the engine. In the test, this happens

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

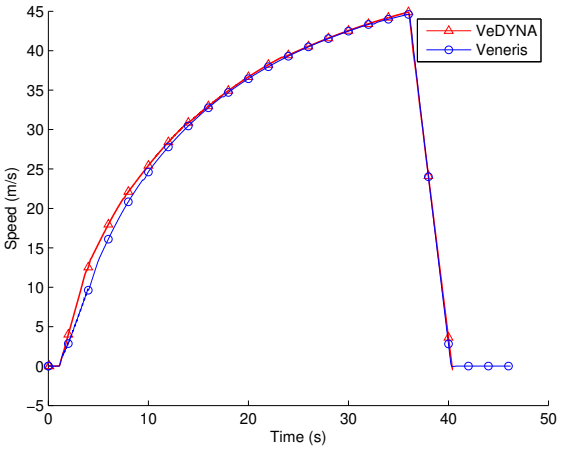


FIGURE 6: Acceleration-Deceleration profile

after the first gear is engaged, when the engine is rotating at its absolute maximum speed while the wheels are motionless. However, in typical traffic scenarios this should not happen, as the driver behavior will be smoother. Decreasing the integration time step would alleviate this issue, but it incurs in high performance costs for only small accuracy gains.

The second test, the steering wheel step test, analyzes the vehicle lateral response after a step input at the steering wheel. After starting the test, a proportional, integral, derivative (PID) controller is used to reach and maintain a target speed of 40 km/h. When the speed is stabilized, the steering wheel starts turning at a rate of 300 degrees per second until it reaches a final rotation of 100 degrees. Then, the steering wheel is kept in this position. The execution of this test required to make some minor adjustments to the car setup. It was hard to match the behavior of the tires and the wheel suspensions as they are defined by different parameters in both simulation environments. These two components are part of the PhysX component for the wheels and, therefore, have not been modified to avoid this situation. For the tires, the curves used to obtain tire forces were fine-tuned empirically. As for suspensions, the roll center had to be modified to avoid oscillations with large integration steps.

Figure 7 shows the results of the steering wheel step test together with the Mean Absolute Error. It can be seen that the trajectories followed in Veneris and VeDYNA, although not identical, are quite similar with a MAE=0.0834 m. The Veneris vehicle has also a slightly higher yaw angle than the VeDYNA. These differences are mainly due to the use of different tire models and the difficulty of matching them. On the other hand, the roll of the Veneris vehicle depicted in the figure is smaller than the one measured with VeDYNA. The reason for this is the aforementioned displacement in the roll center of the wheels that had to be made because of oscillations. This fact reduced the roll of the vehicle but it did not affect the trajectory followed by the vehicle. In general, it can be observed that our base vehicle has a realistic behavior. Both longitudinal and lateral vehicle responses match well those of VeDYNA. There are only minor differences between them, which are explained by the use of a simpler model and a larger integration time step, which is justified because it results in a better performance.

B. TRAFFIC SIMULATION VALIDATION

The validation of the traffic-related behaviour of Veneris is specially important, since users have to be confident that Veneris provides an accurate recreation of real traffic. A first approach would be to directly compare the Veneris results with the SUMO results. Since SUMO is a well tested and validated traffic simulator, if Veneris is able to reproduce the results of SUMO, users should be confident about the validity of Veneris. However, Veneris incorporate a more realistic physical behaviour of the vehicle, and so this difference should show up in the results, which may raise doubts about whether the simulator is flawed or the results are correct and the disagreement comes from the models themselves. So, we resort to an additional experimental data source as follows.

Since the validation of synthetic mobility traces is difficult, as discussed in the Section II, mainly due the lack of available real traces, we have followed the methodology proposed in [23] based on publicly available data. We use the navigation and directions service provided by Google [24]. The results provided by such services are based on a number of real sources and return accurate values that can be retrieved with dedicated APIs. In particular, the Google API returns different travel times according to the departure time and some predefined traffic models: *optimistic*, *pessimistic* and *bestguess*, which indicates that it takes into account the potential congestion for different periods of the day.

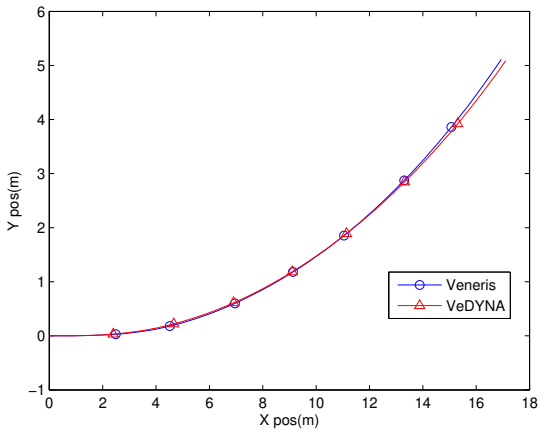
Since the results obtained from these sources are given by a typical situation in the urban area we are interested, it is clear that the scenario to be compared against them needs to be of a relatively large size and with a realistic traffic demand, in order to match the real conditions. This allows also to test the quality of the traffic demands generated for some scenarios, as we discuss later. However, as pointed out in [21] there is a lack of freely-available and properly working scenarios for SUMO as well.

In the following, we discuss first our methodology for the validation and afterwards the results obtained.

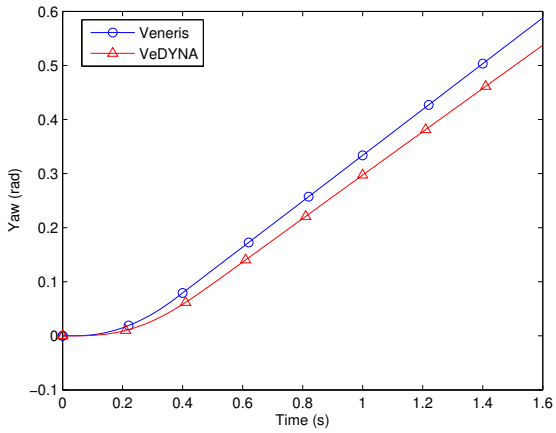
1) Selection of the scenario and traffic demand

Some real mobility traces of highway scenarios are publicly available, see [22] for a recent discussion. But, since Veneris is not suitable at the moment for highway simulation due to its lack of discretionary lane changing model, we focus on urban scenarios. Very few scenarios are available and most of them have problems [21]. Only four of them are linked in the SUMO website. The Luxembourg (LuST) scenario [21] is the most complete and well-developed one. However, with more than 5600 edges and more than 70000 vehicles over 24 hours, it cannot be simulated as a whole with Veneris. It must be remarked here that the network size can perfectly

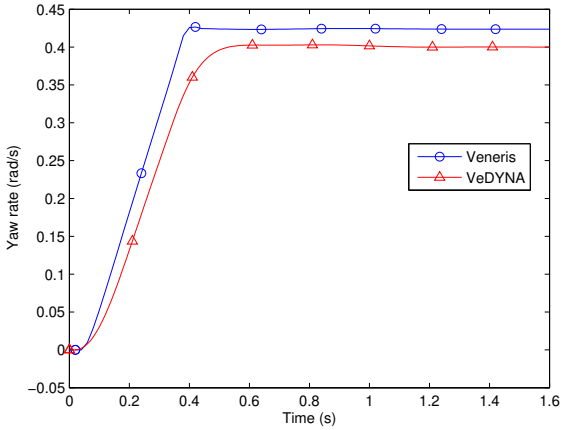
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access



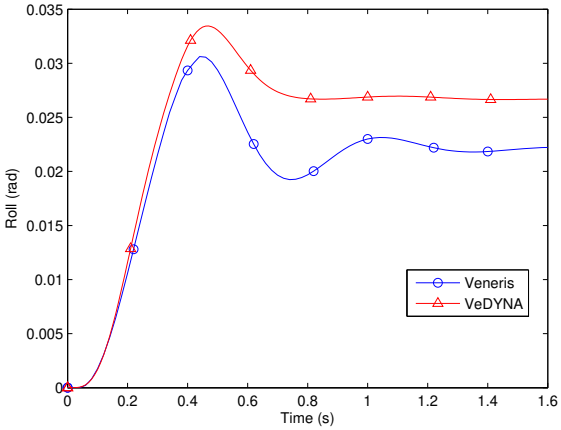
(a) Vehicle trajectory comparison. MAE=0.0834 m



(b) Vehicle yaw comparison. MAE=0.0293 rad.



(c) Vehicle yaw rate comparison. MAE=0.0292 rad/s.



(d) Vehicle roll comparison. MAE=0.0039 rad.

FIGURE 7: Steering wheel step tests results.

be handled by Veneris, but the traffic demand is too large, that is, the number of *simultaneously* running vehicles can rise up to 4000. The Bologna Ringway dataset described in [23] is also available, but as warned in the SUMO website, it is not working properly: it has unsafe traffic lights which results in collisions. Another large scale scenario is the TAPAS/Cologne. Nevertheless, as also warned in the SUMO website, the scenario is hardly usable due to several problems.

So, we finally resorted to the Bologna scenario [36]. The scenario was built as part of the iTETRIS project [37]. This package provides two scenarios, the Andrea Costa and the Pasubio areas which can be joined together in a larger area. We have used the Pasubio area, which has a reasonable size, with 111 edges, 65 junctions and 16 traffic lights, and more importantly, the size of the traffic demand is suitable: just one hour at the morning peak hour, from 8:00 to 9:00, with 8680 trips. A snapshot of the simulated scenario is shown in Fig. 8.

FIGURE 8: The Pasubio scenario from Bologna. Snapshot from Veneris.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

SUMO is able to simulate the complete traffic demand and its results show only moderate congestion. On the contrary, with Veneris, the simulation of the 100% of the traffic demand (8680 trips) results in a highly congested scenario which in turn heavily degrades the performance of the simulator. Therefore, we reduced the traffic demand to 30% (2893 trips) and 50% (4340 trips) of the original demand by uniformly removing trips, which improves both the results and the performance, as we discuss later in this section.

2) Navigation services

Following the approach in [23], we have retrieved trip duration from available navigation services. We have used the Google Maps Directions API [24]. We can ask for the route from an origin to a destination and the service returns the recommended route, together with additional information, in particular, the route length and the duration. Intermediate waypoints can be provided as part of the query and the service returns a route passing by these intermediate points. The Google Maps API also allows to indicate the departure time and a traffic model among *optimistic*, *pessimistic* and *bestguess*. According to the description of the service, specifying the departure time allows to receive a trip duration that takes the traffic conditions into account. Moreover, live traffic becomes more important the closer is the departure time to the actual time.

We have used the 50% traffic demand of the Bologna scenario and all the trips have been converted to Google Maps queries, generating a total of 4189 replies. The routes gathered are equal in most of the cases to the routes used by the vehicles in the simulator. In a few cases, the routes generated are slightly different from the ones used in the simulator, either because some minor roads are not modelled in the simulator or due to differences in the map information with respect to the converted network. We ask for the *pessimistic* and *bestguess* traffic models and the departure time has been set to 8:30 hours, but queried in the evening to avoid results too close to a particular day live traffic.

3) Results

The durations of the collected trips for the simulations and the navigation services, for the 50% and 30% of traffic demand, are shown in Fig. 9. As can be seen, with a 50% traffic demand, that is, aproximately 4000 vehicles/hour, Veneris shows signs of traffic congestion, which is neglected in the results of the *bestguess* model of Google Maps, while SUMO does not show congestion and the durations are clearly shorter than those provided by Google. However, the results for the *pessimistic* Google model match better the ones provided by Veneris, though the latter still shows more dispersion due to congestion. In fact, when we compare sample to sample the trip durations with the MAE, summarized in Table 1, we see that the MAE for Veneris with respect to Google Maps pessimistic is clearly lower than the one for SUMO, showing better coincidence.

TABLE 1: Mean Absolute Error (MAE) in m/s between Google Maps bestguess (G-bg) and pessimistic (G-pes), SUMO (S) and Veneris (V)

Traffic demand	V-G bg	S-G bg	V-G pes	S-G pes
50%	2.194	2.673	2.13	5.37
30%	1.62	2.80	3.53	5.42

TABLE 2: Cramér-von Mises Goodness of Fit test (p-value) for Google Maps bestguess (G-bg) and pessimistic (G-pes), SUMO (S) and Veneris (V)

Traffic demand	V-G bg	S-G bg	V-G pes	S-G pes
50%	0.0005	0.081	0	0
30%	0.61	0.12	0	0

TABLE 3: Veneris performance for the Pasubio Scenario

Metric	50% Traffic demand	30% Traffic demand
av. physics rate	28.75 cps	49.95 cps
av. frame rate	15.04 fps	57.87 fps
max active vehicles	600	227
av. active vehicles	317.07	156.67
real/simulated time	1.74	1.001

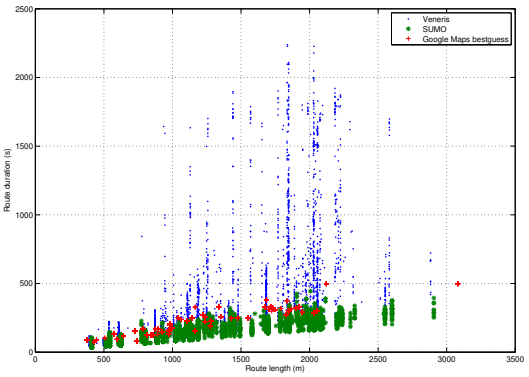
If we look at the results for a lower traffic demand, 30% of the original one, we see that the results of Veneris agree with Google Maps bestguess model closely, even better than SUMO in fact. Both, the dispersion plot in Fig. 9c and the MAE for Veneris with respect to Google Maps bestguess are in better agreement than SUMO. On the contrary, neither Veneris nor SUMO replicate the results of Google Maps pessimistic in this scenario.

To further confirm this results, we have performed goodness of fit tests to the datasets. We tested the datasets *in pairs*, that is, we have computed the Cramér von Mises statistic [38] to test the null hypothesis that the pair of datasets come from the same distribution, with a cutoff value of 0.05. The results of the Cramér-Von Mises test for the datasets are summarized in table 2.

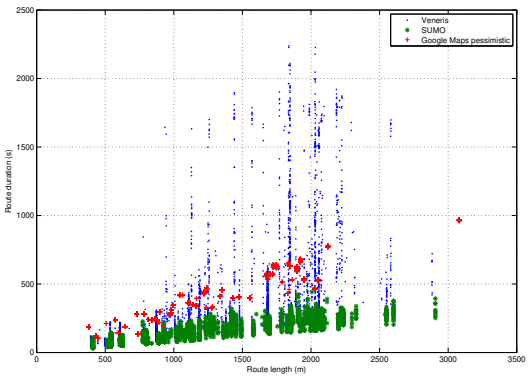
We see that the null hypothesis is not rejected for SUMO and Google Maps bestguess for the 50% traffic demand and also for the 30% demand, whereas for Veneris it is not rejected only for the 30% traffic demand. However, in this latter case, the larger p-value supports a better match between Veneris and Google Maps bestguess than SUMO. For the rest of the cases the null hypothesis is rejected.

Finally, since this a relative large scenario, in terms of traffic demands, we discuss the performance of the simulator. Veneris performance metrics are shown in Table 3. In Unity the physics simulation calls and the rendering calls are separated. The simulated time advances in fixed steps, every time the physics engine is called. That is why we have separated the physics and the rendering (frame) rates in the Table. Most of the simulator logic is implemented during the physics calls, since we mainly use the physics library for the vehicle implementation. From the point of view of the

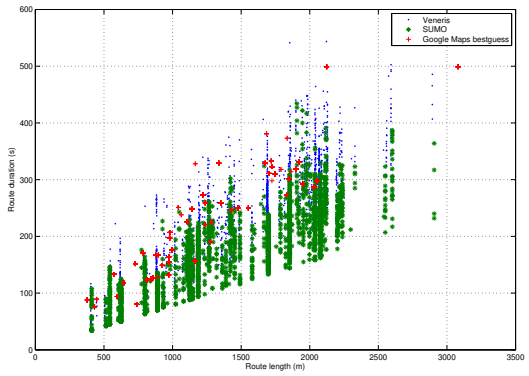
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access



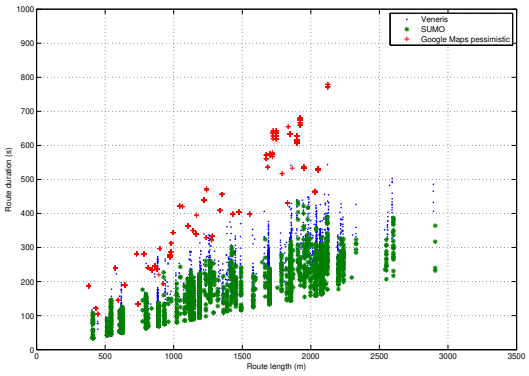
(a) Trip duration versus route length for 50% of traffic demand. Veneris, SUMO and Google Maps with bestguess model



(b) Trip duration versus route length for 50% of traffic demand. Veneris, SUMO and Google Maps with pessimistic model



(c) Trip duration versus route length for 30% of traffic demand. Veneris, SUMO and Google Maps with bestguess model



(d) Trip duration versus route length for 30% of traffic demand. Veneris, SUMO and Google Maps with pessimistic model

FIGURE 9: Validation of traffic model. Duration versus route length for the simulations (Veneris and SUMO) and the navigations services (Google Maps Directions API)

simulation results, the frame rate is irrelevant. The frame rate becomes important when the simulation is interactive, when a human player is controlling some vehicle or providing some user input. We have set the physics time step to 0.02 s which, since Unity tries to advance the simulation time at a real-time rate, means that the physics simulation is run 50 times per second (calls per second, cps). The frame rate goal is set to 60 frames per second (fps) by default. These goals are met for the 30% traffic demand scenario, with almost 50 cps and 60 fps, which means it can be simulated without problems in a fully interactive way. As shown, Veneris can handle 227 simultaneously running vehicles (max active vehicles row) without noticeable degradation of performance for the user. On the contrary, the 50% traffic demand scenario could not be simulated interactively due to the degradation of performance. A drop to 15.04 fps on average is perceived by a user as unresponsive and unacceptably slow simulation. But, let us remark, that for a non-interactive simulation, the performance is perfectly reasonable, In this case a one-hour simulated time only requires 45 additional minutes to complete.

Let us summarize the results:

- The speed profiles for both SUMO and Veneris approximate well the real world ones according to Google Maps models.
- Veneris shows clear congestion for the 50% traffic demand, whereas for the 30% traffic demand the results match the Google bestguess model better than SUMO. This indicates that (1) the typical demand as computed from the Google historical data is actually closer to the 30% rather than the full traffic demand of the scenario and (2) that Veneris is able to provide as good as or even more accurate results than SUMO. On the contrary, SUMO tends to underestimate the duration of the trips for all the traffic demands of the scenario.
- Veneris can be run in a fully interactive way for typical real-world demands of the Pasubio scenario, the 30% and 50% traffic demands, even though it cannot cope with the (unrealistic) full demand.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

TABLE 4: Configuration parameters for the Chicago, two-ray (TR) and street-crossing (SC) scenario.

	Chicago	TR-SC
Ray AS (degrees)	1	1/0.1
Max. number of reflections	10	2-5
Frequency (GHz)	5.875	5.9
Receiver sphere radius (m)	1	1/5
Relative permitivity (η)	3.75 (building), 1.02 (floor), 6 (vehicles)	3.75
Relative conductivity (σ)	0	0.038
Nakagami parameters	m=0.75, path loss= 2.26	–
GEMV^2 parameters	NLOSb min. σ= 0 dB	–
Polarization	Vertical	
Power	1 W	

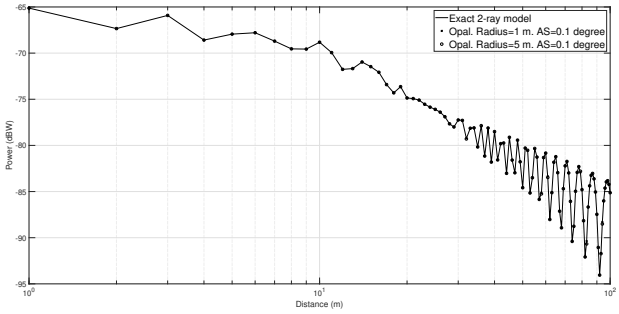


FIGURE 10: Validation of two-ray model. Exact vs Opal received power with 1 and 5 m of reception radius and 0.1 degree of AS.

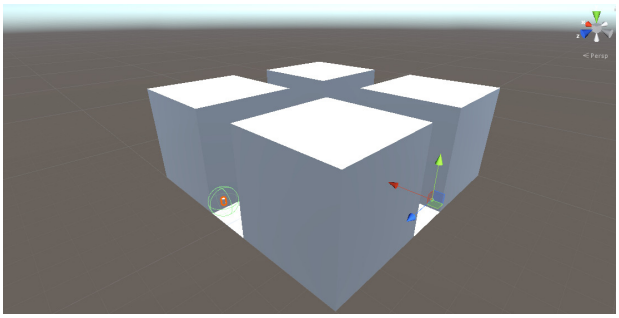


FIGURE 11: The street crossing scenario. Transmitter (arrows) and receiver (sphere) are at Y=10 m, building heigth is 40 m. Transmitter moves along the X axis from -50 to 50 m.

C. PROPAGATION VALIDATION

In this section we validate the simulation of propagation with Opal. The configuration parameters used, unless other are explicitly mentioned in the text, are shown in Table 4. We use a ray sphere with both 1 degree and 0.1 degree of Angular Spacing (AS), which result in 65160 and 6483600 rays traced respectively.

1) Stand-alone validation

As a first step, Opal has been validated as a stand-alone application, that is, Veneris is not used, and meshes are loaded directly from Unity. A trivial test with a transmitter and a receiver in free space has been performed, resulting in perfect agreement with free space propagation. The first non-trivial test involves a receiver and a transmitter over an infinite plane, that is, a two-ray model, at different distances. The height of transmitter is 10 m and receiver is 2 m and the plane has been assigned brick material properties, according to [39]. In this case, only a direct ray and one reflection should hit the receiver. The results, compared with the exact two-ray model computed with Matlab, are shown in Fig 10. As can be seen, perfect matching with the theoretical model is achieved with an AS of 0.1 degrees (MAE $< 10^{-4}$ dB). More importantly, this scenario tests our online duplicate filtering algorithm: it correctly discards all the duplicate reflections and keep the closest one. Let us remark that, with a AS=0.1 degrees, for instance, at 35 m, more than 4500 reflections hit the receiver.

The second scenario is a street crossing, modeled with a plane and four 40x40x40 m cubes, representing buildings, as shown in Fig. 11. The receiver is placed at coordinates (0,10,100)^2, whereas the transmitter starts at (-50,10,50) and moves 100 m along the X axis until the point (50,10,50), transmitting every 1 m. Again the material chosen for plane and buildings is brick. To validate this scenario we have compared the Opal results with the results provided by a 3D Ray-tracing program implemented in Matlab [40]. It is based on image theory and estimates the reflected, diffracted components and combinations thereof; it is also able to compute single and double diffuse components. The environment geometry is implemented with libraries specifically programmed for the tool. The number of images computed in a image-based tracer scales exponentially with the number of reflections and faces in the environment. The execution time of the Matlab tracer for 2 to 5 reflexions is 31, 59.4, 490.7 and 16227 s, respectively. For comparison, with Opal the total time to run the tests took around 2.7 s for AS=0.1 degrees and around 0.4 s for AS=1 degree, independently of the number of reflections. The results shown in Fig. 12 again confirm that AS=0.1 degrees matches the Matlab results, independently of the reception sphere radius: MAE=0.219 dB with radius=1 m and 5 reflections. There are differences, especially in non line-of-sight (LoS) zones, for AS=1 degree due to undersampling, but not large: MAE= 3.75 dB with radius=5 m and 5 reflections. We also show the results for 4 reflections to remark that beyond this number of reflections there is little difference in the results. The MAE for the other combinations of number of reflections and radius is lower than the previously mentioned one.

2) Hybrid simulation

For this scenario we have simulated a full unidirectional simulation in a urban scenario. We have used an area from

[2]Unity uses a left-handed coordinate system, with height usually represented by the Y axis.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access

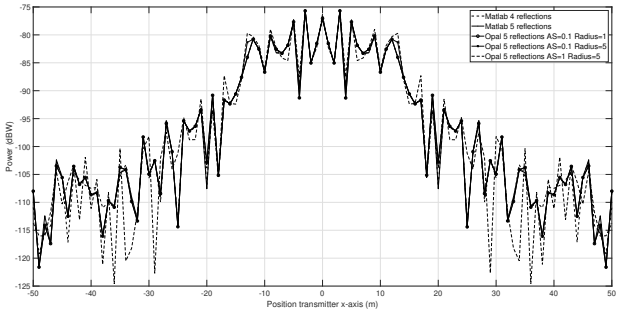


FIGURE 12: Validation of street crossing. Received power for Matlab ray tracer and Opal with 1 and 5 m of reception radius and 1 and 0.1 degrees of AS.

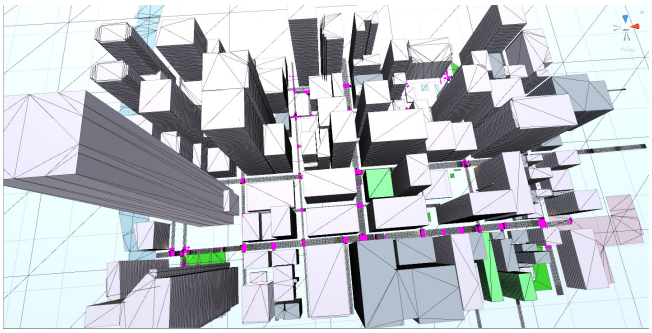


FIGURE 13: The Chicago downtown scenario. It has 235 roads, 21 traffic lights, 129 intersections and 163 buildings.

Chicago downtown, because it is a rich multipath environment, due to its grid road network surrounded by tall buildings, a picture is shown in Fig. 13. The validation of propagation mechanisms implemented in Opal, basically reflections, has been done in the previous subsections. In this section we are interested in higher-layer metrics and performance of a full unidirectional simulation with a protocol stack against a hybrid propagation model. We have configured OMNET++/INET to simulate 802.11p network interfaces at 5.875 GHZ, which is the carrier frequency of the first channel in the 5.9 GHz DSRC band, with a data rate of 6 Mbps and a periodic beacon generator on top of it. We use 500 bytes beacons at 10 beacons/s for all vehicles. We have generated a traffic demand that inserts 157 vehicles, one every 1.5 s, from the borders of the scenario. The simulation runs for 738 s of simulated time, until all vehicles have finished their routes and left the simulation. The simulations have been replicated 5 times with different seeds. We run Veneris with Opal propagation versus a usual stochastic model, Nakagami-m, and an accurate hybrid model, the implementation of GEMV^2 [17] for OMNET++ from the Artery framework [41]. The parameters of the model are shown in Table 4. The simulations have been run on a commodity computer with an intel i5-6400 CPU, 16 GB of memory and a NVIDIA GeForce GTX 970 GPU.

In Fig. 14 we show two relevant metrics for vehicular

TABLE 5: Average run time (s) for the Chicago scenario

Veneris-Opal	GEMV^2	GEMV^2 ND	Nakagami-m
4711.0 $\pm$ 11.68	16396.1 $\pm$ 191.5	9355.4 $\pm$ 10.28	608.7 $\pm$ 14.5

applications with their 95% confidence intervals. The average inter-beacon reception time (IRT) and the average number of neighbors are crucial for congestion control protocols [42] and used as an indirect measure of the channel quality. As can be seen, the use of a stochastic model such as Nakagami-m, oblivious to the environment geometry, results in a large overestimation of these two metrics. To remark the importance of using more accurate values for these metrics in the design of higher layer protocols consider that congestion control limits the beaconing rate, which in turn, decreases the quality of the cooperative awareness basic service, on top of which multiple CAD applications are built. On the contrary, with geometry-aware models, such as GEMV^2 and Opal, the blocking effects of buildings are properly accounted for. Even though GEMV^2 incorporates propagations mechanisms not implemented by Opal, such as diffraction, the results are much closer: the MAE for the average IRT between GEMV^2 and Opal is 0.039 s and the MAE for the average number of neighbors is 5.5032. Since GEMV^2 has been validated in several works [17], [18], our results also validate Opal in its current implementation. We left a more thorough validation as a future work when we have added diffraction to Opal.

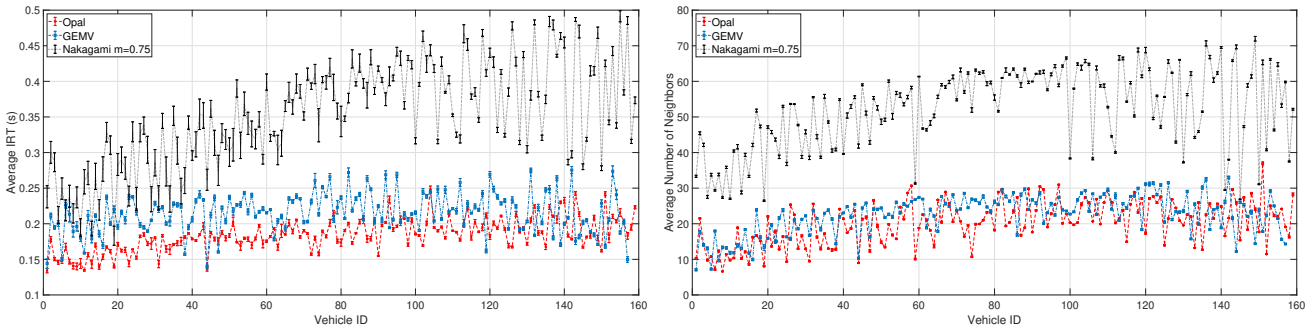
Regarding the performance, Veneris is able to simulate the scenario in real time at 50 fps, as should be expected since the number of active vehicles, with a maximum of 104 and a time average of 42.39, is lower than in the Pasubio scenario in Sect. IV-B. As a unidirectional hybrid simulation, the messages are serialized to a file and later fed to the Veneris server in OMNET++. The average time to run the simulations is shown in Table 5 together with the 95% confidence interval. As can be seen, the computation of geometry-based propagation results in an increase of computational cost, whereas for a stochastic model the cost is not significant and may be used in an interactive real-time simulation. Even though GEMV^2 is a simplified model, the average simulation duration is 3.48 times longer than the Opal one. To provide a fairer performance comparison with Opal, which does not simulate diffraction, we have also simulated GEMV^2 without vehicle diffraction (GEMV^2 ND): vehicle obstacles are simulated as NLOSb [17]. In this case, GEMV^2 simulations still take 1.9 times longer than Opal ones. Overall, the results show that ray-tracing propagation on GPU is a feasible and practical approach for medium size scenarios with promising potential.

D. CURRENT LIMITATIONS

Finally, we discuss here the current limitations of the Veneris framework. Most of them can be overcome in future releases as new functionality is added.

- Urban simulation. Although Veneris can be used for

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2908651, IEEE Access



(a) Comparison of average IRT between GEMV^2, Veneris with Opal and Veneris with Nakagami-m propagation.

(b) Comparison of average number of neighbors between GEMV^2, Veneris with Opal and Veneris with Nakagami-m propagation.

FIGURE 14: Comparison of higher-layer results for unidirectional simulation with different propagation models.

simulating a highway, it lacks a *discretionary* lane changing model, that is, a model where the driver motivation to change lane is based on a perceived improvement of the driving conditions. Such a model should have a relevant influence on the simulation of high capacity roads.

- Medium size scenarios. Veneris can handle interactive traffic-only simulations with up to around 150 active vehicles in real time. For hybrid simulations, interactivity has to be given up for such a number of vehicles due to the network simulation performance cost. But they can be run in a reasonable time even with Opal propagation. Moreover, interactive simulations using just OMNET++ stochastic propagation models can be run for scenarios involving a hundred vehicles.
- No diffraction. Only reflections are simulated currently with Opal. At least a simplified diffraction modeling should be added and has been left as a priority addition for future releases.
- Multiple parallel transmissions only for electromagnetic characterization. Opal can simulate simultaneous transmissions in parallel. For instance, it may simulate a few base stations and several hundreds of receiver locations with a single launch. However, when used with OMNET++, only single-transmitter/multiple-receiver transmissions are computed due to the difficulty of integrating parallel transmissions with INET.

V. CONCLUSION

We have described and validated Veneris and Opal, a novel hybrid traffic-network simulator implemented with a game engine and a GPU ray tracer. Our tests show that it provides realistic results in key aspects, such as vehicle and traffic dynamics and propagation, that match specialized tools.

The more relevant features of the framework have been briefly described and discussed in relation to the related work. In particular, we have introduced a novel duplicate ray filtering algorithm for ray-launching methods and adapted the IDM and MOBIL models to a 3D environment as well as developed a cost-based lane change selection mechanism.

In future works we plan to elaborate on these novel features providing a detailed evaluation of the influence of physically accurate models in traffic-related results. We are currently implementing diffraction for Opal and intend to validate its results with measurements. A more thorough evaluation of the framework performance is planned in order to optimize its operation as well as to test its potential for interactive simulations.

REFERENCES

[1] J. Guanetti, Y. Kim and F. Borrelli,"Control of connected and automated vehicles: State of the art and future challenges," Annual Reviews in Control, vol.45, pp. 18–40, 2018.

[2] K. C. Dey *et al.*, "A Review of Communication, Driver Characteristics, and Controls Aspects of Cooperative Adaptive Cruise Control (CACC)," IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 2, pp. 491–509, 2016.

[3] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," IEEE Transactions on Intelligent Transportation Systems, vol. 4, no. 3, pp. 143–153, 2003.

[4] K. von Neumann-Cosel, M. Dupuis, and C. Weiss, "Virtual test drive provision of a consistent tool-set for [d,h,s,v]-in-the-loop," Proceedings of the Driving Simulation Conference, Monaco, 2009

[5] M. Schiller, M. Dupius, D. Krajzewicz, A. Kern, A. Knoll, "Multi-resolution Traffic Simulation for Large-Scale High-Fidelity Evaluation of VANET Applications," in Simulating Urban Traffic Scenarios,Behrisch M., Weber M. (eds), Lecture Notes in Mobility, Springer, Cham, 2019.

[6] D. Krajzewicz, J. Erdmann, M. Behrisch and L. Bieker, "Recent Development and Applications of SUMO – Simulation of Urban MObility," International Journal On Advances in Systems and Measurements, 5 (3,4), pp. 128–138, 2012.

[7] INET, Open-source OMNET++ module suite for wired and wireless mobile networks. [Online]. Available: https://inet.omnetpp.org/

[8] C. Sommer, J. Härri, F. Hrizi, B. Schünemann and F. Dressler, "Simulation Tools and Techniques for Vehicular Communications and Applications," in , Vehicular ad hoc Networks, C. Campolo and A. Molinaro and R. Scopigno (eds), pp. 365–392, Springer International Publishing, 2015.

[9] I. Llatser *et al.*, "Simulation of cooperative automated driving by bidirectional coupling of vehicle and network simulators," IEEE Intelligent Vehicles Symposium (IV) 2017, pp. 1881–1886, 2017.

[10] M. Segata *et al.*, "Plexe: A platooning extension for Veins," IEEE Vehicular Networking Conference (VNC) 2014, pp. 53–60, 2014.

[11] C. Garcia-Costa, E. Egea-Lopez, J. Garcia-Haro, "A stochastic model for design and evaluation of chain collision avoidance applications," Transportation Research Part C: Emerging Technologies, vol. 30, pp. 126–142, 2013.

[12] T. Erez, Y. Tassa and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," International Conference on Robotics and Automation, 2015.

[13] M. Lewis and J. Jacobson, "Game engines," Communications of the ACM, 45(1), pp. 27–31, 2002.
[14] Unity Engine. [Online]. Available: https://unity3d.com
[15] S. Santini *et al.*, "A consensus-based approach for platooning with inter-vehicular communications and its validation in realistic scenarios," IEEE Transactions on Vehicular Technology vol. 66(3), pp. 1985–1999, 2017.
[16] Z. Yun and M. F. Iskander, "Ray Tracing for Radio Propagation Modeling: Principles and Applications," IEEE Access, vol. 3, pp. 1089-1100, 2015.
[17] M. Boban, J. Barros, and O. K. Tonguz, "Geometry-based vehicle-to-vehicle channel modeling for large-scale simulation," IEEE Transactions on Vehicular Technology vol. 63(9), pp. 4146–4164, 2014.
[18] M. Schiller, A. Knoll, M. Mocker and T. Eibert, "GPU accelerated ray launching for high-fidelity virtual test drives of VANET applications," 2015 International Conference on High Performance Computing and Simulation (HPCS), Amsterdam, pp. 262–268, 2015.
[19] M. Treiber, A. Hennecke and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," Physical Review E, vol. 62, pp. 1805–1824, 2000.
[20] A. Kesting, M. Treiber, and D. Helbing, "General Lane-Changing Model MOBIL for Car-Following Models," Transportation Research Record, 1999(1), pp. 86-âĂŞ94, 2007.
[21] L. Codeca, R. Frank, S. Faye and T. Engel, "Luxembourg SUMO Traffic (LuST) Scenario: Traffic Demand Evaluation," IEEE Intelligent Transportation Systems Magazine, vol. 9, no. 2, pp. 52–63, 2017.
[22] M. Gramaglia, O. Trullols-Cruces, D. Naboulsi, M. Fiore and M. Calderon,"Mobility and connectivity in highway vehicular networks: A case study in Madrid," Computer Communications, Volume 78, pp. 28–44, 2016.
[23] L. Bedogni, M. Gramaglia, A. Vesco, M. Fiore, J. Härri and F. Ferrero, "The Bologna Ringway Dataset: Improving Road Network Conversion in SUMO and Validating Urban Mobility via Navigation Services," IEEE Transactions on Vehicular Technology, vol. 64, no. 12, pp. 5464–5476, 2015.
[24] Google Maps Directions API. [Online]. Available: https://developers.google.com/maps/
[25] W. Qiu *et al.*, "UnrealCV: Virtual Worlds for Computer Vision," Proceedings of the 25th ACM international conference on Multimedia (MM '17), pp. 1221–1224, 2017.
[26] D. Garcia-Roger *et al.*, "5G multi-antenna V2V channel modeling with a 3D game engine," 2018 IEEE Wireless Communications and Networking Conference Workshops (WCNCW), Barcelona, pp. 284–289, 2018.
[27] S. Parker, *et al.*, "OptiX: A General Purpose Ray Tracing Engine," ACM Transactions on Graphics, 29(4), Article 66, 2010.
[28] OpenStreetMap. [Online]. Available: https://www.openstreetmap.org
[29] NVIDIA PhysX. [Online]. Available: https://developer.nvidia.com/gameworks-physx-overview
[30] Flatbuffers serialization library. [Online]. Available: https://google.github.io/flatbuffers/
[31] Z. Yun, M. F. Iskander and Z. Zhang, "Development of a new shooting-and-bouncing ray (SBR) tracing method that avoids ray double counting," IEEE Antennas and Propagation Society International Symposium, pp. 464-467 vol.1, Boston, MA, USA, 2001.
[32] D. Schramm, M. Hiller y R. Bardini, Vehicle dynamics, Springer, 2014.
[33] R. Rajamani, Vehicle dynamics and control, Springer Science & Business Media, 2011.
[34] T. E. S. I. S. DYNAware, "veDYNA Vehicle Dynamics Simulation in Real-Time: Overview,". [Online]. Available: https://www.tesis-dynaware.com/en/products/vedyna/overview.html
[35] N. Bell, J. Hoberock, "Thrust: A Productivity-Oriented Library for CUDA," in Applications of GPU Computing Series, eds. Wen-mei, W. Hwu, GPU Computing Gems Jade Edition, pp. 359-371, Morgan Kaufmann, 2012.
[36] L. Bieker, D. Krajzewicz, A.P. Morra, C. Michelacci and F. Cartolano, "Traffic simulation for all: a real world traffic scenario from the city of Bologna," Proceedings of SUMO 2014, pp. 15–16, 2014.
[37] Krajzewicz D. et al. "iTETRIS - A System for the Evaluation of Cooperative Traffic Management Solutions." in Advanced Microsystems for Automotive Applications 2010, Meyer G., Valldorf J. (eds), VDI-Buch. Springer, 2010.
[38] T. W. Anderson, "On the Distribution of the Two-Sample Cramer-von Mises Criterion," The Annals of Mathematical Statistics, vol. 33(3), pp. 1148–1159, 1962.
[39] Recommendation ITU-R P.2040-1, "Effects of building materials and structures on radiowave propagation above about 100 MHz", 2015.
[40] M. Martinez-Ingles, D. P. Gaillot, J. Pascual-Garcia, J. Molina-Garcia-Pardo and M. Lienard and J. Rodriguez, "Deterministic and Experimental Indoor mmW Channel Modeling," IEEE Antennas and Wireless Propagation Letters, vol. 13, pp. 1047–1050, 2014.
[41] R. Riebl, H. J. Günther, C. Facchi, and L. Wolf, "Artery: Extending veins for vanet applications," 2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS), pp. 450–456, 2015.
[42] E. Egea-Lopez and P. Pavon-Mariño, "Distributed and Fair Beaconing Rate Adaptation for Congestion Control in Vehicular Networks," IEEE Transactions on Mobile Computing, vol. 15, no. 12, pp. 3028–3041, 2016.