OpenDS Tutorial

Rafael Math

11/11/2019

OpenDS Tutorial

Overview

- Requirements
- Download and Getting Started
- Assets and Task Description Files
- Converting Existing Models
- Extending the Scenery (Sky, Road Signs, Weather, Traffic)
- Interaction

Requirements

Operating system	Windows, Linux, Mac OS
Memory (JVM heap size)	> 40 MB + memory for assets
CPU	> 1 GHz
Graphic card	AMD/ATI Radeon 9500, NVIDIA GeForce 5 FX, Intel GMA 4500, or better supporting OpenGL 2.0 or better
Java Runtime Environment	JRE 8 or higher

No programming skills required

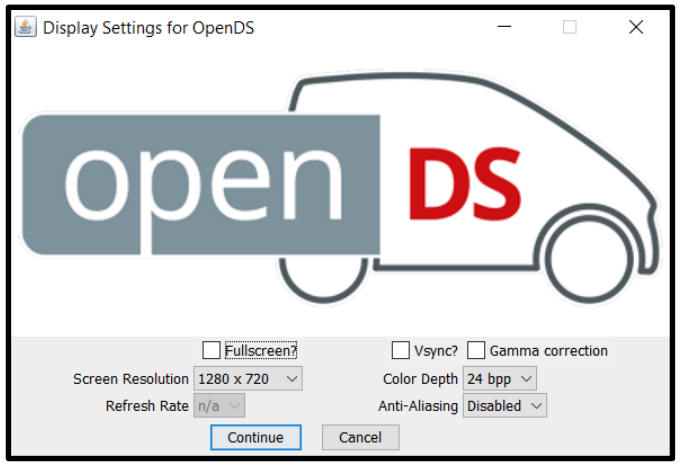
Download and Getting Started

- Download the latest version:
  https://cloud.dfki.de/owncloud/index.php/s/8yr6KozAF3ceBBY

- Unzip OpenDS to any folder where you have write access

- Run OpenDS.jar (e.g. “java -jar OpenDS.jar”)

- Select the following settings and click “Continue”:
  - Fullscreen: NO
  - Vsync: NO
  - Gamma correction: NO
  - Screen resolution: 800 x 600 (or larger)
  - Color Depth: 24 bpp
  - Refresh Rate: n/a
  - Anti-Aliasing: Disabled

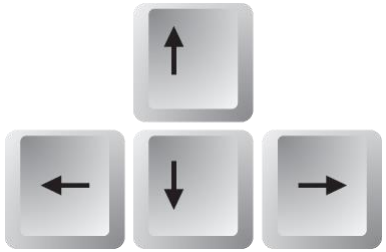


- Select Driving Task: assets/DrivingTasks/Projects/Tutorial/tutorial.xml and click “Start”

Basic Key Assignment

You can use the following keys for driving the car:

- accelerate
- accelerate backwards
- steer left
- steer right



- brake

- change camera view

- detailed key mapping

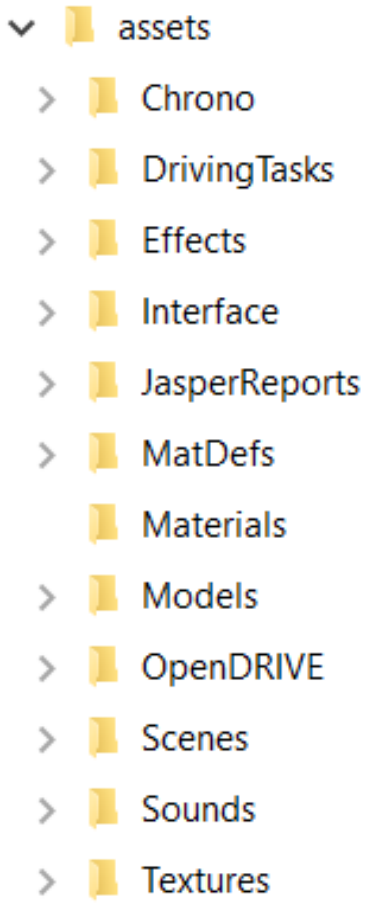
- shut down

Assets Folder

- The assets folder is the central repository for
  - driving environments (**Scenes**),
  - scene objects (**Models**),
  - audio files (**Sounds**)
  - image files (**Textures**)
  - GUI definitions (**Interface**)
  - templates for performance reports (**JasperReports**)

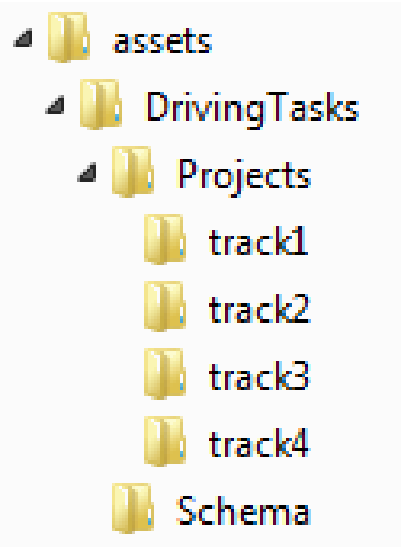
- Resources stored in the central assets folder can be accessed by multiple driving setups

- Task description files (**DrivingTasks**) are used to describe a driving setup.



Task Description Files

- The **DrivingTasks** folder consists of the two subfolders **Projects** and **Schema**:
  - **Projects** contains some sample projects which typically consist of the following XML files:
    - openDRIVE.xodr
    - scene.xml
    - scenario.xml
    - interaction.xml
    - settings.xml
    - (task.xml)
  - **Schema** contains the XML schema files for those XML files



Task Description Files

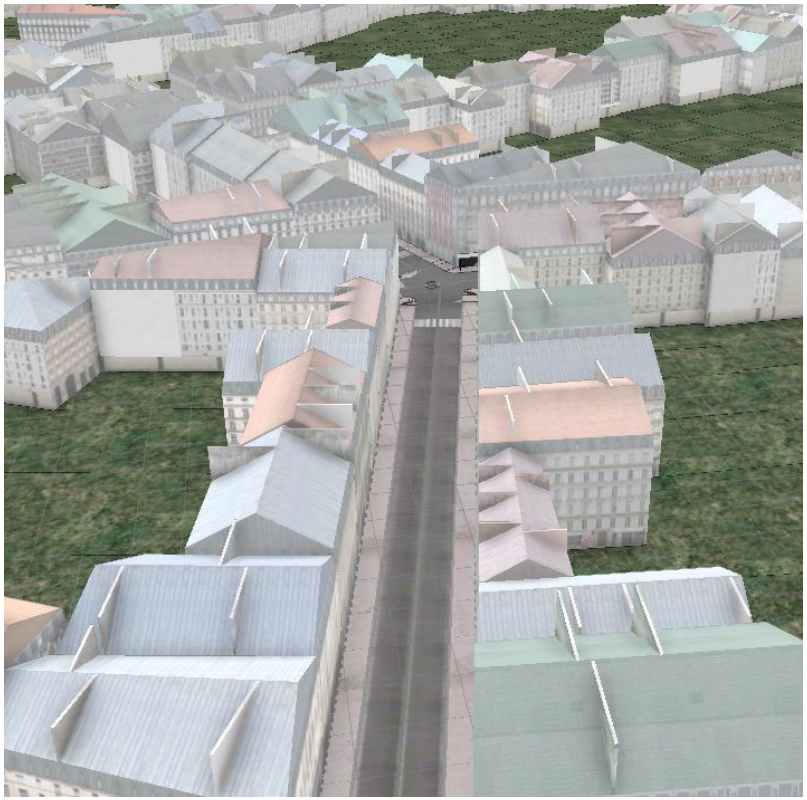
openDRIVE .xodr	scene .xml	scenario .xml	interaction .xml	settings .xml

• Road network:
  • Geometry
  • Lanes
  • Junctions
	• Road objects:
	  • Shape
	  • Translation
	  • Rotation
	  • Scale
	  • Mass
	• Environment:
	  • Sun
	  • Sky
		• Environment:
		  • Weather
		  • Driver
		  • Traffic
		  • Traffic Lights
			• Triggers:
			  • Conditions
			  • Actions
				• Camera settings
				• CAN settings
				• Mirror settings
				• Key mapping

Task 1a: Creating a Road Model

Sample Tool: **Esri CityEngine**

- Commercial tool (free 30-day trial)
- Easy road and building creation
- Export to OBJ format



Task 1b: Converting the Road Model

- Convert OBJ to OgreXML format with **Obj2OgreConverter**
  - Python script using Blender 2.49b and the following plugins:
    - Wavefront OBJ Importer (built-in)
    - dotScene Exporter: http://www.ogre3d.org/tikiwiki/Blender+dotScene+Exporter
    - Meshes Exporter: http://www.ogre3d.org/tikiwiki/Blender+Exporter
  - The following files will be created:

Multiplicity	File Extension	Description
1..*	**.mesh.xml*	Contains the mesh (polygons) of a scene object
1	**.scene*	List of all objects that will be included in the scene
1	**.material*	Assignment of materials (e.g. textures) to scene objects
0..*	**.jpg, *.gif, *.png*	Optional textures

- Advantages of OgreXML format:
  - OgreXML is the native model format of OpenDS
  - OgreXML allows to apply filters

Task 2: Adding the Model to the Scene

- You will find the exported city model in your assets folder at:
  assets/Scenes/Tutorial/

- To include the model to the scene add the following code to the <models> element of the project's scene.xml (assets/DrivingTasks/Projects/Tutorial/scene.xml):

```
<model id="City" key="Scenes/Tutorial/city.scene">
    <mass>0</mass>
    <visible>true</visible>
    <collisionShape>meshShape</collisionShape>
    <scale>
        <vector jtype="java_lang_Float" size="3">
            <entry>0.4</entry>
            <entry>0.4</entry>
            <entry>0.4</entry>
        </vector>
    </scale>
    <rotation quaternion="false">
        <vector jtype="java_lang_Float" size="3">
            <entry>0</entry>
            <entry>90</entry>
            <entry>0</entry>
        </vector>
    </rotation>
    <translation>
        <vector jtype="java_lang_Float" size="3">
            <entry>0</entry>
            <entry>0</entry>
            <entry>0</entry>
        </vector>
    </translation>
</model>
```

XML snippet available in the tutorial package: tutorial_02.xml

Scale model to 40 % of original size

Rotate model 90° around up axis

Set model to position (0,0,0)

Task 3: Optimize Start Properties

The **startProperties.properties** file can be used to:

- Skip the resolution setup screen (`showsettingsscreen=true`)
- Set the default resolution (`width`, `height`)
- Set the default driving task (e.g. `drivingtask=assets/DrivingTasks/Projects/track1/track1.xml`)

The default driving task can furthermore be provided as **command line arguments**, e.g:

```
java –jar OpenDS.jar assets/DrivingTasks/Projects/track4/track4.xml
```

Task 4: Setting the Sky Texture

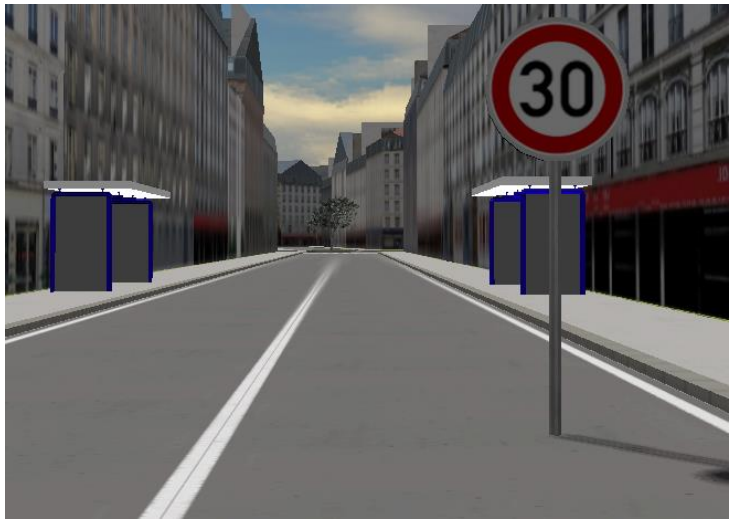
- The sky is a large textured box that cannot be reached by the driving car
- The path to a given texture can be set individually in each project's scene.xml file, e.g. assets/DrivingTasks/Projects/Tutorial/scene.xml
- Change the path of the <skyTexture> element to "Textures/Sky/Bright/mountain.dds":

Textures/Sky/Bright/BrightSky.dds

Textures/Sky/Bright/mountain.dds

Adding Road Objects

Recommended tool: **ObjectLocator**



For example a speed limit sign can be “dropped” at any position

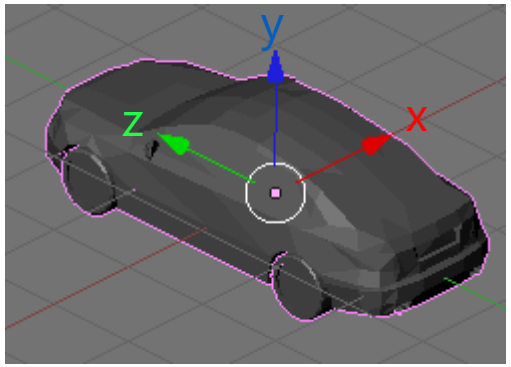
- A list of road objects must be provided (e.g. *trafficObjects.txt*)
- Objects from the list can be selected and placed while driving
- Objects will not be added permanently to the scene
  - → XML representation will be stored in the latest subfolder of *analyzerData*
  - → User may decide which road objects to add permanently

Task 5a: Adding a Road Sign

1. Add road sign to the inventory list of the ObjectLocator (trafficObjects.txt):

```
TrafficLight_;  Models/TrafficLight/trafficlight.scene;  1,-1,-5;  0,0,0;  0.7,0.7,0.7;
```

translation rotation scale



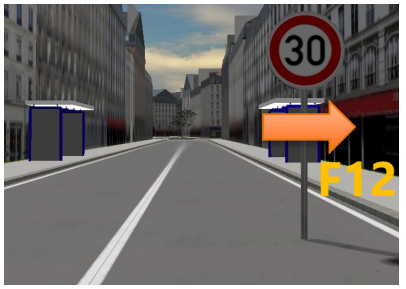
axis	value < 0	value > 0
x	left	right
y	down	up
z	forth	back

2. Start OpenDS with the ObjectLocator enabled and a pointer to *trafficObjects.txt* (settings.xml):

```
<objectLocator>
      <enable>true</enable>
      <fileName>trafficObjects.txt</fileName>
</objectLocator>
```

Task 5a: Adding a Road Sign

3. Drive to the position where to place a road sign



4. Press **F12** until the desired road sign is shown

5. Adjust the rotation of the road sign
   - **F7** / **F8** fast rotation clockwise / counter-clockwise
   - **F9** / **F10** slow rotation clockwise / counter-clockwise

6. Press **F11** to “drop” the road sign at its designated position

7. Repeat procedure for further road signs

Task 5a: Adding a Road Sign

- The output of the ObjectLocator might look like the following XML example

- To include e.g. the speed limit sign to the scene add the following code to the <models> element of the scene.xml (assets/DrivingTasks/Projects/Tutorial/scene.xml):

```
<model id="speedLimit50_1" key="Models/RoadSigns/
     speedLimits/speedLimit50/speedLimit50.scene" ref="">
     <mass>0</mass>
     <visible>true</visible>
     <collisionShape>meshShape</collisionShape>
     <scale>
          <vector jtype="java_lang_Float" size="3">
               <entry>1.0</entry>
               <entry>1.0</entry>
               <entry>1.0</entry>
          </vector>
     </scale>
     <rotation quaternion="false">
          <vector jtype="java_lang_Float" size="3">
               <entry>0</entry>
               <entry>3.37</entry>
               <entry>0</entry>
          </vector>
     </rotation>
     <translation>
          <vector jtype="java_lang_Float" size="3">
               <entry>-67.57</entry>
               <entry>-0.27</entry>
               <entry>19.67</entry>
          </vector>
     </translation>
</model>
```

XML snippet: tutorial_05a.xml

100 % of original size

Rotate model 3.37° around up axis

Set model to position (-67.57 , -0.27 , 19.67)

Task 5b: Adding more Road Signs

- Add the model definition of `tutorial_05b.xml` to the <models> element of the scene.xml in the same way

- When running OpenDS, the result should look like:

Weather Conditions

- Simple weather conditions can be set in the <weather> element of the project's scenario.xml (assets/DrivingTasks/Projects/Tutorial/scenario.xml):

```
<environment>
        <weather>
                <snowingPercentage>100</snowingPercentage>
                <rainingPercentage>0</rainingPercentage>
                <fogPercentage>50</fogPercentage>
        </weather>
</environment>
```

- Simulation of snow and rain by particle emitter
  - Particle emitter is attached to the driving car
  - resource consuming → set to "-1" if not used
- Simulation of fog is a global effect

Task 6a: Traffic

- Setting up a computer-controlled traffic requires:
  - a vehicle (or pedestrian) model with parameters like mass, acceleration, deceleration,
  - a list of waypoints,
  - a list of segments.

- Add the following XML code to the project's scenario.xml (assets/DrivingTasks/Projects/Tutorial/scenario.xml):

Example car to <traffic> element

```
<vehicle id="car1">
    <modelPath>Models/Cars/drivingCars/bmw1/Car.j3o</modelPath>
    <mass>800</mass>
    <acceleration>3.3</acceleration>
    <decelerationBrake>8.7</decelerationBrake>
    <decelerationFreeWheel>2.0</decelerationFreeWheel>
    <engineOn>true</engineOn>
    <minDistanceFromPath>2.5</minDistanceFromPath>
    <maxDistanceFromPath>3.5</maxDistanceFromPath>
    <startWayPoint>WayPoint_74</startWayPoint>
</vehicle>
```

XML snippet: tutorial_06a.xml

Example waypoint list to <road> element

```
<wayPoints debug="true">
    <wayPoint id="WayPoint_74">
        <translation>
            <vector jtype="java_lang_Float" size="3">
                <entry>-74.81913</entry>
                <entry>0.1243466</entry>
                <entry>-54.813656</entry>
            </vector>
        </translation>
    </wayPoint>
</wayPoints>
```

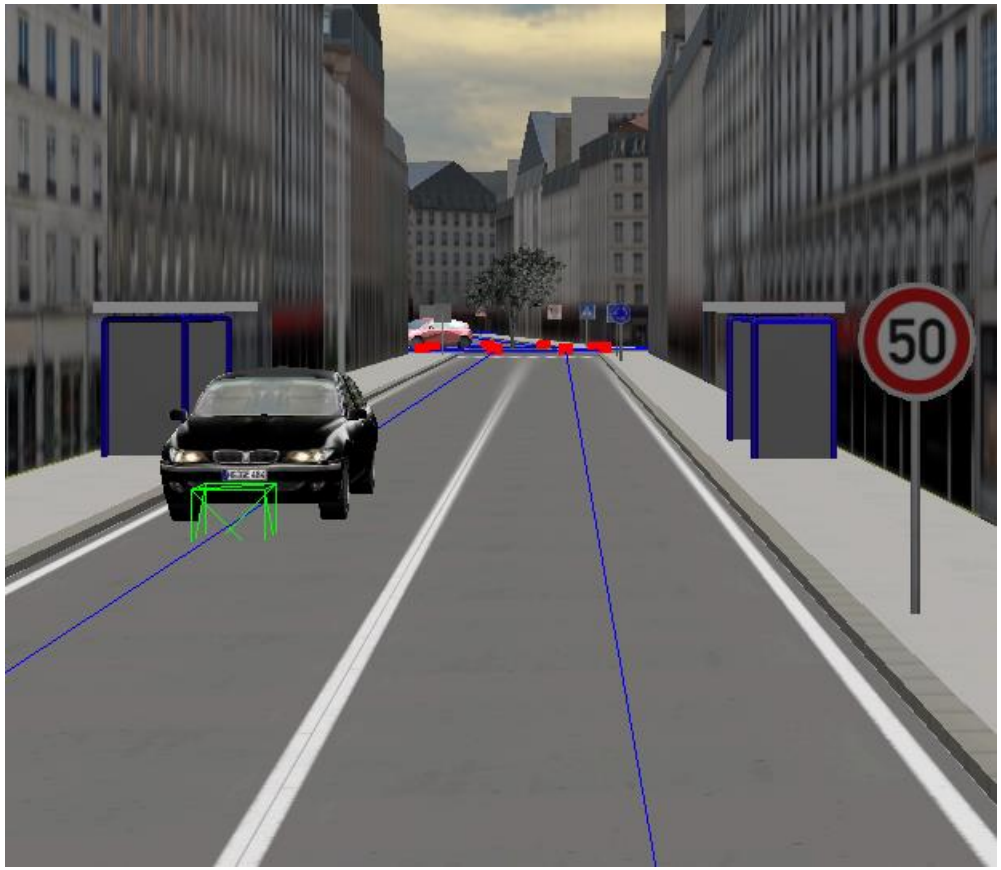
Example segment list to <road> element

```
<segments debug="true">
    <segment id="segment74_32">
        <from>WayPoint_74</from>
        <to>WayPoint_32</to>
        <speed>50</speed>
        <jump>false</jump>
        <probability>1.0</probability>
    </segment>
</segments>
```

Task 6b: Adding more Traffic

- Add the traffic definition of `tutorial_06b.xml` to the <traffic> and <road> element of the scenario.xml in the same way

- When running OpenDS, the result should look like:



Task 7a: Simple Geometries

- The following geometries can be defined by parameters in the scene.xml without model files:
  - **box**: height, width, depth
  - **sphere**: radius, number of axial and radial samples
  - **cylinder**: height, radius, number of axial and radial samples

- Place three identical boxes in the scene:
  1. Add the following geometry definition to the <geometries> element of the scene.xml

```
<box id="box">
   <width>1</width>
   <depth>1</depth>
   <height>0.1</height>
</box>
```

tutorial_07a.xml

  2. Add three model definitions to the <models> element of the scene.xml referencing ("**ref**") the geometry "box"

Task 7b: Simple Geometries

2. Add three model definitions to the <models> element of the scene.xml referencing ("**ref**") the geometry "box"

```
<model id="redBox" key="" ref="box">
[…]
</model>

<model id="blueBox" key="" ref="box">
[…]
</model>

<model id="greenBox" key="" ref="box">
[…]
</models>
```

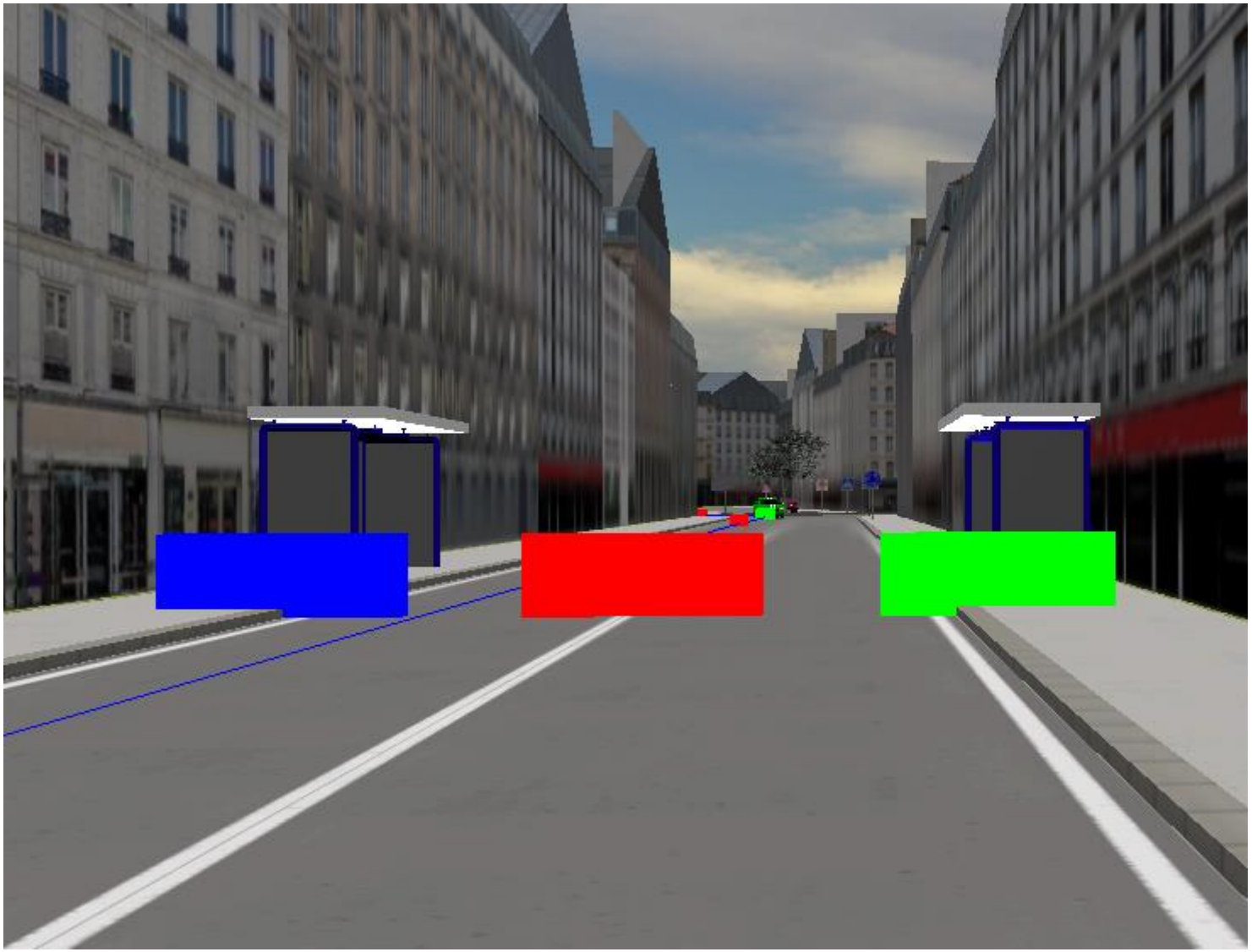
For complete code see: tutorial_07b.xml

3. As the boxes have no material information (e.g. texture, …), set the color manually:

```
<material>
     <color>
          <vector jtype="java_lang_Float" size="4">
               <entry>1</entry>
               <entry>0</entry>
               <entry>0</entry>
               <entry>1</entry>
          </vector>
     </color>
</material>
```

RGBA color value: (1,0,0,1) → red

Simple Geometries



Interaction

- Events – defined in the project's interaction.xml – can be triggered:
  - on collision with a specified scene object,
  - on key, button, or pedal press

- An event trigger consists of:
  - a condition and
  - a list of activities which will be executed when the condition is met

```
<triggers>
    <trigger id="collisionWithRedBox">
        <activities>
            <activity id="moveCar1">
                <action id="moveTraffic" delay="0" repeat="0">
                    <parameter name="trafficObjectID" value="car1" />
                    <parameter name="wayPointID" value="WayPoint_74" />
                </action>
            </activity>
        </activities>
        <condition>
            <collideWith>
                <modelID>redBox</modelID>
            </collideWith>
        </condition>
    </trigger>
</triggers>
```

activity list

condition

This example will move the computer-controlled vehicle "car1" to "Waypoint_6" when the driver's car collides with the model "redBox"

Interaction

- Events – defined in the project's interaction.xml – can be triggered:
  - on collision with a specified scene object,
  - on key, button, or pedal press

- An event trigger consists of:
  - a condition and
  - a list of activities which will be executed when t

```
<triggers>
    <trigger id="collisionWithRedBox">
        <activities>
            <activity id="moveCar1">
                <action id="moveTraffic" delay="0" repeat
                    <parameter name="trafficObjectID" va
                    <parameter name="wayPointID" value="
                </action>
            </activity>
        </activities>
        <condition>
            <collideWith>
                <modelID>redBox</modelID>
            </collideWith>
        </condition>
    </trigger>
</triggers>
```

Some implemented events:

- manipulateObject
- manipulatePicture
- pauseSimulation
- startRecording
- stopRecording
- resetCar
- **moveTraffic**
- startPresentationTask
- setCurrentSpeedLimit
- measureTimeUntilBrake
- measureTimeUntilSpeedChange
- playSound
- sendMessage
- requestGreenTrafficLight
- startReactionMeasurement
- openInstructionsScreen
- …

This example will move the computer-controlled vehicle "car1" to "Waypoint_6" when the driver's car collides with the model "redBox"

Task 8a: Collision Interaction

- Set up a collision interaction for the three boxes as follows:
  1. Define three activities
  2. Define three triggers, one for each box
  3. For each trigger, add a collision condition and assign one of the activities

- Add the following XML code to the <activities> element the project's interaction.xml (assets/DrivingTasks/Projects/Tutorial/interaction.xml)

```
<activity id="movePedestrianCrossing_1">
     <action id="manipulateObject" delay="0" repeat="0">
          [parameter list …]
     </action>
</activity>
<activity id="movePedestrianCrossing_1Back">
     <action id="manipulateObject" delay="0" repeat="0">
          [parameter list …]
     </action>
</activity>
<activity id="moveCar1">
     <action id="moveTraffic" delay="0" repeat="0">
          [parameter list …]
     </action>
</activity>
```

manipulating a scene object

manipulating a scene object

moving traffic

For complete code see: tutorial_08a.xml

Task 8b: Collision Interaction

- Add the following XML code to the <triggers> element the project's interaction.xml (assets/DrivingTasks/Projects/Tutorial/interaction.xml)

```
<trigger id="collisionWithRedBox">
    <activities>
        <activity ref="moveCar1"/>
    </activities>
    <condition>
        <collideWith>
            <modelID>redBox</modelID>
        </collideWith>
    </condition>
</trigger>

<trigger id="collisionWithBlueBox">
    <activities>
        <activity ref="movePedestrianCrossing_1"/>
    </activities>
    <condition>
        <collideWith>
            <modelID>blueBox</modelID>
        </collideWith>
 </condition>
</trigger>
```

on collision with "redBox", activity "moveCar1" will be executed

on collision with "blueBox", activity "movePedestrianCrossing_1" will be executed

For complete code see: tutorial_08b.xml

Note: local definition and reference to global definition allowed!

End of Tutorial

Questions?

Thank You For Your Attention