Date:	2018-05-02
Author:	Zhou Huang, CarMaker Service Team Germany
Release No.:	CarMaker 6.x – 7.x

User Visualization for IPGMovie using TclGeo

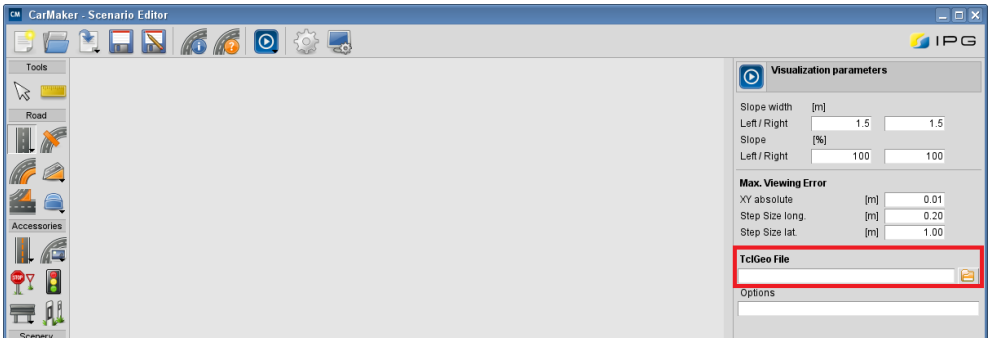
If you have any special visualization need in IPGMovie, a TclGeo script could be an option for achieving that. It offers the opportunity to draw in IPGMovie some simple geometries for static objects or display dynamic animated objects depending on parameters or movements of the vehicle or rather any other calculated quantities during the simulation.

1. How to select a TclGeo file?

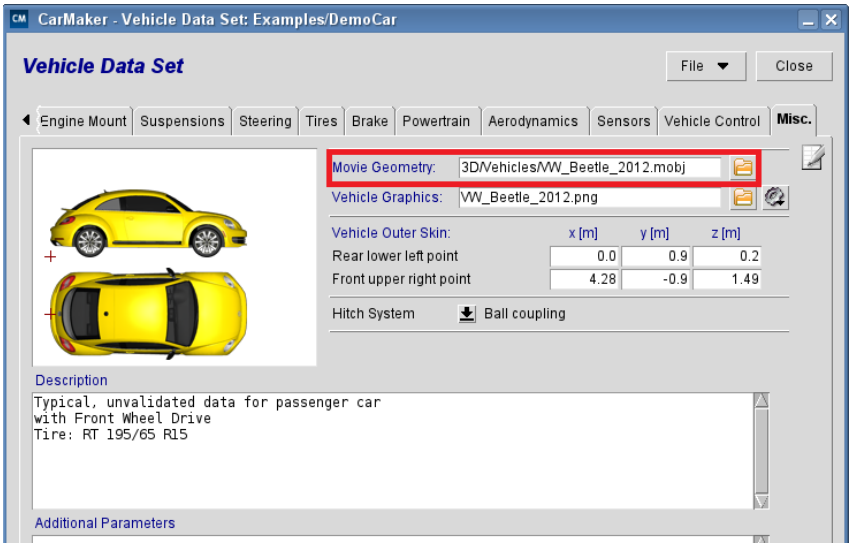
A TclGeo file could be selected on several places and will be passed to IPGMovie for evaluation. On successful evaluation, the content will be immediately visualized in the IPGMovie 3D World.

Locations to select a TclGeo file:

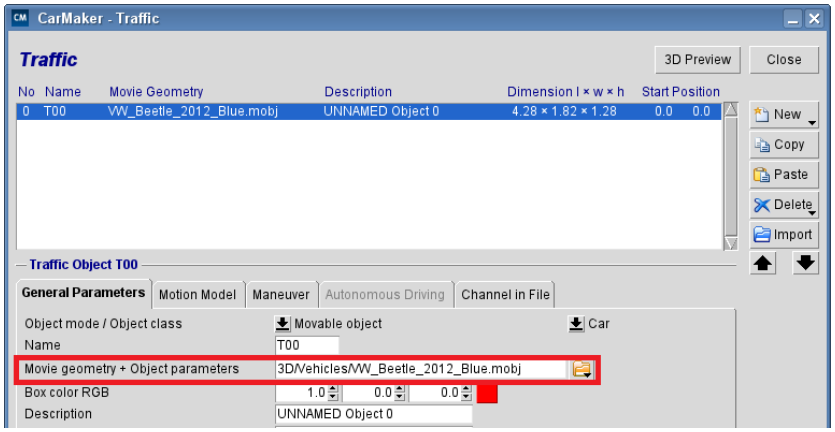
- The Scenario Editor of CarMaker. When you long click on the button of "3D Preview" in Scenario Editor, you will open the settings for visualization parameters. There is an option for selecting a TclGeo file.



- Vehicle Data Set GUI/Misc./Movie Geometry



- Traffic GUI/General Parameters/Movie geometry + Object parameters



- However, in the last two cases a TclGeo file is referenced trough another 3D geometry file.

The file formats supported by IPGMovie for geometry objects are *.obj, *.mobj, *.dae, *.kmz and *.tclgeo (script language Tcl/Tk).

It is possible to include a *.tclgeo file into a 3D geometry file, e.g. in an *.obj file, as the following shows:

```
1:  ### BEGIN IPG-MOVIE-INFO
2:  # Include YYY.tclgeo
3:  ### END IPG-MOVIE-INFO
```

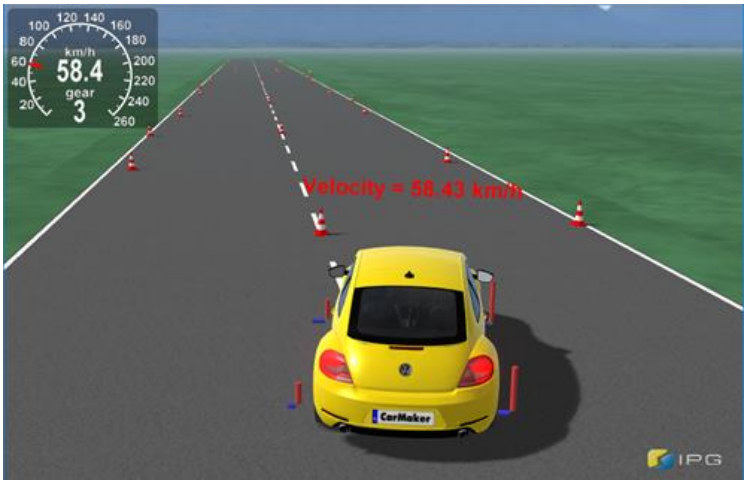
Please note that the *.tclgeo and *.obj files should be in the same folder.

Alternatively, an *.obj file can be also referenced in a *.tclgeo file by the following approaches:

```
1:  ### BEGIN IPG-MOVIE-INFO
2:  # Include XXX.obj
3:  ### END IPG-MOVIE-INFO
```

2. Application Examples

Display of Vehicle Velocity



The following TclGeo code example demonstrates how to display the current velocity over the ego vehicle in IPGMovie.

```
1:   ### BEGIN IPG-MOVIE-INFO
2:
3:   ### END IPG-MOVIE-INFO
4:   proc DrawVehicle {} {
5:       global Qu
6:       # Add your code here
7:       glDisableLighting
8:       glEnableTexture2DModulate
9:       gl color 1 0 0 1; # red
10:      set velocity [format %.2f $Qu(Vhcl.v)]
11:      OGL drawtext sans-bold "Velocity = $velocity m/s" -center \
12:      -pos 2.0 0.0 2.0 -dir 0 -.25 0 -up 0 0 .25
13:      glDisableTexture
14:      glEnableLighting
15:  }
16:
17:  return -1
```

Line 1 - 3:

The quantities used in *.tclgeo files should be subscribed at the beginning. Since the ego vehicle velocity is initially subscribed to IPGMovie, there is no need to subscribe it again.

Line 5:

In order to access the quantity of the ego vehicle velocity “Vhcl.v” that is from CarMaker Data Dictionary, it is necessary to announce the corresponding namespace “Qu”.

Line 7, 8, 13, 14:

In order to show the text in the correct color, the lighting effect should be disabled and the textured surfaces shades need to be enabled.

Line 9:

Definition of text colors. The fourth parameter defines the color intensity from 0.0 to 1.0 (full intensity). The first three parameters (RGB) for several other colors are listed in the following table:

Color	Parameter
Green	0 1 0
Cyan	0 1 1
Black	0 0 0
Blue	0 0 1
Yellow	1 1 0

Line 11, 12:

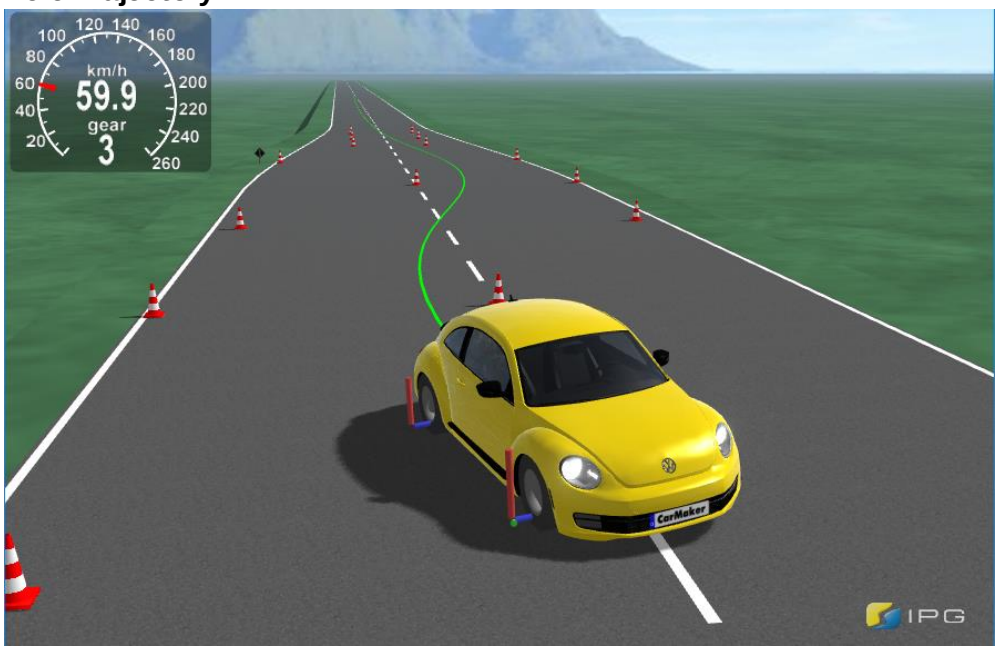
The default coordinate system of the callback “DrawVehicle” is Fr1 (information about the definition of Fr1 please see Reference Manual). The current velocity is displayed at the position of 2.0m in front of and 2.0m higher than the origin of Fr1.

Besides of the callback “DrawVehicle”, there are also other basic available callbacks (TclGeo procedures) that are automatically called for each frame in IPGMovie, see the table in appendix. For example, if something related to traffic needs to be drawn in Frame Fr0, the TclGeo procedure named DrawBackground should be called.

Line 17:

If a *.tclgeo file is not referenced trough another 3D geometry file, this line must to be added at the very end: return -1.

Draw Vehicle Trajectory



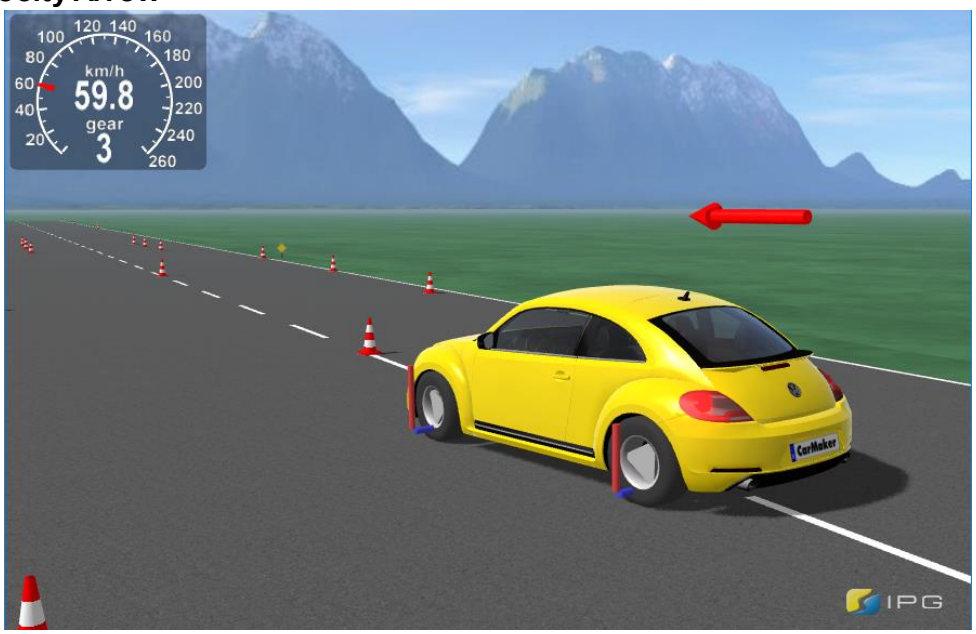
The following TclGeo code example shows how to draw the driven trajectory of ego vehicle in IPGMovie.

```
1:  ### BEGIN IPG-MOVIE-INFO
2:  # Subscribe Vhcl.PoI.x Vhcl.PoI.y Vhcl.PoI.z
3:  ### END IPG-MOVIE-INFO
4:
5:  set PoI(Data) ""
6:  set PoI(Width)  0.05
7:  set PoI(Height) -0.4
8:
9:  proc DrawStart {} {
10:
11:     global Qu
12:     variable PoI
13:     # Add new data and visualize driven trajectory
14:     if { $PoI(Data) eq "" || ![dict exists $PoI(Data) $Qu(Time)]} {
15:         dict set PoI(Data) $Qu(Time) [list $Qu(Vhcl.PoI.x) $Qu(Vhcl.PoI.y) \
16:                                     [expr {$Qu(Vhcl.PoI.z)+$PoI(Height)}] \
17:                             $Qu(Vhcl.Distance) $Qu(Vhcl.Yaw) $Qu(Vhcl.Roll)]
18:     }
19:
20:     gl pushmatrix
21:     glDisableLighting
22:     gl color 0 1 0 1; #Green
23:     gl begin quad_strip
24:
25:     foreach key_act [lsort -real [dict keys $PoI(Data)]] {
26:         set elm [dict get $PoI(Data) $key act]
27:         if {([lindex $elm 3] > 0.0 && [lindex $elm 3] <= $Qu(Vhcl.Distance))} {
28:             set val_x [expr {0.5*$PoI(Width)*sin([lindex $elm 4])}]
29:             set val_y [expr {0.5*$PoI(Width)*cos([lindex $elm 4])}]
30:             set val_z [expr {0.5*$PoI(Width)*sin([lindex $elm 5])}]
31:
32:             gl vertex [expr {[lindex $elm 0]+$val_x}] \
33:                       [expr {[lindex $elm 1]-$val_y-tan($val_z)}] \
34:                       [expr {[lindex $elm 2]-$val z}]
35:
36:             gl vertex [expr {[lindex $elm 0]-$val_x}] \
37:                       [expr {[lindex $elm 1]+$val_y-tan($val_z)}] \
38:                       [expr {[lindex $elm 2]+$val_z}]
39:         }
40:     }
41:
```

```
42:     gl end
43:     glEnableLighting
44:     gl popmatrix
45: }
46:
47: return -1
```

Line 1 - 3:
Some quantities do not need to be specifically subscribed, since IPGMovie will do it initially. But these quantities need to be subscribed manually using "#Subscribe" between the beginning and the end of "IPG-MOVIE-INFO", because they are not initially subscribed.
Line 20, 44:
They are the same as the OpenGL commands "gl pushmatrix" and "gl popmatrix". They are used for pushing the current matrix stack down by one, duplicating the current matrix and poping the current matrix stack, replacing the current matrix with the one below it on the stack.
Line 21, 43:
In order to show the text in the correct color, the lighting effect should be disabled.
Line 23:
The command "gl begin" is always used together with "gl end". The mode "quad_strip" draws a connected group of quadrilaterals. One quadrilateral is defined for each pair of vertices presented after the first pair.
Line 32 - 35:
Specifies the coordinates of vertexes.

Draw Velocity Arrow



This example shows you how to present the current velocity of ego vehicle by using an extendable dynamic arrow.

```
1:  ### BEGIN IPG-MOVIE-INFO
2:
3:  ### END IPG-MOVIE-INFO
4:
5:  proc DrawVehicle {} {
6:
7:      global Qu
8:      set length    [expr {$Qu(Vhcl.v)*3.6/75}]
9:
10:     gl pushmatrix
11:     gl translate 0 0 2.0
12:     gl rotate 90 0 1 0
```

```
13:     gl material f+b a+d 1 0 0 1; # red
14:     gl Cylinder 0.05 0.05 $length 16 1
15:     gl translate 0 0 $length
16:     gl Cylinder 0.1 0 0.25 16 1
17:     gl popmatrix
18: }
19:
20: return -1
```

Line 10, 17:
The command “gl pushmatrix” is always used together with “gl popmatrix”. They are necessary when coordinate needs to be translated or rotated.
Line 11, 12:
Translation and rotation of the current coordinate system (Fr1).
Line 13:
It defines the material on the front and back faces (f+b) of the object. It is ambient and diffuse (a+d). The color can be defined by RGBA.
Line 14:
Draws a cylinder (part of an arrow) with the length that depends on the current velocity of ego vehicle.

3. Appendix

The following table shows all the available callbacks for each frame in IPGMovie.

Callback	2D/3D	Frame	Draw
PostLoadScene			object
DrawCamera			background
DrawStart	2D	Fr0	road
DrawBackground	3D	Fr0	traffic
DrawVehicle DrawVehicleA DrawVehicleB	3D	Fr1	vehicle
DrawTrailer DrawTrailer2	3D		trailer
DrawEnd	3D		
DrawOverlay	2D		

The procedure of PostLoadScene will be called for loading some movie objects only one time at the beginning of a simulation.

Disclaimer:

Using TclGeo for IPGMovie visualization extensions needs the programming knowledge of Tcl and OpenGL.

Any added TclGeo file could possibly create unexpected visualization effects. Therefore, this functionality is not officially supported. Any reports related to the usage of TclGeo files may not be processed.