Tunnel Simulator for Traffic Video Detection

Sofie Van Hoecke*†, Steven Verstockt*†, Koen Samyn*†, Mike Slembrouck‡, Rik Van de Walle†
*Electronics and Information Technology Lab (ELIT), Howest, Ghent University Association, Belgium
†Multimedia Lab (MMLab), ELIS, IBBT - Ghent University, Belgium
‡Traficon, Trafic Video Detection, Belgium
Email: {sofie.van.hoecke, steven.verstockt, koen.samyn}@howest.be, ms@traficon.be, rik.vandewalle@ugent.be

***Abstract*—Testing and evaluating advanced traffic video detection algorithms and systems requires photorealistic source material to be generated. As recording real life traffic situations has severe limitations, a Tunnel Simulator is developed to create custom test scenarios within tunnels. The Tunnel Simulator enables the creation of custom tunnels by setting properties for the tunnel and individual traffic events. Based on the settings, a photorealistic scene is generated with the specified tunnel, traffic events and ground truth. The scene can then be previewed in real time 3D view and/or rendered in order to test the video detection algorithms on it. The presented Tunnel Simulator is the first high-quality traffic simulator that succeeds in generating photorealistic source material that can be used to evaluate traffic video detection algorithms.**

***Keywords*-simulator, traffic video detection, photorealism.**

I. Introduction

Over the years, traffic volume and complexity have been growing at a steady pace. As a result, traffic managers are faced with an increased demand in automated traffic monitoring systems. These systems do not prevent primary incidents, but once an incident has been positively identified, protocols for reducing secondary incidents are initiated such as tuning green-red cycles of traffic lights, closing the tunnel entrance to prevent collisions, adjusting tunnel ventilation to cut off oxygen supply to fires, or providing paramedics with accurate information and images.

New algorithms and features for automated traffic monitoring using video image processing need extensive testing in order to assess quality and reliability under various conditions. In order to test and evaluate new detection algorithms, accurate video source material is required. This real life recorded video source footage is then fed to the traffic monitoring system and the generated output is compared to a ground truth. In order to do so, a ground truth needs to be manually created for each relevant video feed. As traffic and system complexity grows, developers no longer find the appropriate source material. Despite numerous videos documenting real traffic incidents, many possible scenarios and events remain uncaptured on film. Up till now this has been resolved by creating and recording custom traffic scenarios. However, this method does not allow for dangerous situations to be created, e.g., driving a burning bus through a tunnel filled with regular traffic. Such an event is a good example of potentially catastrophic situations that have to be detected as soon as possible in order to minimize casualties.

As a solution, a Tunnel Simulator was designed and developed that generates user specified video source material and ground truth. The choice for focusing on tunnels was straightforward as a tunnel is potentially the most dangerous place for incidents, illustrated by the tragic disaster in the Gotthard tunnel in 2001 where a collision between two trucks resulted in a fire incident in which eleven persons died and that lasted over 24 hours before fire-fighters were able to bring the situation under control. Ever since, traffic video detection is mandatory in tunnels. Road tunnels must be equipped with the appropriate equipment for detection and monitoring, including sensors for temperature, visibility, CO_2, and smoke, as well as video cameras [1]. Incident detection in tunnels became a key safety aspect for every major traffic manager, and, even today, it remains a very important aspect of automated traffic detection systems.

The remainder of this paper is as follows. Section II outlines the state of the art. Subsequently, Section III defines the advanced features of our tunnel simulator. Section IV covers the internal design details. Next, Section V covers the evaluation results, after which we summarize the most important conclusions of our work in Section VI.

II. State of the art

During the past decades, a considerable amount of research has been done on developing real-time traffic surveillance algorithms based on roadside video aiming to extract reliable traffic state information [2]-[4]. However, the research is limited to detecting events such as stopped vehicles or car collisions in open road. As a result, ground truth material is widely available.

Also different traffic models [5] and simulators have been built [6]-[9]. These simulators are however intelligent transportation systems (ITS) and flow control systems and focus on optimal traffic flow control and calculating the most efficient route to go from A to B.

Closests to our research goal is the tunnel simulator of the Swedish Road Administration (SRA). The SRA has built a Tunnel Simulator [10] in order to provide training for traffic managers, staff, and paramedics. The large variety of objects (e.g., grass, trees, vehicles, billboards, buildings, railings and lights) allows to create accurate descriptions of real tunnels, but the quality of rendered computer-generated imagery (CGI) is low compared to today's standards. To the authors' knowledge, no high-quality traffic simulator has been reported upon yet.

III. TUNNEL SIMULATOR

Road tunnel operation depends heavily on traffic control and monitoring, as well as on well prepared and tested emergency and rescue plans [1]. However, the principle risk in road tunnels is the driver, a risk that can never be excluded. Therefore, new algorithms using video image processing are constantly developed to positively identify incidents better and faster.

In order to allow extensive testing of these algorithms to assess quality and reliability, a Tunnel Simulator was designed and developed to generate user specified video source material and ground truth. Figure 1 presents the general concept and Figure 2 a screenshot of the final result.

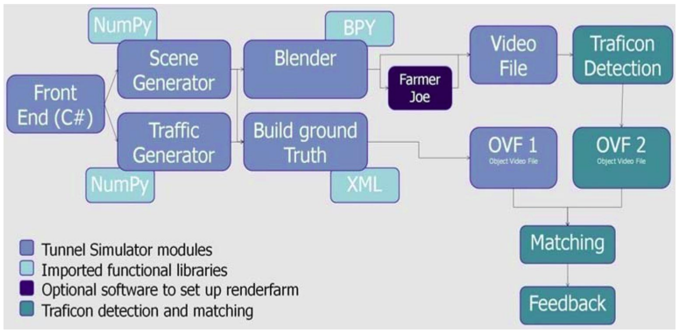


Figure 1. General concept of the Tunnel Simulator

As can be seen on Figure 1, the Tunnel Simulator consists of a scene generating module, a traffic generating module and a module to gather the user's input. The front-end enables users to create a custom tunnel by setting and editing tunnel properties such as height, length, lighting, angle and direction. Similar steps can be followed to create custom traffic events inside the tunnel. The front-end checks if properties are valid before passing them on to the scene and/or traffic generator. Communication between the generators and the 3D application is necessary to create a virtual world with traffic events. The scene generator provides the necessary information to build the tunnel, while the traffic generator adds vehicles and animation paths to the scene, to set up the animations. Each generator provides information to the ground truth module in order to build the based Object Video File (OVF). The 3D application will use its internal render engine to render the CGI. Optional tools (e.g., a render farm) are provided to speed up the rendering process. These loosely coupled modules ensure easy adaptation/replacement to meet future requirements. The Tunnel Simulator will generate two outputs: a video containing the rendered animation and a reference OVF file, containing a description what happens in the video. This video can then be fed to a detection system that will generate a second OVF file. Both OVFs can then be compared in order to assess and evaluate detection accuracy and quality.

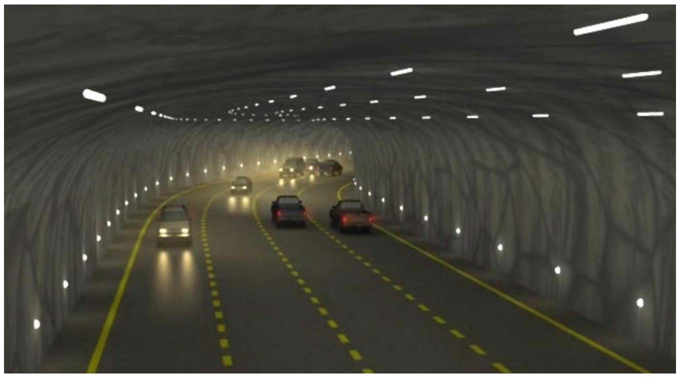
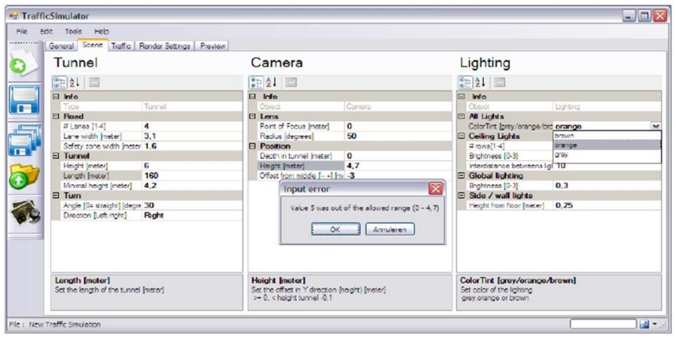


Figure 2. Screenshot of a created tunnel scene by the Tunnel Simulator

The Tunnel Simulator offers advanced features listed below:

1) **User-friendly creation and configuration:** Creating and configuring the tunnel and according scenes is straightforward by using a user-friendly interface so that application training can be minimized. Within this user interface (see Figure 3) all parameters can be set to shape a tunnel and define a traffic situation.



Tunnel

Info	
Type	Tunnel
Road	
# Lanes [1-4]	4
Lane width [meter]	3,1
Safety zone width [meter	1.6
Tunnel	
Height [meter]	6
Length [meter]	160
Minimal height [meter]	4,2
Turn	
Angle [0= straight] [degre	30
Direction [Left/right]	Right

Length [meter]
Set the length of the tunnel [meter]

Camera

Info	
Object	Camera
Lens	
Point of Focus [meter]	0
Radius [degrees]	50
Position	
Depth in tunnel [meter]	0
Height [meter]	4,7
Offset from middle [-- +] [m	-3

Input error
Value 5 was out of the allowed range (0 - 4,7)
OK Annuleren

Height [meter]
Set the offset in Y direction (height) [meter]
>= 0, < height tunnel -0.1

Lighting

Info	
Object	Lighting
All Lights	
ColorTint [grey/orange/bro	orange
Ceiling Lights	
# rows[1-4]	
Brightness [0-3]	
Interdistance betweens lig	10
Global lighting	
Brightness [0-3]	0,3
Side / wall lights	
Height from floor [meter]	0,25

brown
orange
grey

ColorTint [grey/orange/brown]
Set color of the lighting
grey,orange or brown

File : New Traffic Simulation

Figure 3. User interface for tunnel creation and traffic event configuration

2) **Photorealism:** The traffic detection algorithms use specular lightning spots, intensity and reflections to detect vehicles and objects so photorealism is required. In order to achieve this, for example, two light types are used per car: spots to light up the environment, and normal lamps as visible lights on car. Figure 4 shows the result.
3) **Adjustable tunnel shape and driving lanes:** Different shapes of tunnels can be created using the simulator by configuring straight and curved sections, the tunnel shape (e.g., cylinder, rounded rectangle), the height and width, amount of lanes, uni- or bidirectional

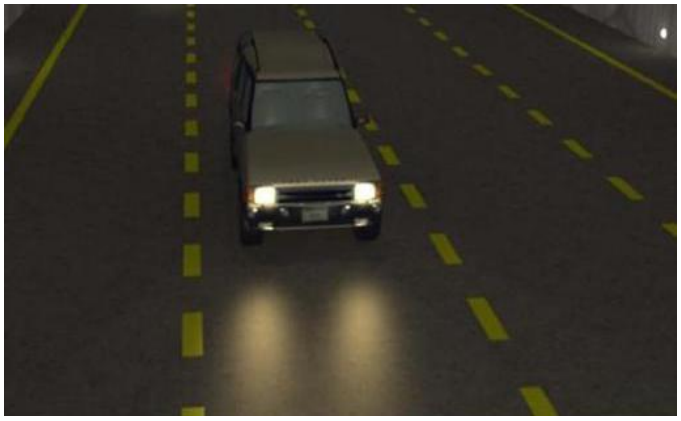


Figure 4. Modeling the car lights using two light types

traffic, and dead zones at the sides. Also markings are added on the road to visualize individual driving lanes. Both markings and lanes are in accordance with road administration guidelines.

4) **Custom traffic events:** Users can create a tunnel with custom traffic events by setting properties for each individual event. As each car gets a speed description in advance, this description will determine the car's behaviour during the animation. This way, any traffic event can be generated. Currently supported traffic events are: regular traffic, cars stopping on specific lane, traffic jam, falling objects (see Figure 5), and ghost driver. More events will be added in the future.

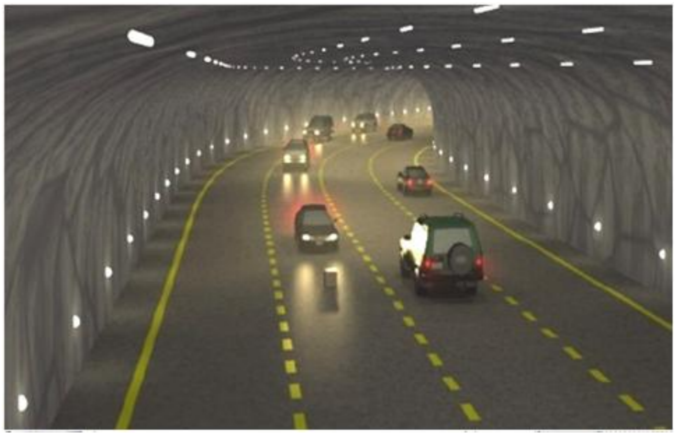


Figure 5. Traffic event in which fallen object hinders regular traffic

5) **Adjustable lighting:** Lighting is another crucial element for the scene generation as it changes perception dramatically. Height, position and amount of light sources can be configured, as well as the color, intensity and range for each light source. Slight variations in color and intensity change the level of perceived realism. This way, physically correct lighting can be achieved in the tunnel. This is important for detection algorithms as changes in intensity (e.g., specular lighting spots) and reflections are means of detecting vehicles and other objects.
6) **Adjustable camera settings:** According to the adjustable lighting, the amount, height and position of cameras can be configured, as well as the viewing angle, focal point and lens.
7) **Modular design:** The simulator is designed in a modular way, with loosely coupled modules so that third parties can create and/or integrate their own modules into the simulator. This way, the Blender render engine can be easily replaced by more performant render engines such as 3ds Max and Maya. Additionally, new scenarios, such as urban settings and highways, can be built and easily integrated into the tunnel simulator.
8) **Material / texture for road and tunnel:** As each material has unique properties and reacts differently to light, the best materials are chosen in order to ensure proper perceived realism. Currently one preset material is used for the road, and another preset material is used for the tunnel. However, due to the modular design, easy integration of new materials and textures is possible, should the need arise.
9) **Preview functionality:** The simulator provides the ability to preview current settings at all times, in order to facilitate fine tuning and check settings before starting the rendering process. A simplified visualization of the generated tunnel and traffic events is herefore constructed and allows real time 3D preview. Once a satisfactory tunnel is constructed, users can save the tunnel settings in an XML-file. Scene and traffic specifications can be saved separately, enabling users to create custom content blocks and linking them.
10) **Performance optimization:** Based on the parameters set in the user interface, a scene is built and the traffic flow is created. The animation is rendered at a chosen quality, resulting in a video file and photorealistic, accurate image suitable for video detection. In order to improve performance, only what can be seen is created. As viewing distance inside tunnels is obstructed by curves, this improves the rendering process by not wasting resources on visualizing invisible objects. Additionally, quality options such as resolution, oversampling, ambient occlusion and motion blur can be turned down to speed up the render process. And finally, a render farm can be used for quicker results. In our implementation, Farmerjoe was used, but thanks to the modular design, this can easily be replaced by another render farm.

IV. Internal design details

The Tunnel Simulator has been implemented and is currently evaluated by Traficon. Below is an overview of the main internal design details.

A. Constructing the 3D tunnel using Blender

Due to the high license costs of 3ds Max and Maya, Blender was chosen to develop the Tunnel Simulator. Blender features an internal render engine capable of features such as ray-tracing, motion blur, oversampling and ambient occlusion, but lacks real global illumination capabilities, apart from using the radiosity solver, which requires enormous amounts of processing power.

Creating a 3D tunnel model using Blender can be done in several ways. One approach is to use basic objects such as triangles and squares, and use them like building blocks by scaling, translating and rotating them in order to create surfaces. Joining multiple surfaces creates objects, called meshes. The process of joining multiple surfaces can be compared to welding two metal sheets together. Another method for constructing a 3D tunnel, and the one we used, is by moving a cross section along a path in order to define the tunnel's shell, similar to extrusion. The first step is creating a guiding path for the cross section, such as a curve. Blender has three main types of curves: Bezier curves, NURBS and paths. Essentially, paths are the same as Bezier curves, but are initially locked to a 2D plane and have a start and ending point, giving them a direction from start to end. Any of these types can be used as a guiding path for the cross section. Figure 6 displays a path, which was used to create the outer shell. The path is constructed by seven control points, called knots. The curve is generated by a Bezier algorithm. All kinds of free form curves can be created by manipulating knots (e.g., translations, insertions and adding weight to knots).

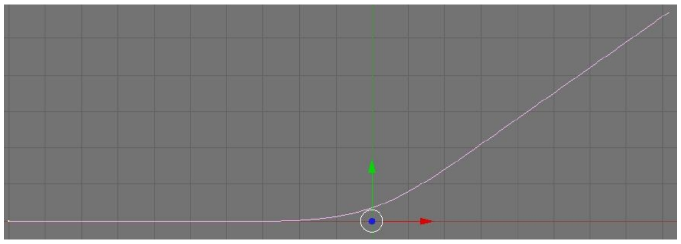


Figure 6. A path created by using Blender

Once the path is created, a cross section needs to be constructed. This cross section will then be extruded along the path in order to create the shell. The cross section is created by using a closed curve such as a Bezier circle or NURBS circle. The actual shape in Figure 7 is generated by four Bezier circles, each one creating a quadrant and joined together. The final step for creating the shell is extrusion, realized by setting the cross section as the curve's bevel shape (see Figure 7). The main advantage of this method is its easy adjustability. By adjusting the knots and handles, the shape, length and curvature of the tunnel can be adjusted.

Finally, a road needs to be constructed and placed inside the shell. This process is similar to creating the shell. A new

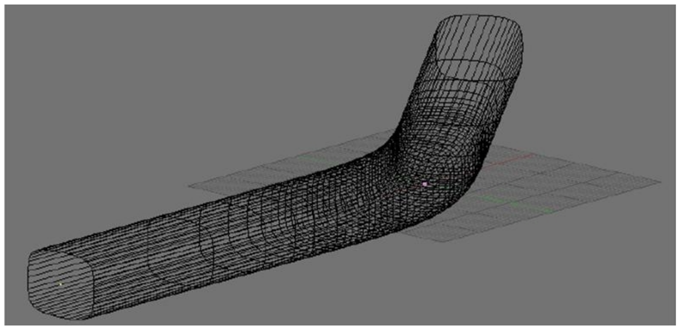


Figure 7. Modeling the 3D tunnel with Blender

cross section needs to be constructed. This object is a line with identical width as the tunnel cross section. An identical path is created by duplicating the tunnel path and placing it on top of the tunnel path.

Once the tunnel shape is created, materials are assigned to it in order to define its appearance. Each material has unique properties, and reacts differently to light. Therefore, different (fixed) materials are chosen for the tunnel and road to ensure proper perceived realism. As already stated, due to the modular design, easy integration of new materials and textures is possible, should the need arise.

Changing the tunnel's width and cross section shape cannot be directly controlled by users through the graphical user interface in order to prevent mismatches between cross section diameter and road settings. The user is only allowed to change road settings such as number of lanes, driving lane width and dead zone space. Adjusting these settings will automatically cause the tunnel cross section to scale to a suitable size.

B. Creating the vehicle database

The 3D cars and SUVs used by the Tunnel Simulator are provided by Marlin Studios. Each vehicle is constructed by approximately 2100 to 4000 polygons and is made up by four groups of sub objects and a high resolution texture: glass windows, body, interior and wheels. The high resolution texture is improved by implementing normal mapping and ambient occlusion baking. Most of the car's interior was removed since this is not visible when rendering.

As no (free) models of trucks with good quality and level of detail for Blender could be found, we built the truck model ourselves. The model uses 3789 polygons and texture mapping to provide extra detail. Figure 8 presents the three different trucks currently supported in the Tunnel Simulator.

C. Creating the lighting setup

Now that the scene and vehicle database is created, the lighting setup needs to be constructed to mimic lighting conditions inside a real life tunnel. Adding lights to a closed scene can change perception dramatically.

There are five different kinds of lights available in Blender: a normal lamp, sun, spot, hemi and area light. Testing indicated that spots produced the most realistic lighting

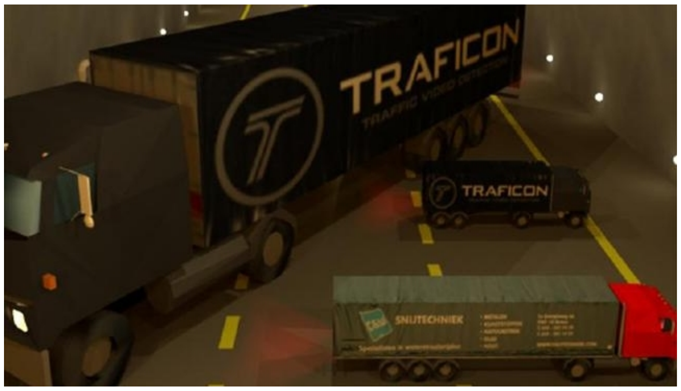


Figure 8. Blender truck models

effect inside the tunnel, followed closely by regular lamps. However, regular lights considerably reduce rendering time, making these lamps the best choice for the scene generator.

Finding the correct lighting setup was done by studying real videos taken inside a tunnel and analyzing intensity, color and positioning, and trying out those settings inside the virtual environment. Additionally, the lighting setup must be flexible in order to correspond with tunnel customization. In Blender this problem can be solved by parenting the lights to an identical Bezier curve that defines the tunnel's path. Next, the lights need to placed at a regular interval along the tunnel, creating a specific pattern. This process can be accelerated by using Blender's DupliFrame function.

DupliFrames stands for Duplication at Frames and is a modeling technique for objects which are repeated along a path, such as the wooden sleepers in a railroad, the boards in a fence or the links in a chain and thus also for creating the lighting setup along the tunnel's path. Figure 9 displays the results from the DupliFrame process. In this case, 22 lights are equally placed along the tunnel's path. This process is also used to position lamp placeholders along the ceiling and sides of the tunnel and to place markings on the road.

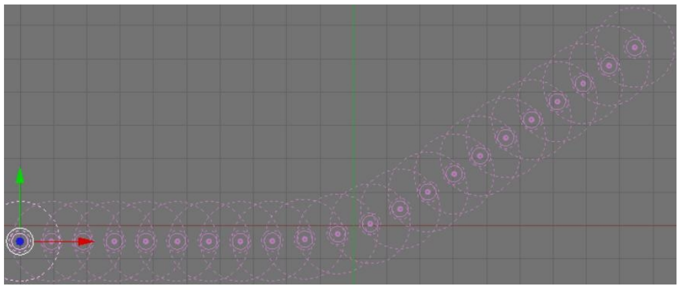


Figure 9. DupliFramed lights are equally placed along the tunnel's path

D. Adding a camera to the scene and generating traffic

The next step for the scene generator is defining a view by placing a camera into the scene. Rendered images are created from the camera's point of view, similar to shooting a real life video. At this point the entire 3D tunnel has been built, a lighting setup has been added, and the camera has been placed inside the tunnel.

The final step is generating the traffic, linking the traffic to the tunnel by copying the tunnel's path, and setting up individual animations for each vehicle.

The real time event character of game engines, where one can intervene and influence the scene in real time, is not possible using Blender to slow down or speed up a car. Neither is it possible to detect speed changes of cars in order to slow down or speed up other vehicles too. As Blender can only render predefined animations, each car needs to have a given speed description.

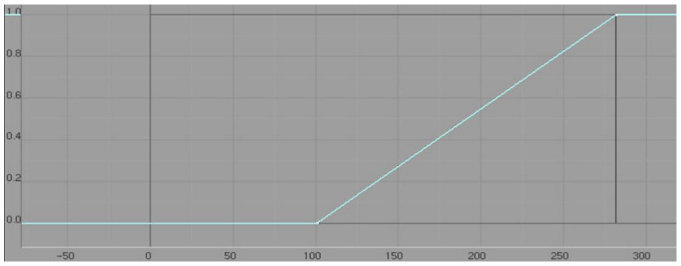


Figure 10. Exemplary speed description for a car

Once the animation is created and starts, nothing can be changed. Whenever the user changes some settings, the animation needs to be built again. Each animation consists of three parts: (i) the object that is animated, in this case the vehicle, (ii) the path to follow, i.e., an offset to the middle of the tunnel in order to choose a lane, and (iii) the speed description using interpolation curves. Note that a small, random deviation is added to the offset so that not every car drives exactly in the middle of the lane. Figure 10 presents an exemplary speed description for a car. The speed of a car can be set by tuning the frame rate. This frame rate is divided by the requested speed (in m/s), and multiplied with the total length of the visible route section (in m). This way, the number of frames for the entire route is retrieved. As presented in Figure 10, from the perspective of one specific camera, the car starts to drive at frame 100. About 180 frames later, the car arrives at the end of the visible route section for that camera. As the speed follows a straight line from 0 to 1, the position of the car will change linearly in time. As a result, the car will drive with constant speed from start to end. The speed description uses Bezier curves, composed out of Bezier triples (two handlers, one knot), linked by a smooth line.

Normal traffic occurs when no special events, such as traffic jams or collisions, occur during the traffic flow. In this case, the user can set four parameters: speed, flow rate (i.e., cars per minute), statistical distribution of vehicle classes, uni- or bidirectional traffic. These four parameters are used to simulate the desired traffic by repeating the animation for a random car and placing it in the scene. A Gaussian distribution is used to average the distance between the cars.

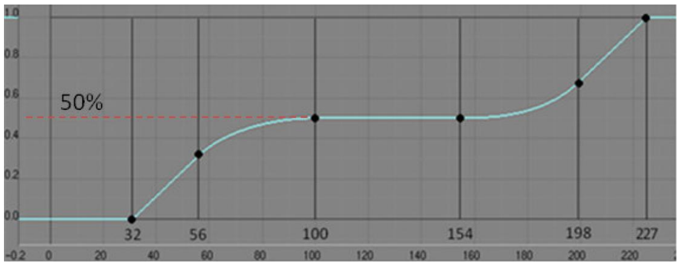
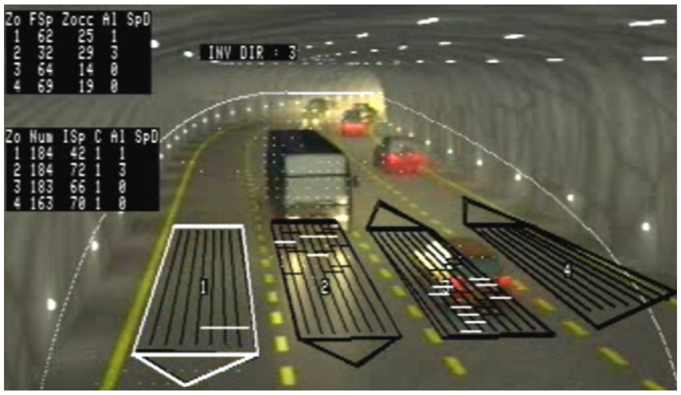


Figure 11. Speed description for a car that halfway stops and then speeds up again



```
Zo FSp Zocc Al SpD
 1  62   25  1
 2  32   29  3
 3  64   14  0
 4  69   19  0
```

```
INV DIR : 3
```

```
Zo Num ISp C Al SpD
 1 184  42 1  1
 2 184  72 1  3
 3 183  66 1  0
 4 163  70 1  0
```

Figure 12. Detection testing using video generated by Tunnel Simulator

Also special traffic events can be implemented. Figure 11 presents the speed description for a car that stops halfway.

The car after a stationary car should also stop to avoid a collision. Therefore, a custom speed description for these cars need to be provided as well by slightly shifting the new speed curve to the right and lowering this curve to make the car stop after this first car. A for loop is used to repeat this and let more than one car stop after the stationary car.

In order to make the stops more photorealistic, both back lights and flashes are added to the cars, lighting up when the car brakes, respectively warning when standing still. Both features are implemented using layer descriptions.

The same way, also traffic jams, falling objects, and ghost driver events are implemented in the Tunnel Simulator.

V. EVALUATION

The Tunnel Simulator fully works according to the specifications. Users are able to create a small tunnel segment (a single driving lane) to a large (four driving lanes) tunnel segment. Evaluations at Traficon indicated that all scene generator properties are intuitive for most of the users. Users were able to create the desired tunnel with custom traffic events, which ultimately resulted in a movie and OVF file. Detection testing at Traficon (see Figure 12) indicated that the generated movies are well suited for their intended purpose, with high correlation between both OVF files. Video footage and a demonstration of the finished project can be found on [11].

VI. CONCLUSION

The developed Tunnel Simulator enables users to create a tunnel with custom traffic events by setting properties for the tunnel and each individual event. As a result, a scene is generated with the specified tunnel, traffic events and ground truth. The scene can then be rendered in order to use it for testing video dectection algorithms. This way, the quality and reliability of new video detection algorithms can be extensively tested without the need to create and record dangerous traffic situations to have video source material.

Future work includes the development of new scenes (e.g., intersections), vehicle tracking, the addition of fire and smoke events (e.g., burning vehicle), and pedestrian traffic.

ACKNOWLEDGMENT

The authors would like to thank Frédérique Vanneste, Karel Azou and Kristof Hennebel for their valuable work.

REFERENCES

[1] M.P. Müller, Tunnel Safety: where are we now, Swiss Re, Risk Engineering Services, 2005.

[2] G. Wang, D. Xiao, and J. Gu, Review on vehicle detection based on video for traffic surveillance, Proc. of IEEE International Conference on Automation and Logistics, pp. 2961-2966, 2008.

[3] A. Bevilacqua and S. Vaccari, Real time detection of stopped vehicles in traffic scenes, Proc. of IEEE Conference on Advanced Video and Signal Based Surveillance, pp. 266-270, 2007.

[4] O. Akoz and M.E. Karsligil, Video-based traffic accident analysis at intersections using partial vehicle trajectories, Proc. of 17th IEEE International Conference on Image Processing, pp. 4693-4696, 2010.

[5] N. Farhi, M. Goursat, and J.P. Quadrat, The traffic phases of road networks, Transportation Research Part C, 19(1), pp. 85-102, 2011.

[6] J. Miller and E. Horowitz, FreeSim - a free real-time freeway traffic simulator, Proc. of IEEE Intelligent Transportation Systems Conference, pp. 18-23, 2007.

[7] T. Ishikawa, Development of a road traffic simulator, IEEE Vehicular Technology Society, 47(3), pp. 1066-1071, 1998.

[8] T. Winters, M. Johnson, and V. Paruchuri, LITS: Lightweight Intelligent Traffic Simulator, Proc. of International Conference on Network-Based Information Systems, pp. 386-390, 2009.

[9] L. Yong and C.Y. Li, A microscopic simulator for urban traffic systems, Proc. of 5th IEEE International Conference on Intelligent Transportation System, pp. 622- 626, 2002.

[10] P. Wessel and J. Rossberg, Increase traffic safety by increasing traffic management skills using advanced training simulator, Proc. of 4th International Conference on Traffic and Safety in Road Tunnels, pp. 1-10, 2007.

[11] Tunnel Simulator demonstration, http://elit.howest.be/demos.