18 Making Traffic Visualization Movies by Scripting a Graphical User Interface

Daniel Krajzewicz;
German Aerospace Center, Berlin, Germany

18.1 Abstract

This report presents a recent extension to the open source traffic simulation "SUMO", which allows to generate demonstration movies by scripting the user interface from an external application. The scripting was realised by extending an available on-line interface. For obtaining a final movie, further steps are necessary, which are described. Examples for using the extension are given and discussed.

Keywords: Road Traffic Simulation, Visualization, Application Programmer Interface.

18.2 Introduction

SUMO (Simulation of Urban Mobility, [1], [2]) is a microscopic road traffic simulation, developed at the Institute of Transportation Systems at the German Aerospace Center. SUMO's development started in 2001, with a first version released for open use in 2002. Since that time, SUMO evolved to a mature state and became a popular simulation tool, mainly used in academic context for research on vehicular communications (V2X), see also [3]. SUMO has a built-in graphical user interface (GUI), based on a 2D visualization using openGL [4] which is embedded in a FOX-Toolkit [5] window.

The open source nature of SUMO allows external parties to extend it for meeting their purpose. In 2008, staff members from the Technical University of Lübeck, Germany, supported a simulation scripting interface based on a socket connection, which allows an on-line interaction between a running simulation and an external application [6]. The interface is called "TraCI", an acronym for "Traffic Control Interface". In the recent past, this interface was reworked, extending it by further possibilities to access and manipulate objects that contribute to the simulation and splitting the access to different kinds of objects into separate modules for a cleaner implementation.

Within this report, a recent extension to the scripting interface is presented, namely the ability to script the simulator's GUI using an external script or program. The extension was motivated by the need to generate a demonstration movie for the PRE-DRIVE C2X project. A very common approach for generating movies, is to use a tool that periodically captures the contents of the computer screen – either completely or for a chosen area of the screen only. This method has two major disadvantages. At first, as the tool runs in parallel to the "recorded" application, it competes for resources, such as CPU time or access to the hard discs. This often yields in delays in executing one of the applications, visible in the resulting movies. The second is, that usually a movie contains complex movements of the camera and/or zoom changes. If a screen capture application would be used, then the user had to adapt the view to such a story plot manually and in real time. It is hardly possible to perform this in a perfect way.

As TraCI was undergoing a restructuring process at the same time, the idea to use it in combination with already available possibility to save screenshots via the SUMO GUI, was quite straightforward. "Atomic" actions for interacting with a "view", the instance visualizing the simulation scenario, were recognized and implemented into a new TraCI module. Besides forcing sumo-gui to take snapshots of the visualized scenario and save them into image files, these "atomic" actions allow, for example, setting the camera's zoom and position to a given value. As usually the camera is moving to a new position and the zoom is smoothly adapted to a new value, a higher-level API was built on top of these atomic actions. Further post-processing is used to embed decals and/or additional information in the generated images and combine them into a presentable movie.

The remaining part of this report is structured as following: at first, SUMO's user interface and the on-line interaction API are presented, upon which the scripting interface was implemented. Then, the scripting interface itself is introduced, followed by a description of a higher-level interaction API which encapsulates the access to the scripting interface for an easier set-up of non-atomic operations. One possible post-processing solution is presented afterwards, followed by examples where the complete process was employed. Then, some further observations done while using this approach for generating movies are given. The report closes with a summarizing section including possible future extensions, and acknowledgements.

18.3 Graphical User Interface in SUMO

The applications belonging to the SUMO suite were originally set up as command line applications. Later, a graphical user interface was built upon the already existing simulation application for demonstration purposes. It is a multi-document window, which allows to open different views at the loaded scenario. The user may zoom into and move across the network using the mouse. The GUI supports different colouring schemes for evaluation of the current vehicle behaviour or of the state of the road network. In addition to roads, intersections, and vehicles, infrastructure objects, such as simulated sensors, bus stops, or traffic lights are rendered, as well as abstract shapes – polygons and points of interest (PoIs), which do not participate in the simulation process. Both, polygons and PoIs can be assigned to "layers" for ordering. In combination with network and shape importers included in the SUMO package, the GUI allows to present complex scenarios with a high range of detail, see Figure 18-1.

Figure 18-1: Two sample screenshots from the same scenario (top: scenario overview, bottom: zoomed at an intersection).

The GUI was implemented by deriving visualization classes from already existing simulation classes that represent artefacts the simulation model consists of, such as lanes, vehicles, bus

stops, etc. Instantiation of the right class is done by using factory classes [7]. Figure 18-2 shows this approach by example for bus stops.

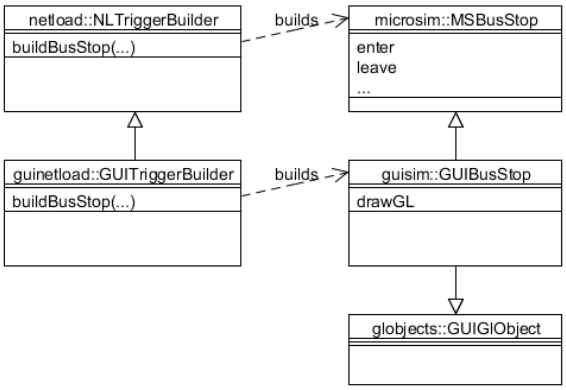


Figure 18-2: Building visualization classes derived from simulation classes using factories; this example shows the building process of simulated bus stops.

Besides implementing the drawing method, each visualization class is additionally derived from an abstract `GUIGlObject` class and implements this class' interfaces, such as:

- Retrieving the object's id and name (used to determine the object below the cursor);
- Retrieving the object's bounding box for centering the view around the object;
- Opening an interaction popup dialog;
- Opening a table which shows the object's fixed and changing parameter;
- Drawing the object's name.

SUMO was designed to handle large road networks, see [1] and [8] for examples. For increasing the rendering speed, the GUI takes advantage of an R-tree structure [9] [10] that calls the visualization classes' drawing methods for visible objects only. In parallel, the level-of-detail is increased when zooming into the road network. For example, bus stop names or vehicles are rendered only, if being at least one pixel in size. Vehicles, though visualized, are not stored in the R-tree directly, but are rendered by the roads they are currently placed at. This was done to avoid the need to update the vehicles' positions within the R-tree. The c++ R-tree implementation by Greg Douglas was used [11].

When traversed, the R-tree calls each found object's "`drawGL`" method, passing a structure named "`GUIVisualizationSettings`". This structure holds information about how objects shall be drawn, e.g., which values or methods shall be used for determining a lane's or a vehicle's colour. The visualization settings structure holds an enumeration value for the chosen colour scheme and the lane, when being drawn, calls the appropriate method to determine the value and chooses the colour representing this value from a user-defined look-up table.

Other possibilities exist to adapt the rendered objects' appearance to one's needs. Vehicles, points of interest, signs and detectors can be exaggerated in size. Their names can be displayed if wished, using custom colour and size. Additional "decal" images, usually photos taken from planes or satellites, can be loaded and displayed. After being edited in a dialog, the current visualization setting can be saved to a file and included in the simulation configuration. The file format is, as almost all files read and generated by SUMO applications, based on XML and described using a XML schema definition.

The GUI is capable to save screenshots of the currently displayed area in different bitmap and vector formats, among them .gif, .bmp, .tif, .svg, .ps, and, if compiled with the according

additional libraries, .png and .jpg. Image writing capabilities are realized using FOX-Toolkit's native image support for bitmap graphics, whereas vector output is realized using the gl2ps library [12].

18.4 Online-Interaction Interface

The simulation application SUMO includes a socket based on-line interaction interface named "TraCI". When being started with the command line argument "--port-number", the simulation does not start immediately after loading the scenario, but opens a socket connection and starts to wait for a connecting client. After connection, the client is responsible for triggering simulation steps and for ending the simulation.

The client has the access to a large number of objects the simulation consists of, including vehicles, lanes, edges, intersections, traffic lights, simulated sensors, etc. Besides retrieving values from named objects, TraCI also allows to change their behaviour. For example, it is possible to change the current plan of a traffic light, change the speed of a vehicle, etc. The complete – still growing – list of accessible objects and values is described within SUMO's documentation at [2].

18.5 Basic GUI Interaction

The scripting interface for the GUI is built upon TraCI and the current message format. Each opened openGL view can be addressed in the same way as the simulated structures and allows to retrieve and change its settings. Table 1 shows which values of a view can be accessed.

Table 18-1: Parameter of a view that can be retrieved or set.

Value	Get	Set	Type
view ids	X		string list
zoom	X	X	double
offset	X	X	double*2
schema	X	X	string
boundary	X	X	double*4
track vehicle		X	string
screenshot		X	string

Here, "view ids" denotes the list of opened views, "zoom" an abstract factor by which the shown area is scaled, "offset" the offset of the view's centre from the loaded network's centre in meters, "schema" the name of the used visualization scheme, "boundary" the coordinates of the right upper and the left bottom corners of the visible part of the network. "Track vehicle" allows to centre the view on a named vehicle and the camera keeps the vehicle in the centre when running the simulation. "Screenshot" saves the current view under the given name locally on the computer the simulation is running on. The file format is defined by the given file name's extension.

Besides interacting with the view, it is also possible to add and to remove polygons and points-of-interest (PoIs) and to change their appearance. Additionally, the simulated vehicles' colours can be changed via the interface.

18.6 Higher Level Interaction

The presented on-line interface allows to interact with a single view by setting its values directly. But usually, a movie consists of complex camera movements, including fluid changes of the zoom and/or the position the camera looks at. The functionality realizing such actions was implemented in Python. A "script" consisting of entries with the fields "begin", "end", and "action" was processed. Sorted by "begin", the "action" was executed as soon as the simulation reached the simulation step "begin", and until it reached "end". "begin" and "end" are floating point numbers which should match the used simulation step size's granularity. Table 2 presents the implemented "actions".

Table 18-2: Actions for interacting with a view.

Value	Parameter	Type
snapshot	<OUTPATH>	Saves a snapshot in each step between *begin* and *end* to <OUTPATH>. <OUTPATH> may contain the placeholder '%s', which is replaced by the current image number.
move_at	<VEHICLE_ID>	Interpolates "offset" between the position at *begin* and the position of the vehicle with the given id *end*
move_to	<POSITION>	Interpolates "offset" between the offset at *begin* and <POSITION> at *end*
track	<VEHICLE_ID>	Keep "offset" at the position of the vehicle with the given id. If an empty string occurs, "tracking" ends.
zoom_to	<DOUBLE>	Interpolates "zoom" between the value at *begin* and <DOUBLE> at *end*
show_ego_range		Obtains a measure for communication quality and uses it to colour vehicles around the ego vehicle.
show_per		Draws a circle around vehicles chosen as "equipped".

Taking "snapshot" apart, the actions fall into two groups: simple interpolation between values, may them be positions or zoom levels, and complex scripts that are strongly dependent to data from the movie generating client. While the first group could be included in SUMO's API, there are no plans for realising a user-friendly solution for the second group of actions.

18.7 Post-Processing

As described, the GUI is capable to store single screenshots only. Using tools such as VirtualDub [13], available under the GPL, they can already be replayed as a movie, though often not very fluidly, as no compression is yet applied, so that both, reading the image files from the hard discs, and displaying them is more time consuming than reading a movie file. For generating movies, these images have to be merged into a single, usually compressed file. Also, it is often wished to have some additional decals, explanation text, or copyright information, see Figures 18-4 and 18-5. To realize this, we use the procedure depicted in Figure 18-3, which is described in the following.

Single screenshots are generated as described before. Afterwards, we use a second script for embedding decals and/or other additional information into the original images, one by one. This script is also responsible for cropping the images to their final size. Our script, implemented in Python, uses the "convert" application from the open source image manipulation library ImageMagick [14] for image processing. The last step, merging of the images into a compressed movie, is done using VirtualDub, which is also responsible for compressing the movie.

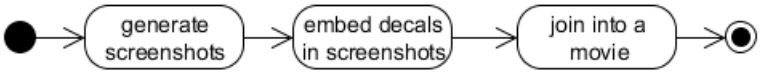
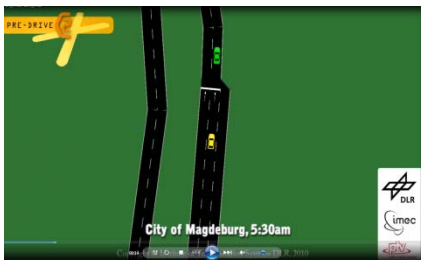


Figure 18-3: Steps for obtaining a movie.

It should be noted that the post-processing steps described here are surely not matching any state-of-the-art approaches. Sophisticated tools designed for such purposes only exist. Such tools are probably faster and more user friendly than performing the post-processing steps by scripting. The method described here is given to deliver a complete description about generating movies.

18.8 Examples

The visualization scripting API was already used to prepare presentation material within the projects iTETRIS and PRE-DRIVE C2X, both co-funded by the European Commission and both working on vehicular communication (V2X). Figure 18-4 shows two screenshots from a movie, which introduces the vehicular communication model developed in PRE-DRIVE C2X, described in [15]. The movie starts with a single "ego" vehicle (Figure 18-4a) and ends with an overview of the V2X coverage of the road network (Figure 18-4b). The penetration rate of vehicles equipped with a V2X device is 10%. The transmission ranges on the right side are visualized using polygons.

a) 

b) 

Figure 18-4: Screenshots from a PRE-DRIVE C2X movie.

The "ego" vehicle was coloured in yellow, while other equipped vehicles were coloured in dependence on the probability of receiving a message from "ego". Non-equipped vehicles were coloured in grey. For visualizing the coverage, each equipped vehicle was coloured in yellow and additionally a circle-shaped polygon with the radius of communication range (here: 300m) was put below it.

Figure 18-5 shows screenshots from two movies from the iTETRIS project which demonstrate traffic management applications based on V2X. Image 5a visualizes speed collection using cooperative awareness messages (CAMs), where each dot, realized using added Pols represents one collected value and the colour of the dot represents the speed ranging from 0m/s (red) to 13.9m/s (green).

Image 5b puts two simulations together. In both cases, an emergency vehicle is simulated, once without (left) and once with a V2X-controlled traffic light adaptation to incoming emergency vehicles (right). The emergency light was rendered using the AnoP library [16].

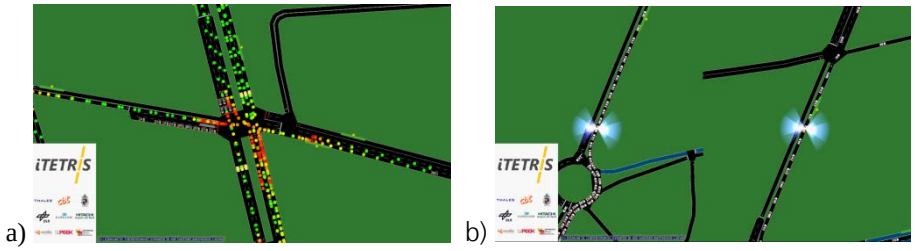


Figure 18-5: Screenshots from two iTETRIS demonstration movies.

In both cases, adding text, logos and copyright information was done by post-processing single images as described earlier.

18.9 Observations and Advices

While working on the presented examples, some experiences have been made, which are shared in the following.

18.9.1 Resolution

Some standard resolutions exist [17], partially shown in Figure 18-6. The screenshots taken by the API – as all screenshots from sumo-gui – have exactly the size of the display area. As the display area is surrounded by borders, it is quite difficult to set the size of the display area to a wished resolution of the images to generate. The size of the borders must be added to the desired resolution, and, as no other method is available by now, the window size would have to be set to the resulting dimensions manually. In addition, the window cannot be larger than the screen size, limiting the resolution of the generated movie to the screen size minus window border sizes.

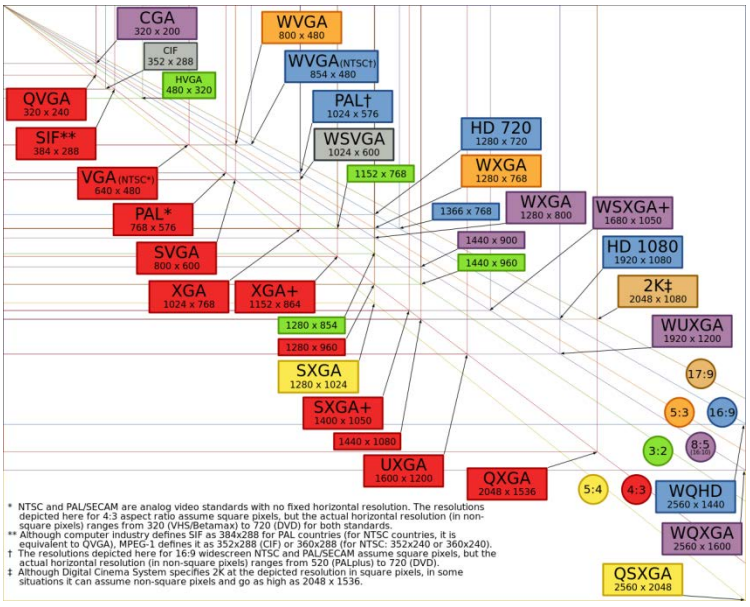


Figure 18-6: Common Video Standards Resolutions [18].

One could think of possible extensions, such as opening a full-screen openGL-view or exporting images in a higher resolution by generating them via an off-screen buffer. Both methods are not available, currently. The only possibility to obtain images in a desired size was to generate larger images, and pruning them during the post-processing step.

18.9.2 Frame Rate

The frame rate [19] of a movie describes how many images ("frames" in this context) are shown per second. 24 or 25 fps (frames per second) are common. In the context of the purpose presented here, the frame rate should be put against the simulation step size, as running the simulation with a step size of 1 s and generating a movie with 24 fps yields in a very fast, hard to follow visualization of the scenario, especially if the camera is moved.

As long as the camera is focussing on a single position on the network, or tracking a single vehicle, like in both iTETRIS scenarios, using simulation step sizes of one second and a frame rate of 10 fps brought satisfying results. The PRE-DRIVE C2X movie, including different kinds of camera movements, was generated with a frame rate of 24 fps and a simulation step size of .2 seconds.

18.9.3 Technical Parts of Story Telling

The shown examples differ in story telling. Within the iTETRIS movies, the camera was fixed – either to an intersection, or to a vehicle. But the PRE-DRIVE C2X example was meant to contain a more complicated plot, as described earlier. The realisation of this story into pictures turned out to be more challenging than the implementation of the APIs, due to the following, non-functional issues.

9. In the first transition from a complete scenario view towards the "ego" vehicle, the camera was moved at a high zoom (low altitude) along the scenario. It was found that this confuses the person watching the movie, as streets pass by at a high speed without any remarkable context. Using a lower velocity yielded in a too long "prelude", so that several – time consuming – attempts had to be done to get an acceptable combination of zoom and movement.
10. If the complete communication range, e.g. 300 m, of a vehicle shall cover the complete visualized area (1280 x 720 pixels) then a vehicle is about 2 x 4 pixels in size. This is too small to recognize the colour of other passing, V2X-equipped vehicles, which shall be coloured by the probability to receive a message from "ego". In contrary, when zooming more towards the "ego" vehicle, the development of the connectivity is incomplete. In addition, the other vehicles pass too fast if running into the opposite direction than the "ego" vehicle.
11. It should be noted, that a demonstration movie is usually presented to a bigger audience, and it is hardly predictable which parts of the shown scene get into a single viewer's focus, especially if the movie is shown more often than once. As a result, the maker should be aware that any erroneous or strange behaviour will be visible.

To summarize, further traditional movie making skills – those of a director or a director of photography – are supposed to be of help.

18.9.4 Overall Performance

In comparison to the number of iterations needed to adjust the movement of the camera, the duration of generating the images and post-processing them into a movie was not found to be critical.

The major bottleneck was posed by the size and number of generated images. In all presented cases, it was decided to use .bmp images as output format, as they are lossless and offer a colour resolution of 24 bpp (bits per pixel). Such an image is about 3 MB in size. For a

movie of one minute length with a frame rate of 24 fps, a hard disc storage of more than 4 GB is needed.

18.10 Summary and Outlook

The visualization scripting interface has already proved to be a usable tool for generating presentation material and we assume some of the methods described herein to be valuable for the developers of other simulation packages as well. It is worth to mention that all the steps yielding in a movie could be performed using open source applications only.

Currently, the simulation supports the possibility to connect only to a single client. As the simulation is often used in conjunction with other clients than the described movie-making script, it is currently necessary to embed the movie-making script into the client which realizes the desired simulation behaviour. This need can hopefully be solved in the future by allowing more than one client to connect to the application.

In the future, some work is assumed to be put into a 3D scene representation. First steps towards a 3D view were already done, incorporating the OSG library embedded directly into the same window. Figure 18-7 shows a screenshot of the currently implemented 3D view. Releasing the 3D view as open source is under discussion.

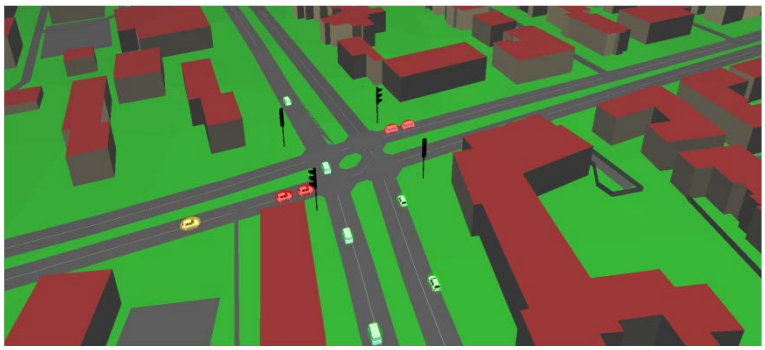


Figure 18-7: A 3D view on the simulation, currently under implementation.

18.11 Acknowledgements

We want to thank the European Commission which co-founded two projects where the developed extensions were needed, namely PRE-DRIVE C2X and iTETRIS. We also want to thank the authors and contributors of the open source and/or free projects which were used to obtain the complete movies.

18.12 References

[1] Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L.: Recent Development and Applications of SUMO - Simulation of Urban MObility. In: International Journal on Advances in Systems and Measurements, 5 (3&4), pp.128-138. ISSN 1942-261x (2012)

[2] DLR and contributors: SUMO homepage. http://sumo.sourceforge.net/ (2013)

[3] Krajzewicz, D.: Summary on Publications citing SUMO, 2002-2012. In: Proceedings of the SUMO2013 Conference (2013)

[4] Khronos Consortium: openGL homepage. http://www.opengl.org/, last visited on 08.04.2013

[5] Van der Zijp, J.: FOX-toolkit homepage. http://www.fox-toolkit.org/, last visited on 08.04.2013

[6] Wegener, A., Piórkowski, M., Raya, M., Hellbrück, H., Fischer, S., Hubaux, J.-P.: TraCI: An Interface for Coupling Road Traffic and Network Simulators. In: 11th Communications and Networking Simulation Symposium (2008)

[7] Gamma, E.: Design Patterns. Addison-Wesley Publishing Company, ISBN 0-201-63361-2 (1995)

[8] Krajzewicz, D., Bonert, M., Wagner, P.: The Open Source Traffic Simulation Package SUMO. In: RoboCup 2006 (2006)

[9] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: Proceedings of the 1984 ACM SIGMOD international conference on Management of data - SIGMOD '84. pp. 47. doi:10.1145/602259.602266. ISBN 0897911288 (1984)

[10] Wikipedia: R-Tree. http://en.wikipedia.org/wiki/R-tree, last visited on 08.04.2013

[11] Douglas, G.: different projects. http://www.superliminal.com/sources/sources.htm, last visited on 08.04.2013

[12] Geuzaine, C.: gl2ps homepage. http://geuz.org/gl2ps/, last visited on 08.04.2013

[13] unknown author: VirtualDub homepage. http://www.virtualdub.org/, last visited on 08.04.2013

[14] ImageMagick Studio LLC. ImageMagick homepage. http://www.imagemagick.org/, last visited on 08.04.2013

[15] Bieker, L., Krajzewicz, D., Röckl, M., Capelle, H.: Derivation of a fast, approximating 802.11p simulation model. In: Intelligent Transport Systems Telecommunications (ITST2010), Japan (2010)

[16] Krajzewicz, D.: AnoPlib homepage. http://www.krajzewicz.de/small_vision/anop.php, last visited on 08.04.2013

[17] Wikipedia: "Display resolution". http://en.wikipedia.org/wiki/Display_resolution, last visited on 08.04.2013

[18] "Jedi787plus": Wikimedia Commons, File:Vector Video Standards4.svg. http://commons.wikimedia.org/wiki/File:Vector_Video_Standards4.svg, last visited on 08.04.2013 (2009)