VTD FAQ ImageGenerator

Back to top-level FAQ

Sky Model

> *see VIG Sky Model.*

OS Optimizations

Operating systems try to provide user friendliness with a minimum of hardware's power consumption. However, several features do not fit into real time rendering requirements.
The linked document is a suggestion how to optimize your OS for real time rendering using the VIG.

> *see VIG OS Optimization.*

Extensions

The availability of the extensions within VTD depends on the packages that have been licensed to the respective user. If you don't have a license you also won't have to worry about the contents of the following chapters.

- **Real-Time Ray-Tracing (via Optix)**

> *see VIG-OptiX SDK*

- **Plugin Example**

> *see Plugin Example*

- **IG plugin development**

> *see VTD IG plugins*

Task Configuration

How can I run without IG?

The IG does not necessarily have to run if you don't need a graphical output (e.g. while just creating scenarios etc.). In order to disable the IG, please perform the following steps:

- In the TaskControl configuration file (typically: *Data/Setups/Current/Config/TaskControl/taskControl.xml*) adjust the settings:

```
<TaskControl>
    ...
    <ImageGenerator  ignore="true"
                     imgPortConnect="false"
                     ctrlPortConnect="false"
                     autoConfig="false"
                     autoRestart="false"
                     ... />
    <Sync source="intern" .../>
</TaskControl>
```

  Concerning the sync source, anything but `extern` may be used.

- De-register the IG from the components which have to be loaded: edit the file *Data/Setups/Current/Config/Simulation/simSettings.cfg* and disable the entry

```
set TASK_LIST = $TASK_LIST igCtr
```

How do I assign IG threads to dedicated CPUs?

The load of the IG on the system may be very high. Therefore, it makes sense to distribute the IG's threads and assign them to dedicated CPUs. By this, context switching will also be avoided and the IG will run at maximum performance.

The CPUs of the IG's threads are defined in the file

`Data/Setups/Current/Config/ImageGenerator/AutoCfgDisplay.cfg`.

The corresponding attributes are

`drawThreadCPUAffinity="1"` and
`appCullThreadCPUAffinity="0"`.

You may change these settings manually. However, if your system is in auto configuration mode (see `Data/Setups/Current/Config/TaskControl/taskControl.xml` and check `<ImageGenerator ....autoConfig="true".../>`) these settings may be overwritten. In order to assign your CPUs also in auto configuration mode, perform the following steps:

- create a directory `Data/Setups/Current/Scripts`
- copy `Data/Setups/Common/Scripts/configureDisplay.sh` to the new directory
- modify the attributes `drawThreadCPUAffinity="1"` and `appCullThreadCPUAffinity="0"` in this script

Now, each time the IG is reconfigured, your specified CPUs will be assigned to the processes.

One more item:

If you're using the HDR version of the Image Generator, there is an additional CPU assignment in

`Data/Setups/Current/Config/ImageGenerator/IGbase.xml`.

Look for

```
<TAKATA ... convertLightmapThreadCPUAffinity="0" .../>
```

and set the CPU value accordingly.

How can I configure the procedural grass generation (i.e. vegetation renderer)?

- in `Data/Setups/Current/Config/ImageGenerator/AutoCfg.xml`, add the line
  `<Include file="IGLinks/Vig/data/Vegetation.xml" />`

NOTE: this line must be inserted somewhere AFTER the one referring to `IGbase.xml`

- in `Data/Setups/Current/Config/ImageGenerator/IGbase.xml` add the plug-in
  `<Plugin file="VegetationRenderer" />`

- also in `Data/Setups/Current/Config/ImageGenerator/IGbase.xml` extend the file path by
  `<FilePath path="IGLinks/Vig/data/VegetationRenderer" />`

Now you'll have even greener IT!

How to (de-)activate components in *IGbase.xml*

The current version of the IG uses strict XML syntax, so that disabling of components must be performed by using regular XML comments (_<!-- -->. Previous "tricks" by simply invalidating the names of tags will no longer work.

How can I turn off or customize automatic display configuration?

You may turn off automatic display configuration by providing your own configuration script:

- create a directory `Data/Setups/Current/Scripts`
- copy `Data/Setups/Common/Scripts/configureDisplay.sh` to the new directory

If this script is empty or its first command is `exit`, then no display configuration will be executed at all. Otherwise, you are free to configure the parameters that you want to have customized.

How can I run multiple image generators?

VTD 2.0

Running multiple image generators means that multiple instances of the `vigcar` process have to run on one or more computers. Here's a recipe for the configuration of a 2-channel system on one computer:

1) Provide the process configuration for each instance of the image generator in the file `Data/Setups/Current/Config/SimServer/simServer.xml`:

```
<Process
      group="igGroup"
      name="igCtr"
      auto="false"
      explicitLoad="false"
      path="$VI_CORE_DIR/ImageGenerator/bin"
      executable="vigcar"
      cmdline="$VI_CURRENT_SETUP/Config/ImageGenerator/AutoCfg.xml -m 0x01"
      useXterm="true"
      xtermOptions="-fg Black -bg LightGoldenRod -geometry 80x10+508+163"
      affinitymask="0xFF"
      schedPolicy="SCHED_RR"
      schedPriority="20"
      workDir="$VI_CURRENT_SETUP/Bin">
      <EnvVar name="LD_LIBRARY_PATH"      val="$VI_CORE_DIR/ImageGenerator/bin:$LD_LIBRARY_PATH"   />
    </Process>

    <Process
      group="igGroup"
      name="igCtr2"
      auto="false"
```

```
        explicitLoad="false"
        path="$VI_CORE_DIR/ImageGenerator/bin"
        executable="vigcar"
        cmdline="$VI_CURRENT_SETUP/Config/ImageGenerator/AutoCfg.xml -m 0x02"
        useXterm="true"
        xtermOptions="-fg Black -bg LightGoldenRod -geometry 80x10+508+163"
        affinitymask="0xFF"
        schedPolicy="SCHED_RR"
        schedPriority="20"
        workDir="$VI_CURRENT_SETUP/Bin">
        <EnvVar name="LD_LIBRARY_PATH"      val="$VI_CORE_DIR/ImageGenerator/bin:$LD_LIBRARY_PATH"   />
      </Process>
```

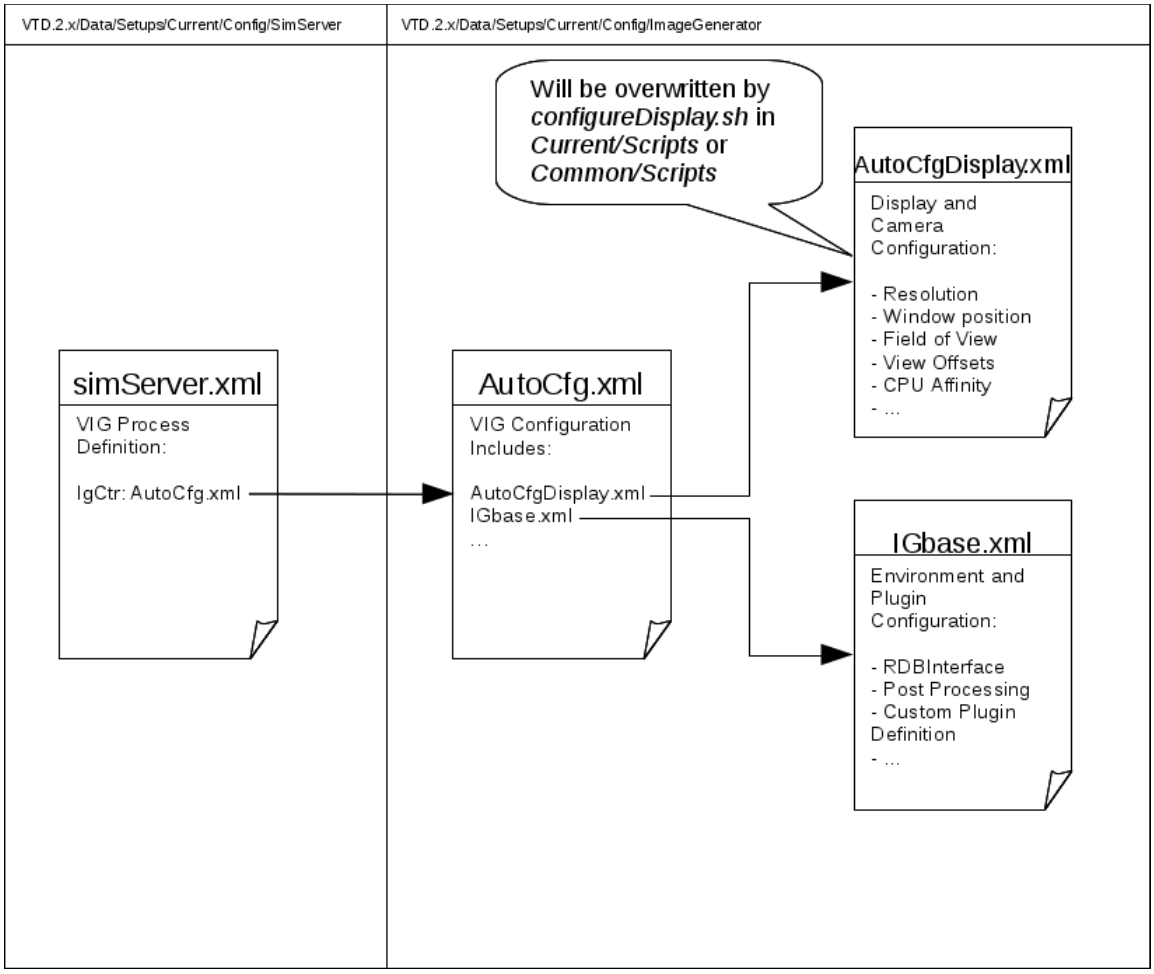
2) Configure the individual channels. This can be done at runtime using the Camera SCP message or using individual configuration files

2.1) Runtime: Send a Camera SCP message with channel mask

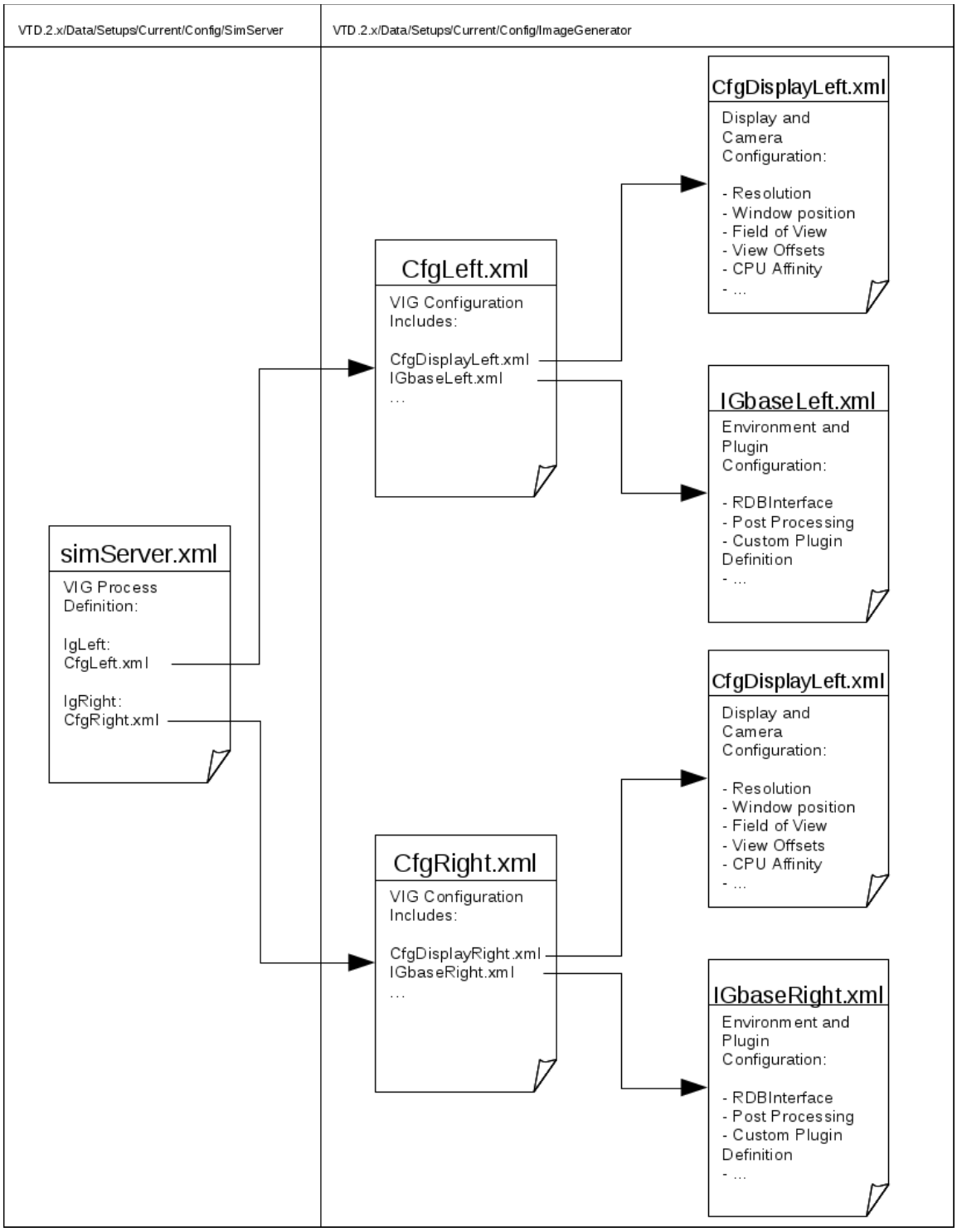
```
<Camera><Offset dx="2" dy="3" dz="0" dhDeg="0" dpDeg="0" drDeg="0" /><Set channel="0x2" /></Camera>
```

2.2) Write the configuration files for each channel

Default Configuration:



Configuration with multiple channels:



- Copy the `Setups/Current/Config/ImageGenerator/AutoCfg.xml` to create `CfgRight.xml` and `CfgLeft.xml` in the same folder

```
Data/Setups/Current/Config/ImageGenerator/
     AutoCfg.xml
     CfgRight.xml
     CfgLeft.xml
```

- Change both uses of `Setups/Current/Config/ImageGenerator/AutoCfg.xml` in `Data/Setups/Current/Config/SimServer/simServer.xml` to use `CfgRight.xml` and `CfgLeft.xml`
- Copy the `Setups/Current/Config/ImageGenerator/AutoCfgDisplay.xml` to create `CfgDisplayRight.xml` and `CfgDisplayLeft.xml` in the same folder
- Change both `CfgRight.xml` and `CfgLeft.xml` to refer to `CfgDisplayRight.xml` and `CfgDisplayLeft.xml` by Replacing their reference to the

AutoCfgDisplay.xml

```
<Include file="IGLinks/Setup/Config/ImageGenerator/AutoCfgDisplay.xml" />
```

- Change ignoreSetProjectionMessages to "1" in `Setups/Current/Config/ImageGenerator/IGbase.xml"`

```
<Components> ... <TAKATA  ...    ignoreSetProjectionMessages="1" ... >
```

- Change the offsetHPR and offsetXYZ values in `CfgDisplayRight.xml` and `CfgDisplayLeft.xml` to the expected offset

```
<SymmetricPerspectiveAngles  fovX="45.00" fovY="30.00" near="1" far="1500" offsetHPR="0 0 0" offsetXYZ="0 0.0 0"/>
```

This configuration will not receive Runtime changes for the camera projection as the SetProjection message is ignored. Changes to the window size and position from the VtGui are also ignored as they only affect the AutoCfgDisplay.xml by default.

VTD 1.4

Running multiple image generators means that multiple instances of the `vigcar` process have to run on one or more computers. Here's a recipe for the configuration of a 3-channel system:

1) Create the configuration files of the individual channels (see next topic about the configuration of viewing angles)

2) Provide the start commands for each instance of the image generator in the file `Data/Setups/Current/Config/Simulation/taskSettings`:

```
#
#
# ......... IG LEFT CHANNEL
#
if { test $TASK_TYPE = "igLeft" } then

  set BIN_NAME      = vigcar
  set OBJ_DIR       = ( $VI_CORE_DIR/ImageGenerator/bin )
  set WORK_DIR      = ( . )
  set CMD_ARGS      = ( $VI_SETUP_DIR/Config/ImageGenerator/CfgLeft.xml notrigger )
  set START_CMD     = ( source $ENV_SETTINGS ; cd $WORK_DIR ; $OBJ_DIR/$BIN_NAME $CMD_ARGS $RECORD_CMD ; sleep $XTERM_LIFETIME
  set USE_XTERM     = true
  set XTERM_OPTIONS = ( $XTERM_OPTIONS_COMMON -bg LightGoldenRod -geometry 80x10+508+163 -title IMAGEGENERATOR )
set REMOTE_HOST = "vtd-visual02"
set REMOTE_ARGS = "setenv DISPLAY :0.0 ; cd VTD/Data/Setups/Current/Bin ; "
set REMOTE_USER = "vtd"
endif
#
#
# ......... IG CENTER CHANNEL
#
if { test $TASK_TYPE = "igCtr" } then

  set BIN_NAME      = vigcar
  set OBJ_DIR       = ( $VI_CORE_DIR/ImageGenerator/bin )
  set WORK_DIR      = ( . )
  set CMD_ARGS      = ( $VI_SETUP_DIR/Config/ImageGenerator/CfgCtr.xml )
  set START_CMD     = ( source $ENV_SETTINGS ; cd $WORK_DIR ; $OBJ_DIR/$BIN_NAME $CMD_ARGS $RECORD_CMD ; sleep $XTERM_LIFETIME
  set USE_XTERM     = true
  set XTERM_OPTIONS = ( $XTERM_OPTIONS_COMMON -bg LightGoldenRod -geometry 80x10+508+163 -title IMAGEGENERATOR )
set REMOTE_HOST = "vtd-visual04"
set REMOTE_ARGS = "setenv DISPLAY :0.0 ; cd VTD/Data/Setups/Current/Bin ; "
set REMOTE_USER = "vtd"
endif
#
# .
#........ IG RIGHT CHANNEL
#
if { test $TASK_TYPE = "igRight" } then

  set BIN_NAME      = vigcar
  set OBJ_DIR       = ( $VI_CORE_DIR/ImageGenerator/bin )
  set WORK_DIR      = ( . )
  set CMD_ARGS      = ( $VI_SETUP_DIR/Config/ImageGenerator/CfgRight.xml  notrigger)
  set START_CMD     = ( source $ENV_SETTINGS ; cd $WORK_DIR ; $OBJ_DIR/$BIN_NAME $CMD_ARGS $RECORD_CMD ; sleep $XTERM_LIFETIME
  set USE_XTERM     = true
  set XTERM_OPTIONS = ( $XTERM_OPTIONS_COMMON -bg LightGoldenRod -geometry 80x10+508+163 -title IMAGEGENERATOR )
set REMOTE_HOST = "vtd-visual06"
set REMOTE_ARGS = "setenv DISPLAY :0.0 ; cd VTD/Data/Setups/Current/Bin ; "
set REMOTE_USER = "vtd"
endif
```

Important note: In order to not confuse the taskControl, there should be only one image generator in the entire simulation which is providing a trigger signal (here: center channel). For all other instances, append the argument `notrigger` to the command line.

In the above example it is assumed that each IG is running on an individual computer. If you are running more than one instance of `vigcar` on the same computer, make sure that all instances which are started **after** the first one have their task settings extended by the following command line

```
set STOP_CMD     = ( )
```

Otherwise, starting the last instance of `vigcar` on the computer will terminate all previously started ones.

Another important note: the remote computer must be accessible via password-less login from the host computer. In order to achieve this via ssh, perform the following steps

1. in a shell on the host, type `ssh-keygen -t rsa`
2. add the line in `~/.ssh/id_rsa.pub` (on host) to the file `~/.ssh/authorized_keys` on the remote machine (if the file does not exist, create it and add the contents of the source file afterwards)
3. in a shell on the host, type `ssh remoteUser@remoteHost` (replace `remoteUser` and `remoteHost` with the applicable names)
4. you may be asked to confirm that the remote host is added to the list of known hosts on your host
5. after you have logged in successfully, type `exit`
6. type the login command again and check that you can login directly without any further questions (this step is obsolete if step 3 succeeded at the first try)

3) Register the additional instances of `vigcar` in `Data/Setups/Current/Config/Simulation/simSettings.cfg`, e.g.

```
:
set TASK_LIST = "$TASK_LIST igCtr"
set TASK_LIST = "$TASK_LIST igLeft"
set TASK_LIST = "$TASK_LIST igRight"
:
```

4) For automatic restart of the image generators by the taskControl (e.g. if the database file has changed), provide a corresponding start script.

- Copy `startTaskIG.sh` from `Data/Setups/Common/Scripts` to `Data/Setups/Current/Scripts` (create this directory first, if necessary)
- Add the start commands for all IGs, e.g.

```
#!/bin/tcsh
#
# start the Image Generator only
# (c) 2009 by VIRES Simulationstechnologie GmbH
# last modification: 20.10.2009 by M. Dupuis
#
./startTasks igCtr
./startTasks igCtrLeft
./startTasks igCtrRight
```

5) Turn off interpretation of projection messages in the IG

Edit `Data/Setups/Current/Config/ImageGenerator/IGbase.xml`:

```
<TAKATA name="TAKATA"
        :
               ignoreSetProjectionMessages="1"
               :
               :>
```

6) Make sure your remote IGs connect to the host running the taskControl

Edit `Data/Setups/Current/Config/ImageGenerator/IGbase.xml` and replace *aaa.bbb.ccc.ddd* with the address of your host:

```
<TAKATA name="TAKATA"
        :
           taskControlServerAddress="aaa.bbb.ccc.ddd"
        :
       imageTransferServerAddress="aaa.bbb.ccc.ddd"               :
               :>
```

How can I set the threading model?

NOTE: these settings are **for test purposes only**. In no case shall the user change the threading model in regular installations. Instability may result from this action!!!

Now for the content:

The threading model may be specified in the display configuration file (e.g. `AutoCfgDisplay.xml`) as attribute `threadingModel` of the `<Graphics>` tag.

Valid values are: * CullDrawThreadPerContext * CullThreadPerCameraDrawThreadPerContext * SingleThreaded * DrawThreadPerContext (default)

In addition, the threading model may also be modified at run-time by pressing the key *m* while focusing the IG output window.

How can I turn off shadows?

availability: VTD 2.0.x

It might be necessary for some applications to have no shadows at all; this may either be achieved by selecting the *overcast* sky state (which leaves some ambient shadows under the cars) or by explicitly modifying the shadow caster.

Please do the following:

- edit the file `Data/Setups/Current/Config/ImageGenerator/AutoCfg.xml` (or your local equivalent) and replace the reference to `Shadows.xml` with a

local file (here: `NoShadows.xml`)

```
<!-- vIG configuration file, 27.04.2016 by M. Dupuis -->

<IGconfig>
  <Includes>
    <Include file="IGLinks/Setup/Config/ImageGenerator/NoShadows.xml" />
    <Include file="IGLinks/Distro/Config/ImageGenerator/IGConfig.xml" />
    <Include file="IGLinks/Vig/data/Sky/HighContrastSky.xml" />
    <Include file="IGLinks/Distro/Config/ImageGenerator/MaterialsHCS.xml" />
    <Include file="IGLinks/Distro/Config/ImageGenerator/SymbolsStd.xml" />
    <Include file="IGLinks/Distro/Config/ImageGenerator/LightSrcStd.xml" />
    <Include file="IGLinks/Setup/Config/ImageGenerator/AutoCfgDisplay.xml" />
    <Include file="IGLinks/Setup/Config/ImageGenerator/AutoCfgDatabase.xml" />
  </Includes>
</IGconfig>
```

- provide the file `Data/Setups/Current/Config/ImageGenerator/NoShadows.xml` with the following contents:

```
<IGconfig>
  <Components>
      <Shadows name="MyShadows" enableDistanceFadeIn="1" shadowTextureSize="2048" useDebugShader="0"
                polyOffsetUnit="2" polyOffsetFactor="5" filterType="PCF4x4" enableMipmaping="false"
                multisampleSamples="0" multisampleColorSamples="0">

         <Technique name="LISPSM" minLightMargin="180" maxFarPlane="200" />
        <DisableOnTextures>
            <texture name="Street" />
        </DisableOnTextures>
        <ExcludeFromCastersByTexture>
                  <texture name="*"/>
        </ExcludeFromCastersByTexture>
    </Shadows>
  </Components>
</IGconfig>
```

This file will disable all static shadow casters (of the environment) and prevent all vehicles from casting a shadow on polygons with textures containing *Street* in their name; you may add entries as applicable to your case

Channel Configuration

Each IG has a separate file for changing the display configuration parameters. It could be:

- AutoCfgDisplay.xml: default file, which will be overwritten by scripts (change the name and link in (Auto)Cfg.xml for constant values)
- CfgDisplay.xml or similiar

```
<IGconfig>
   <SystemConfig>
     <Graphics smallFeatureCullPixelSize="3.9"  lodscale="0.9" gamma="1 1 1"  enableCursor="0" enableDatabasePagerThread="0" d
       <RenderSurface name="mainRS" x="0" y="0" width="1920" height="1080" depthBits="24" sampleBuffers="0" samples="0" border
       <Camera name="cam1" renderSurface="mainRS" viewPortX="0" viewPortY="0" viewPortWidth="1920" viewPortHeight="1080">
         <SymmetricPerspectiveAngles  fovX="60.00" fovY="33.75" near="1" far="1500" offsetHPR="0 0 0" offsetXYZ="0 0.0 0"/>
       </Camera>
     </Graphics>
   </SystemConfig>
</IGconfig>
```

The parameters can be configured as followed:

Parameter	Tag Name	Attribute	Comment
window width	RenderSurface	width	width of window in pixels
window height	RenderSurface	height	height of window in pixels
window x position	RenderSurface	x	x position of screen; origin is upper left
window y position	RenderSurface	y	y position of screen; origin is upper left
screen number	RenderSurface	screenNum	should be the id of the x window defined in the nvidia-settings
viewport width	Camera	viewPortWidth	width of window in pixels
viewport height	Camera	viewPortHeight	height of window in pixels
viewport x position	Camera	viewPortX	x position within the window; origin is upper left
viewport y position	Camera	viewPortY	y position within the window; origin is upper left
field of view horizontal	SymmetricPerspectiveAngles	fovX	horizontal FOV angle in degrees
field of view vertical	SymmetricPerspectiveAngles	fovY	vertical FOV angle in degrees
xyz offset	SymmetricPerspectiveAngles	offsetXYZ	relative position to the camera; x: right, y: front, z: up
hpr offset	SymmetricPerspectiveAngles	offsetHPR	relative orientation to the camera; x: heading, y: pitch z: roll

draw thread cpu id	Graphics	drawThreadCPUAffinity	cpu id of the draw thread, distributing to different cpus will increase performance
app/cull thread cpu id	Graphics	appCullThreadCPUAffinity	cpu id of the app and cull thread, distributing to different cpus will increase performance

How do I run multiple channels within one render surface?

You may run multiple channels within a single render surface by defining multiple cameras in the IG Display configuration file (e.g. `Data/Setups/Current/Config/ImageGenerator/AutoConfigDisplay.xml`).

Example:

```
<IGconfig>
   <SystemConfig>
     <Graphics smallFeatureCullPixelSize="3.9"  lodscale="0.9" gamma="1 1 1"  enableCursor="0" enableDatabasePagerThread="0" d
       <RenderSurface name="mainRS" x="0" y="0" width="1600" height="600" depthBits="24" sampleBuffers="0" samples="0" borderV
       <Camera name="cam1" renderSurface="mainRS" viewPortX="0" viewPortY="0" viewPortWidth="800" viewPortHeight="600">
         <SymmetricPerspectiveAngles  fovX="45.00" fovY="30.00" near="1" far="1500" offsetHPR="0 0 0" offsetXYZ="0 0.0 0"/>
       </Camera>
       <Camera name="cam2" renderSurface="mainRS" viewPortX="800" viewPortY="0" viewPortWidth="800" viewPortHeight="600">
         <SymmetricPerspectiveAngles  fovX="45.00" fovY="30.00" near="1" far="1500" offsetHPR="180 0 0" offsetXYZ="0 0.0 0"/>
       </Camera>
     </Graphics>
   </SystemConfig>
 </IGconfig>
```

This will run two 800x600 channels side-by-side with the right one looking into the rear direction. If you perform individual settings in the display configuration file, make sure you turn off automatic display configuration and interpretation of projection messages by the IG (see above).

For a fullHD **stereo** setup with 16cm distance between the viewpoints, two channels side-by-side, the config file will look as follows:

```
<IGconfig>
   <SystemConfig>
     <Graphics smallFeatureCullPixelSize="3.9"  lodscale="0.9" gamma="1 1 1"  enableCursor="0" enableDatabasePagerThread="0" d
       <RenderSurface name="mainRS" x="0" y="0" width="1920" height="1080" depthBits="24" sampleBuffers="0" samples="0" border
       <Camera name="left" renderSurface="mainRS" viewPortX="0" viewPortY="0" viewPortWidth="960" viewPortHeight="1080">
         <SymmetricPerspectiveAngles  fovX="40.00" fovY="45" near="1" far="1500" offsetHPR="0 0 0" offsetXYZ="-0.08 0.0 0"/>
       </Camera>
       <Camera name="right" renderSurface="mainRS" viewPortX="960" viewPortY="0" viewPortWidth="960" viewPortHeight="1080">
         <SymmetricPerspectiveAngles  fovX="40.00" fovY="45" near="1" far="1500" offsetHPR="0 0 0" offsetXYZ="0.08 0.0 0"/>
       </Camera>
     </Graphics>
   </SystemConfig>
 </IGconfig>
```

How do I configure the viewing angles for a (multi-channel) projection system?

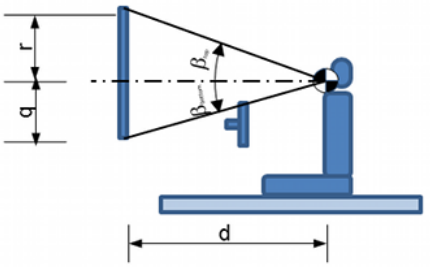
If you are running VTD in a (multi-channel) projection system, the correlation between your eye-point and the projection screens is usually fix. Therefore, you should disable the interpretation of viewing angles sent by the GUI and use a set of individual configuration files instead.

1) For each display, create an individual display configuration file in `Data/Setups/Current/Config/ImageGenerator`, e.g.

```
DisplayLeft.cfg
DisplayCtr.cfg
DisplayRight.cfg
```

In each file, configure the viewing angles of each individual display. The following image shows how these angles are computed:

single channel

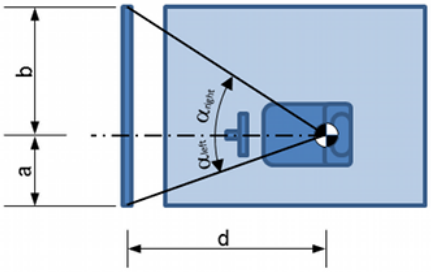


vertical angles

$$\beta_{top} = \text{atan}(\frac{r}{d})$$

$$\beta_{bottom} = \text{atan}(\frac{q}{d})$$

Note: *a, b, q and r are the dimensions of the visible area on the screen (without display frame)*

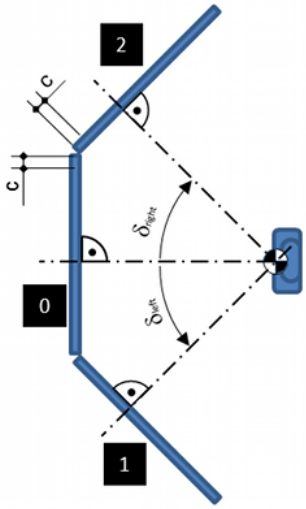


horizontal angles

$$\alpha_{left} = \text{atan}(\frac{a}{d})$$

$$\alpha_{right} = \text{atan}(\frac{b}{d})$$

multi-channel



Setup with displays of identical size:

each channel:
horizontal angles as above
vertical angles identical for all channels

horizontal offset between channels borderless:

$$\delta_{left} = \alpha_{left,0} + \alpha_{right,1}$$

$$\delta_{right} = \alpha_{left,2} + \alpha_{right,0}$$

with border (width c):

$$\delta_{left} = \text{atan}(\frac{a_0+c}{d}) + \text{atan}(\frac{b_1+c}{d})$$

$$\delta_{right} = \text{atan}(\frac{a_2+c}{d}) + \text{atan}(\frac{b_0+c}{d})$$

The resulting numbers should be put into the display configuration file as follows (replace symbolic names alpha... etc. with the actual numbers, angles must be given in [deg]):

symmetric frustum

```
<IGconfig>
  <SystemConfig>
    <Graphics smallFeatureCullPixelSize="3.9" lodscale="1.0" gamma="1 1 1" enableCursor="0">
        <RenderSurface name="myRS" x="0" y="0" width="1920" height="1080" visualID="36" borderVisible="0"/>
        <Camera name="myCam" renderSurface="myRS" viewPortX="0" viewPortY="0" viewPortWidth="1920" viewPortHeight="1080">
         <SymmetricPerspectiveAngles  fovX="alphaLeft + alphaRight" fovY="betaBottom + betaTop" near="1" far="2500" offsetHPR=
    </Camera>
    </Graphics>
  </SystemConfig>
</IGconfig>
```

asymmetric frustum

```
<IGconfig>
  <SystemConfig>
    <Graphics smallFeatureCullPixelSize="3.9" lodscale="1.0" gamma="1 1 1" enableCursor="0">
        <RenderSurface name="myRS" x="0" y="0" width="1920" height="1080" visualID="36" borderVisible="0"/>
        <Camera name="myCam" renderSurface="myRS" viewPortX="0" viewPortY="0" viewPortWidth="1920" viewPortHeight="1080">
      <AsymmetricPerspectiveAngles left="alphaLeft" right="alphaRight" bottom="betaBottom" top="betaTop" near="1.0" far="1000.
    </Camera>
    </Graphics>
```

```
  </SystemConfig>
</IGconfig>
```

3) Refer to the display configuration files in individual setup files, e.g.

```
CfgLeft.cfg
CfgCtr.cfg
CfgRight.cfg
```

The content of each of these files should look similar to the following (`CfgLeft.xml`):

```
<IGconfig>
  <Includes>
    :
    <Include file="IGLinks/Setup/Config/ImageGenerator/DisplayLeft.xml" />
    :
  </Includes>
</IGconfig>
```

4) Modify your configuration file `IGbase.xml` so that messages altering the projection settings are ignored:

```
 <TAKATA name="TAKATA"
        :
               ignoreSetProjectionMessages="1"
               :
               :>
```

Important Note: If you are running multiple displays and you have the impression that the transition between adjacent channels is inconsistent (e.g. parts of the scene are missing before they appear in a side channel) then check the **zoom settings** of your displays and make sure they show the **entire** desktop.

Calculate angles of the viewing frustum

Note: All angles in radians

Rectlinear images

```
vFoV = 2 * atan(tan(hFoV/2) * height/width)
hFoV = 2 * atan(tan(vFoV/2) * width/height)
```

Fisheye or equirectangular images

```
vFoV = hFoV * height/width
hFov = vFoV * width/height
```

How can I run two IGs for two different players on a single computer?

We have prepared a setup *DualIGDualPlayer* for this configuration. It can be found here:

dualIGDualPlayer

Details:

- the setup assumes that the scenario contains two *externally* controlled players **A** and **B**
- for each player, an IG configuration can be found in *Config/ImageGenerator* (`CfgA.xml` and `CfgB.xml`)
- window size and viewing angles are fix for each IG instance and can be adapted in the files `CfgDisplayA.xml` and `CfgDisplayB.xml`
- the start commands for the IGs are in *Config/Simulation/taskSettings.cfg* and *Config/Simulation/simSettings.cfg*
- the script `startTaskIG.sh` has been added to the sub-dir *Scripts* in order to perform a re-start of the IGs upon change of the visual database
- the script `initProject.sh` has been added to the sub-dir *Scripts*. Its purpose is the definition of the individual camera positions (relative to players "A" and "B"), the visibility of the players and some debug information (the latter can, of course, be removed from the script). It is executed each time the "INIT" button is pressed or "INIT" is executed implicitly.
- a moduleManager setup has been added to *Config/ModuleManager* and provides two instances of the simple VIRES vehicle dynamics

A sample scenario which can be used in connection with the setup is provided here:

crossing8DualExt.xml

Defining Channel Offsets

In AutoCfgDisplay.xml the frustum may be defined using the tags <SymmetricPerspectiveAngles> or <AymmetricPerspectiveAngles>; both tags provide an attribute which is called `offsetXYZ`. Please note that the XYZ values must be given in graphics co-ordinates, not in vehicle co-ordinates. The relationship between both systems is:

```
gfx X = - vehicle Y
gfx Y = vehicle X
gfx Z = vehicle Z
```

How can I define a rear-view mirror

Picture-in-picture mode

Rear-view mirrors which shall be displayed picture-in-picture in e.g. a front-view channel must be defined in the file *IGBase.xml* (in *Data/Setups/Current /Config/ImageGenerator*).

Example (for an Audi Q5):

```
  ...
  <Components>
    <Mirror name="leftMirror"
              screenPosX="20" screenPosY="100"
          sizeX="200" sizeY="150"
          left="12" right="7"
          bottom="7" top="7"
          near="0.2" far="200"
          positionOffset="-1.0 2.0 1.2"  orientationOffset="180 0 0"/>
...
```

Explanations:

- name: user defined string
- screenPosX: [pixel] (from left edge)
- screenPosY: [pixel] (from bottom edge)
- sizeX: horizontal dimension [pixel]
- sizeY: vertical dimension [pixel]
- left: frustum angle to the left [deg]
- right: frustum angle to the right [deg]
- bottom: frustum angle to the bottom [deg]
- top: frustum angle to the top [deg]
- near: near clipping plane [m]
- far: far clipping plane [m]
- positionOffset: offset from simCar position, vIG system, x (right) [m], y (forward) [m], z (up) [m]
- orientationOffset: heading (rotation around z-axis) [deg], pitch (rotation around x' axis) [deg], roll (rotation around y" axis) [deg]

Individual Channel

If you want to run an individual channel as mirror channel, you will have to perform the following configuration steps in the image generator setup directory (typically `Data/Setups/Current/Config/ImageGenerator`):

1) Create your own master configuration file (here: copy `AutoCfg.xml` to `CfgMirrorLeft.xml`)
2) Edit the configuration file so that it looks similar to the following **CfgMirrorLeft.xml**

```
<IGconfig>
  <Includes>
    <Include file="IGLinks/Setup/Config/ImageGenerator/IGbaseMirrorLeft.xml" />
    <Include file="IGLinks/Distro/Config/ImageGenerator/Sky.xml" />
    <Include file="IGLinks/Distro/Config/ImageGenerator/Materials.xml" />
    <Include file="IGLinks/Project/Config/ImageGenerator/SymbolsMirrorLeft.xml" />
    <Include file="IGLinks/Project/Config/ImageGenerator/LightSrcStd.xml" />
    <Include file="IGLinks/Setup/Config/ImageGenerator/DisplayMirrorLeft.xml" />
    <Include file="IGLinks/Setup/Config/ImageGenerator/AutoCfgDatabase.xml" />
  </Includes>
</IGconfig>
```

3) Provide a separate display configuration, here **DisplayMirrorLeft.xml**

```
<IGconfig>
  <SystemConfig>
      <Graphics  polygonCullingMode="CULL_FRONTFACE" smallFeatureCullPixelSize="3.9"  lodscale="1.0" gamma="1 1 1" enableCurso
        <RenderSurface name="mainRS" x="0" y="0"  width="640" height="480"  depthBits="24" sampleBuffers="0" samples="0" borde
        <Camera name="cam1" renderSurface="mainRS" viewPortX="0" viewPortY="0" viewPortWidth="640" viewPortHeight="480" >
            <SymmetricPerspectiveAngles  fovX="16" fovY="12" near="1" far="1500" offsetHPR="0 180 0" offsetXYZ="-1.2 1.0 0" mi
        </Camera>
    </Graphics>
  </SystemConfig>
</IGconfig>
```

Note that in this file, the attributes `polygonCullingMode` and `mirror` are set accordingly. For `offsetHPR` and `offsetXYZ` you should define what is appropriate.

4) Provide a separate IG base configuration file, here **IGbaseMirrorLeft.xml**

```
 <TAKATA name="TAKATA"
         :
                ignoreSetProjectionMessages="1"
                :
                :>
```

Note: you only have to disable the interpretation of projection messages in the IGbase file. Therefore, if you want to reduce the number of files in your configuration, you may also think about ignoring the projection messages in general and perform the definitions in the respective display configuration files. If you do this, you don't have to copy `IGbase.xml` but you can use one file for all channels. In the parent configuration file `CfgMirrorLeft.xml` you will then have to refer to `IGbase.xml` instead of `IGbaseMirrorLeft.xml`.

How can I place the IG output window on another X-display but :0.0?

The X-display for the output window is read from the file *AutoCfgDisplay.xml* (in *Data/Setups/Current/Config/ImageGenerator*). Modify the tag `<RenderSurface/>` so that the attributes *displayNum* and *screenNum* have the corresponding values.

Example for output on :0.1:

```
<RenderSurface ... displayNum="0" screenNum="1" />.
```

Usually, the file *AutoCfgDisplay.xml* is created automatically upon re-start of the IG, so that the actual modification should be performed in the script *configureDisplay.sh*. This script can be found at *Data/Setups/* ***Current*** */Scripts/* or *Data/Setups/* ***Common*** */Scripts/*. It is highly recommended to create a local copy of the script in your current setup unless you want to apply the changes to all setups at once.

How I can run multiple ImageGenerators on multiple GPU cards?

Basically, you will need to create a second xscreen with nvidia-settings.
Then you need to set the screenNum to the second screen in your Setup/Config/ImageGenerator/*CfgDisplay.xml:

```
<RenderSurface ... displayNum="0" screenNum="1" />
```

In this way, the ImageGenerator will run on the second GPU.

How can I avoid horizontal tearing of my output?

The video retrace of the image generator must be synchronized with the physical output device that is being used for the respective display. Usually, the graphics card has more than one connector and it is mandatory that the retrace be synchronized with the correct one. Here's what you have to do:

- Start the tool "nvidia-settings"
- Check the name of the output device to which your monitor is connected (e.g. DFP-0, DFP-1, ..)
- In the file `IGbase.xml` or `IGenv???.xml` which belongs to your ouptut channel, add the following line `<EnvVariable value="DFP-0" var="__GL_SYNC_DISPLAY_DEVICE" />`; instead of `DFP-0` type the name of your device.
- You may also define the variable in `Data/Setups/Current/Config/Simulation/envSettings.cfg` if it applies to all IGs in your current setup. If the variable is set in the `envSettings.cfg` and in `IGbase.xml` the latter definition will win.

How can I use server-side rendering for an output channel?

If you want to use server-side rendering, you have to adjust two files:

1) In `Data/Setups/Current/Config/ImageGenerator/AutoCfgDisplay.xml` set the variable `displayNum` to the number of the display on which the rendering on the server will take place. In order to have this variable set automatically upon re-configuration of the IG, copy the script `startTaskIG.sh` from `Data/Setups/Common/Scripts` to `Data/Setups/Current/Scripts` and adjust the variable `displayNum` in this script.

2) In `Data/Setups/Current/Config/Simulation/taskSettings.cfg`, configure the IG start command (variable `START_CMD`), so that it is preceded by `ssrrun`. If the start command for the IG is not yet present in this file, copy the respective lines from the file `Data/Setups/Common/Config/Simulation/taskSettings.cfg`. The configuration in the `Current` setup will supersede the one of the `Common` setup (**note**: the exact description applies only to the situation when you are using "Exceed onDemand" by "OpenText" but other applications might run into similar problems).

How can I address a single channel in multi-channel configurations

Starting with VTD 1.2.3, it is possible to address individual channels in certain commands. The instances of the IG that shall interpret channel masks must be started with the command line parameter `-m <maskId>` where `maskId` is a hex number.

Channel masking is currently available for the following SCP commands:

- `<Symbol channel="..."></Symbol>`
- `<Camera...><Set channel="..."/></Camera>`
- `<Display channel="..."></Display>`
- `<Player...><Visibility channel="..."/></Player>`
- `<Video><Start channel="..."/></Video>`

Further restrictions may apply for each command. For details, please have a look at the SCP communication.

Configuration

In this example, the mask is set to *0x2*. This instance of the IG will only show symbols or apply camera settings without an explicit channel mask or the ones with a channel mask containing *0x2* (see also Symbols in multi-channel configuration

VTD 2.0+

Example (`Data/Setups/Current/Config/SimServer/simServer.xml`):

```
    <Process
      group="igGroup"
      name="igCtr"
      auto="false"
      explicitLoad="false"
      path="$VI_CORE_DIR/ImageGenerator/bin"
      executable="vigcar"
      cmdline="$VI_CURRENT_SETUP/Config/ImageGenerator/AutoCfg.xml -m 0x01"
      useXterm="true"
      xtermOptions="-fg Black -bg LightGoldenRod -geometry 80x10+508+163"
      affinitymask="0xFF"
      schedPolicy="SCHED_RR"
      schedPriority="20"
      workDir="$VI_CURRENT_SETUP/Bin">
      <EnvVar name="LD_LIBRARY_PATH"      val="$VI_CORE_DIR/ImageGenerator/bin:$LD_LIBRARY_PATH"   />
```

```
      </Process>
```

VTD 1.4.3 and earlier:

Example (`Data/Setups/Current/Config/Simulation/taskSettings.cfg`):

```
if { test $TASK_TYPE = "igCtr" } then
  set BIN_NAME      = vigcar
  set OBJ_DIR       = ( $VI_CORE_DIR/ImageGenerator/bin )
  set WORK_DIR      = ( . )
  set CMD_ARGS      = ( $VI_SETUP_DIR/Config/ImageGenerator/AutoCfg.xml -m 0x2 )
  set START_CMD     = ( source $ENV_SETTINGS ; cd $WORK_DIR ; $OBJ_DIR/$BIN_NAME $CMD_ARGS $RECORD_CMD ; sleep $XTERM_LIFETIME
  set USE_XTERM     = true
  set XTERM_OPTIONS = ( $XTERM_OPTIONS_COMMON -bg LightGoldenRod -geometry 80x10+508+163 -title IMAGEGENERATOR )
endif
```

How can I configure warping?

Deprecated 1.3.4:

Warping requires a PostProcessing and a Warping component. Following lines must be added within a <Components> section to configure warping:

```
    <PostProcessing name="MyPostProcessing" />

    <Warping name="MyWarping">
      <WarpingStep name="Warp1"  distortionFile="exampleDistortion.txt" numGridPoints="20 30" blendImageFile="exampleAlphamap.
    </Warping>
```

Current:

```
0    <PostProcessing name="MyPostProcessing" />
1    <PostProcessingPipelineConfigurator name="MyPostProcessingPipelineConfigurator">
2        <Pipeline hideSceneFromDefaultView="1" >
3            <Step type="PPSTextureRect" name="OriginalScene">
4                <Inputs renderLights="1">
5                    <NodeInput  inputNo="0" type="scene"  />
6                </Inputs>
7                <RTT  sizeMode="viewport"  bufferFormat="GL_RGB8" />
8
9                <SAQ lowerLeftX="0" lowerLeftY="0" width="0.5" height="1" isForDebug="1" enableSRGB="0" />
10           </Step>
11
12           <Step type="PPSWarpingMesh" name="Warping" >
13               <Inputs sourceStep="OriginalScene" blendImageAlpha="exampleAlphamap.rgb" />
14               <RTT  sizeMode="viewport"  bufferFormat="GL_RGB8" />
15
16               <Mesh width="20" height="30" file="exampleDistortion.txt" />
17
18               <SAQ lowerLeftX="0.0" lowerLeftY="0" width="1" height="1" isForDebug="0" enableSRGB="0" />
19           </Step>
20       </Pipeline>
21   </PostProcessingPipelineConfigurator>
```

Parameters used for Warping:

- File containing the distorition mesh: Mesh/file (line 16)
- Sizes of the grid in the mesh file Mesh/width and Mesh/height (line 16)
- Optional alpha texture file to fade visible image to black Inputs/blendImageAlpha (line 13)

These lines can be added to the <Components> section of the ProjectIGConfig.xml.

In case a multi-channel setup is used, warping of each channel must be configured differently. In this case separate XML files that contain warping configuration for individual channels must be created (within the Setups) and included in channel configuration xml's (CfgLeft.xml etc.). The format of these separate XML's will be the same as ProjectIGConfig.xml.

How can I configure cube map/wide angle warping?

Cube map warping requires a PostProcessing component with a CubeMap step to render a wide view of the scene and a PPSTextureRect step to compute the warped image.

The example cube map (`Step type="PPSCubeMap"...`) limits the far plane of each side to 150 meters for performance reasons and disables the most likely unused back of the cube (neg_y_enable="0"). The input of the cube map is the scene. Note that the camera direction is explicitly given relative to the camera position provided by the simulation. If you want to use exactly the simulation's camera position, all attributes (`xyz` and `hpr` within `<CameraConfiguration>`) should be zero.

The distortion step (`Step type="PPSTextureRect"...`) uses a simple lookup shader to map each pixel to a direction into the cube map. The first input of the lens step is the cube map texture generated in the previous step, followed by a lookup table (`sourceLookupTable="fisheye_vektoren.dat"`) specifying the distortion and an alpha blend map (`sourceImage="exampleAlphamap.rgb"`).

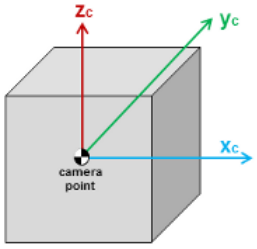
```
    <PostProcessing name="MyPostProcessing />
    <PostProcessingPipelineConfigurator name="MyPostProcessingPipelineConfigurator">
        <Pipeline hideSceneFromDefaultView="1" >
```

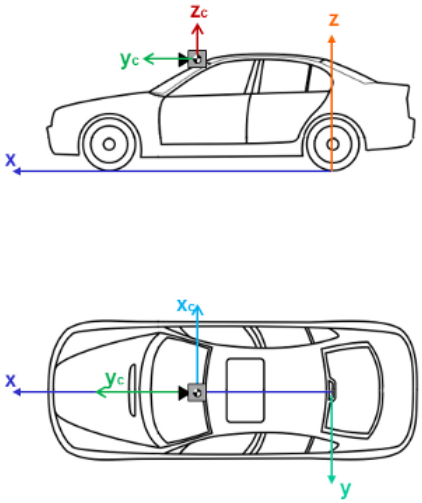
```
 <Step type="PPSCubeMap" name="CubeInput" generateDepthTexture="0" >
        <CameraConfiguration
               xyz="1 2 0.6"
               hpr="-90 0 0"
               pos_y_far="150"
               neg_y_enable="0"
               pos_x_far="150"
               neg_x_far="150"
               pos_z_far="150"
               neg_z_far="150"
               pos_y_near="0.2"
               pos_x_near="0.2"
               neg_x_near="0.2"
               pos_z_near="0.2"
               neg_z_near="0.2" />
        <Inputs renderLights="1">
            <NodeInput inputNo="0" type="scene"  />
        </Inputs>
        <RTT  size="1024"  bufferFormat="GL_RGB16F" />
        <SAQ lowerLeftX="0.5" lowerLeftY="0" width="0.5" height="0.5" isForDebug="1" enableSRGB="0" />
</Step>

<Step type="PPSTextureRect" name="LensColor" >
      <Inputs>
        <TextureInput inputNo="0" sourceStep="CubeInput" outputSlot="color"/>
        <TextureInput inputNo="1" sourceLookupTable="fisheye_vektoren.dat" />
        <TextureInput inputNo="2" textureType="TextureRectangle"  sourceImage="exampleAlphamap.rgb" />
      </Inputs>

      <RTT  sizeMode="explicit" width="800" height="600"  bufferFormat="GL_R32F" />

      <Program vertexShader="../data/Shaders/lookupBasedLens.vert" fragmentShader="../data/Shaders/lookupBasedLens.frag
        <Uniform type="samplerCube" name="sceneTex" value="0"/>
        <Uniform type="sampler2D" name="lookupTex" value="1"/>
        <Uniform type="sampler2D" name="edgeBlendTex" value="2"/>
      </Program>

      <SAQ lowerLeftX="0.0" lowerLeftY="0.5" width="0.5" height="0.5" isForDebug="1" enableSRGB="0" />
</Step>
```

The lookup table is a plain ascii file. Its file format consists of a short header followed by a two dimensional grid of 3-component float vectors. Each vector is used to perform the cube map lookup for the corresponding pixel in the resulting image.
The header specifies a version (line 1) and both width and height (line 2) of the table. Each line in the table consists of width number vectors separated by a "|" and each vector consists of three float values separated by ",".

```
1
1280 800
-0.8208 , -0.2920 ,  0.4910 | -0.8210 , -0.2898 ,  0.4919 | -0.8213 , -0.2876 ,  0.4928 |  ... -0.8213 , -0.2876 ,  0.4928
...
-0.8208 , -0.2920 ,  0.4910 | -0.8210 , -0.2898 ,  0.4919 | -0.8213 , -0.2876 ,  0.4928 |  ... -0.8213 , -0.2876 ,  0.4928
```

Each triplet describes a normalized, directional vector in the following co-ordinate system (note: the cube-map orientation is linked to the camera orientation; in the figure, a camera is mounted pointing into the vehicle's fwd direction; cube map co-ordinates are shown with the suffix "c"). The center coordinate of a front facing image is 0,1,0 with x giving the lookup vector's orientation left (x < 0) to right (x > 0), y front (y > 0) or back (y < 0) and z up (z > 0) or down (z < 0).





How can I use the full screen for the IG display?

Usually, it is sufficient to specify the screen resolution as window and render surface resolution in your IG configuration files. In certain cases, depending on Xserver etc, there may still remain a small stripe of unused pixel at the location of the task bar. In this case, you can force the image generator to use also this portion of the screen. If you use a custom display configuration file (i.e. other than `Data/Setups/Current/Config/ImageGenerator/AutoCfgDisplay.xml`) then set the attribute

```
overrideRedirect="1"
```

in the section `<RenderSurface>` within your file.

If you are using the standard configuration initially delivered with the VTD distribution, perform the following steps:

- create a directory `Data/Setups/Current/Scripts`
- copy the script `Data/Setups/Common/Scripts/configureDisplay.sh` to the new directory
- adjust the `overrideRedirect` attribute within the copied script as described above
- re-start and re-initialize your simulation

Note: by setting the overrideRedirect flag you will no longer be able to issue key or mouse inputs within the IG window. As a consequence, you will also not be able to turn on statistics on-the-fly by pressing the "S" button. If you want to monitor statistics anyway, please enable the corresponding entry in your IG configuration file which is per default

```
Data/Setups/Common/Config/ImageGenerator/CommonIGConfig.xml
```

(just set the attribute `showOnScreen` to `1` in the tag `<Statistics>`)

Another note: the overrideRedirect-flag is intended for full-screen mode only. It may (i.e. it certainly will) interfere with any border mode of the IG window. Therefore, if you want to run an IG with border but not full-screen, you have to set the flag `overrideRedirect` to `0` again.

How can I change the output notify level of the IG/OSG

Change the notify level of the IG and/or the OSG to display more or less log information.

```
<Notification notifyLevel="WARN" OSGNotifyLevel="WARN"/>
```

Common level:

- notifyLevel (IG): FATAL, WARN, INFO, DEBUG
- OSGNotifyLevel (OSG): FATAL, WARN, INFO

located at: `Data/Setups/Common/Config/ImageGenerator/CommonIGConfig.xml`

XServer Configuration

Trees are shown without textures on leaves

If trees look "odd" and also other things like shading don't appear correct, you will most probably have to adjust your graphics driver settings.

- open the NVIDIA X Server Settings program (or type `nvidia-settings` in a shell)
- select the `OpenGL Settings` panel under the respective `XScreen` entry
- disable `Use Conformant Texture Clamping`
- restart your VTD application

Communication

How to avoid UDP broadcasts of the Image Generator communication?

1. Perform the following settings in the file *Data/Setups/Current/Config/TaskControl/taskControl.xml*

```
<ImageGenerator ...
                portType="TCP"
                ctrlPortConnect="true"
                ...
/>
```

2. Perform the following settings in the file *Data/Setups/Current/Config/ImageGenerator/IGbase.xml*

```
<TAKATA ...
   taskControlServerAddress="127.0.0.1"
   taskControlServerPort="13112"
   connectTaskServerTCP="1"
   ...
   trigger="TCP"
   ...
/>
```

The IG will try to connect as client to the TaskControl which acts as server of the connection. If your server (TC) is not local, then adjust the attribute *taskControlServerAddress* accordingly.

Headlights

Note: see also FUSC

How to Address Various Light States

In the configuration file *LightSrcStd.xml* various states for light sources may be defined. The same states may be addressed e.g. via RDB with a numerical ID.

The link between the symbolic names of the states and the numerical IDs is a follows:

```
LowBeamTexture              state=1
HighBeamTexture             state=2
TopBeamTexture              state=3
LowBeamAndTopBeamTexture    state=4
HighBeamAndTopBeamTexture   state=5
FoglightTexture             state=6
LowBeamWithFogTexture       state=7
HighBeamWithFogTexture      state=8
```

How are LightSources for headlights defined?

Headlights consist of a texture which is projected into 3D space with a given intensity, frustum, attenuation etc. The textures for each of the available states are defined in the file which is typically named *LightSrcStd.xml* (usually in the *Distro* or *Data/Projects/Current/Config/ImageGenerator*). The available light definitions are organized as templates, each giving the possibility to define exactly one texture per state. For the availability of different textures per state (for different light sources, of course), multiple templates may be defined.

```
template 0
  state 0: textureA
  state 1: textureB
  state 2: textureC
  :
template 1
  state 0: textureX
  state 1: textureY
  state 2: textureZ
```

Now that templates and states are defined, light sources may be instantiated (typically via RDB) by assigning the corresponding template and state. Templates and state may be defined multiple times to different light sources (see following example):

```
  lightSource 1:  texture from template 0, state 1
  lightSource 2:  texture from template 0, state 0
  lightSource 3:  texture from template 1, state 1
```

For the states, their names in the light source definition file and the numeric state, please see above (How to Address Various Light States)

Light on (vertically) curved road surfaces looks "odd"

On vertically curved road surfaces, headlights may look odd in terms of showing horizontal separating lines. These originate from the tesselation of the database and may be minimized by using more polygons for the database generation. In ROD, this can be done in the setup file `TT_SETUP.DAT`

Lights are Defined but not Visible in ImageGenerator

If you are using ImageGenerator 4.9.0 and above (i.e. after VTD 1.1), the shader associations in your configuration file *LightSrcStd.xml* need to be updated as follows (example is given for standard configuration):

old version:

```
<HeadlightVertexShader   file="light.vert" />
<HeadlightFragmentShader file="light.frag" />

<ShaderPair name="HeadlightTree"      vertexshader="lightrot.vert"  fragmentshader="light.frag"/>
<ShaderPair name="HeadlightCar"       vertexshader="light.vert"     fragmentshader="lightcar.frag"/>
<ShaderPair name="HeadlightCarWheels" vertexshader="light.vert"     fragmentshader="light.frag"/>
```

new version:

```
<HeadlightVertexShader   file="lightVert.glsl" />
<HeadlightFragmentShader file="lightFrag.glsl" />

<ShaderPair name="HeadlightTree"      vertexshader="lightrotVert.glsl"  fragmentshader="lightFrag.glsl"/>
<ShaderPair name="HeadlightCar"       vertexshader="lightVert.glsl"     fragmentshader="lightcarFrag.glsl"/>
<ShaderPair name="HeadlightCarWheels" vertexshader="lightVert.glsl"     fragmentshader="lighFrag.glsl"/>
```

Headlights disappear with increasing distance

The IG will disable the headlights reaching a pre defined (as of VTD.1.4: 200m) distance to the camera.
A modification may yield in a performance drop depending on the number of simultaneously visible headlights.

1. headlightsVisibilityDistance: range headlights are visible [m]
   (see Data/Setups/Current/Config/ImageGenerator/IGBase.xml)

   ```
   <TAKATA name="TAKATA"
     ...
     headlightsVisibilityDistance="300"
     ... />
   ```

2. HeadlightPassNearFar: due to step one, headlights are rendered but can be applied on a specific range [m]. In common, far and headlightsVisiblilityDistance are equal.

(see Runtime/Core/ImageGenerator/data/Headlights.xml)

```
<HeadlightPassNearFar near="1" far="200" />
```

3. FadeInRange: lights have to be faded in or out (avoid "pop out objects" or "pop in" effects) [m]
4. EffectRange: range around the assigned object where the effect takes place [m]
   (see Runtime/Core/ImageGenerator/data/BaseLights.xml)

```
<LightCluster name="VehicleLight" fadeInRange="180 200"  lightSourceGeomRadiance="1 1 0.95" effectRange="200" >
```

Ego Vehicle shows "strange" light cones

If the 3D-model of the Ego vehicle is turned on ("Show Ego" in the GUI), and headlights are ON in general for all vehicles, also the Ego vehicle will show the simplified light cone that is defined for the vehicle model (in *IGbase.xml* or *vehiclCfg.xml*). If you want to show your own vehicle model at night, please define and assign a "black" light texture for your own vehicle model in the appropriate files (see above).

How is Light Source attenuation handled?

Attenuation consists of constant, linear, and quadratic terms. Overall intensity of a light at distance *d* from the source is

```
intensity = sourceIntensity * ( constantAtt + linearAtt/d + quadraticAtt / d^2 )
```

Inverse square law applies for the amount of light that falls onto a surface from a light source. This could be simulated as follows:
constant=0
linear=0
quadratic=1

If the light travels in a medium, such as air, the amount of light that reaches to the surface from a light source might be less than expected because some of the photons will be scattered on the way. This could be simulated by increasing linear and/or constant attenuation factors.

How can I avoid delays in the application of LightSources?

If a light source (e.g. headlight) is sent to the IG for the first time, it will cause an initialization routine which might interrupt the flow of the IG. If you want to avoid this effect and if you know in advance which light sources you are going to address during a simulation, then you may already send the light sources during the init phase of the IG. Just send the corresponding RDB message which controls your light source (e.g. with dummy values for player and position), but make sure that the light source's ID corresponds to the one which you are going to use later-on. Also, take care that the template and state refer to values which are available in the light source definition file (typically *LightSrcStd.xml*). The most important point of the RDB message is that you add the flag

`RDB_LIGHT_SOURCE_FLAG_PERSISTENT`

to the light source's flags (member `flags` in structure *RDB_LIGHT_SOURCE_BASE_t*).

Although there is no harm to the initialization, the RDB message for the pre-configuration should be sent with low frequency (e.g. 2 Hz).

In the TaskControl configuration file (typically *Data/Setups/Current/Config/TaskControl/taskControl.xml*), please include the following setting:

```
<TaskControl>
    <ImageGenerator ...
                    autoHeadlightExt="false"
                    ... />
</TaskControl>
```

Databases and Vehicles

IG crashes when loading SmartDB

If you are using a 32bit IG, you are limited by the 4GB memory that can be allocated. In order to load a SmartDB, you have to derive a setup from *Standard.Smart32.RDB* and you have to load the "2008"-Version of the SmartDB. The setup will cause the IG to ignore the new material definitions that come with the latest version and apply the "old" default materials instead. So, your database will have the same look as a VTD 1.0 - database. The difference to a "regular" setup is that the file *Materials.xml* is not referenced from within *Data/Setups/Current/ImageGenerator/IGbase.xml*

Since this is just a workaround, we highly recommend that you upgrade your IG computer to the 64 bit version. Please see also the Tips'n'Tricks in (see VTD Configuration Issues) for installing new systems.

Integration of a new vehicle model

In contrast to previous editions of VTD, new vehicle models should only be registered within the file *vehicleCfg.xml* which is typically located in *Data/Distros /Current/Config/Players* or *Data/Projects/Current/Config/Players*. In order to have the vehicle configuration forwarded to the IG upon initialization of a simulation, you have to adjust the TaskControl configuration file (typically: *Data/Setups/Current/Config/TaskControl/taskControl.xml*):

```
<TaskControl>
    ...
    <ImageGenerator  dynPlayerConfig="true" .../>
    ...
</TaskControl>
```

For the remaining configuration steps, please refer to FAQ

How can I identify the name of a texture and the material of an object in the scene?

The image generator provides a debug means to identify individual textures and materials within the scene. In order to pick a texture and retrieve its information, perform the following steps:

- select the IG output window with the cursor
- typically, the cursor will not be visible; press "c" to render the cursor visible
- move the cursor over the texture or object that you want to identify
- press "i"
- the texture information will be in the xterm belonging to the image generator and in the log file (typically `/tmp/taskRec_igCtr.txt`)

Example of a texture information entry:

```
-------------------Pick Information-------------------
Hierarchy: /cam1(Camera)(0x10f8510)/IG scene root(Group)(0x1116a00)/IG world global state node(Group)(0x1116ae0)/ShadowedScenel
StateSet :Silver(0x1cfb0210)
Material:(0xf6692f0)

Texture:/home/dw/projects/svn/ModelLibrary/Cars/MANTGL_09/Textures/Texture_MAN_TGL.rgb(0x4c7bd30)s wrap: REPEAT  t wrap: REPEA

Texture: None
Texture: None
Texture: None
Texture: None
Texture: None
Texture: None
Texture: None
Uniforms:
u_genericConfig
u_transparency
Render Bin Mode:USE_RENDERBIN_DETAILS  Render Bin Num:0   Render Bin Name:RenderBin GL_BLEND:OFF GL_SAMPLE_ALPHA_TO_COVERAGE_A
Pointer: 0xf6692f0
Intersection point: 3858.28 -244.57  1.80
-----------------------------------------------------
```

The above example shows, that the object has the texture **Texture_MAN_TGL.rgb** and the the material (StateSet) **Silver**.

How can I take snapshots?

Preparation:

First, configure the IG's FrameBufferReader, so that it can take snapshots. This is done in the file `Data/Setups/Current/Config/ImageGenerator/IGbase.xml` or, for VTD version 1.3.5 and higher, in `Data/Setups/Common/Config/ImageGenerator/CommonIGConfig.xml`

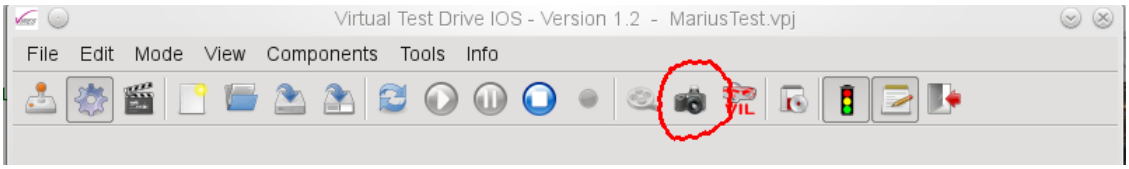
Example:

```
    <FramebufferReader name="MyFBReader" outputPath="/tmp/Buidl" >
        <VideoEncoder filename="/tmp/video.mpg" frameDrops="1" kbps="4000" cpuAffinity="2"/>
    </FramebufferReader>
```

The important part is the attribute `outputPath`. This will determine the location of the resulting snapshot images. The tag `<VideoEncoder>` is for video recording only and should not influence the taking of snapshots.

The comfortable way - via the GUI

In order to take a snapshot, press the camera button on the GUI.



The Image Generator will create two .png files in the target directory.

- `vigScreenshot_nnn.png`
- `vigScreenshot_d_nnn.png`

where nnn is the internal frame number of the IG and _d marks the depth image for the same frame.

The safe and flexible way - via SCP

availability: VTD 1.4+

If you want to trigger the snapshot from outside the GUI or if you want to limit it to a selection of IGs, you may use the following command:

```
<Video><Output name="mySnapshot" path="/tmp" /><Start snapshot="true" channel="0x3"/></Video>
```

This command will take a snapshot of all IGs fulfilling the channel mask *0x3* (i.e. channels *0x1* and *0x2*). If you omit the attribute *channel* all connected IGs will create a snapshot.
Two images will be created:

- `/tmp/mySnapshot_nnn.png`
- `/tmp/mySnapshot_d_nnn.png`

where nnn is the internal frame number of the IG and _d marks the depth image for the same frame.

The old way - via keyboard

If you want to take snapshots of the current IG window, you may do so the following way:

1. focus the IG window (click into the window or move the mouse over it - depending on your desktop settings)
2. press "."

The Image Generator will create two .png files in the target directory.

- `vigScreenshot_nnn.png`
- `vigScreenshot_d_nnn.png`

where nnn is the internal frame number of the IG and _d marks the depth image for the same frame.

NOTE: the snapshot feature will only work if the *FramebufferReader* is configured within the IG configuration file *IGbase.xml*. For the standard distribution, this should already be the case.

How can I increase quality?

Modify the setting `kbps="96000"` for more data per second or/and `fps="60"` for more frames per second according to your requirements in the file

`VTD.2.0/Data/Setups/Common/Config/ImageGenerator/CommonIGConfig.xml` to

```
<VideoEncoder filename="/tmp/video.avi" frameDrops="1" kbps="96000" cpuAffinity="2" fps="60"/>
```

Beware: Don't stop the VIG until the video is written (check file size). The memory usage of the VIG will also increase, so that high bitrates and long videos could lead to problems.

Physical Values

How to convert z-buffer values into distance-to-camera values

The relationship of z-buffer values and distance to xy-plane of camera coordinate system is as follows:

```
z = 0.5*(f+n)/(f-n) + (-f*n)/(f-n) * (1/d) + 0.5

with:
z: z-buffer value (normalized in [0,1]. Non-normalized fixed point zf = z * (s^n - 1 ) where n is bit depth of the depth buffe
d: distance of fragment (pixel) to xy plane of camera coordinate system
n: near plane (camera frustum setting)
f: far plane (camera frustum setting)
```

z-buffer values are generally in the range [0;1]. A fragment with z-buffer value '0'
lies on the near clipping plane, with value '1' on the far clipping plane. However,
if a snapshot of the depth buffer is retrieved in VTD as image via network or
shared memory, the z values are previously scaled to a range between [0;2^32-1] and
stored as unsigned int.

Important note: although the z-buffer may be given in a nominal resolution of 32bit, its values may only contain valid 24bit information. This depends on the graphics card, graphics driver and the visual that is being used for the image generator. If 24bit resolution is the maximum that can be achieved, then the last byte will typically be filled with the values of the first byte.

How do I compute visibility?

VIG uses OpenGL-Fog EXP2. The range of the visibility is defined as the distance where fog reaches 99% density.

fogDensity = sqrtf(-log(0.01f) / log(M_E)) / (m_visibility);

Symbols

How do I define a new symbol?

Symbols are pre-configured in the IG configuration files and are then instantiated via SCP. For the SCP command syntax, see the respective documentation.

Text Symbols

For text symbols, there is nothing special to do. Just issue the corresponding command via SCP.

Textured Symbols

Textured symbols need to be pre-loaded by the image generator. In order to make them available, edit the file `Data/Projects/Current/Config /ImageGenerator/SymbolsStd.xml` and provide either a new `<Symbol>` entry or extend an existing one.

Each `<Symbol>` template must be configured with a unique ID and one or more `<Texture>` entries, each referring to an individual graphics file. The sizes and formats of the files may differ. The other attributes of the `<Symbol>` tag (i.e. `x, y, width, height, scale`) should correspond to the ones in another entry (they are used in execeptions only and are, thus, not relevant to the normal user). Make sure that the attribute `enabled="0"` is set.

For activating a symbol, just refer to the overlay ID and the state (i.e. the texture) via an SCP command.

Example: Display of an on-screen symbol:

```
<Symbol name="scrSymbol" > <Overlay id="0" state="2"/> <PosScreen x="0.5" y="0.5" /> <RectSize width="0.2" height="0.2"/> </Symbol>
```

Symbol Textures from RDB Shared Memory

It is possible to assign a dynamically changing texture to a symbol. The contents of such a texture will be read from an RDB shared memory buffer. For this

purpose <RDBSHMBuffer> keyword must be used. Example:

```
<Symbol id="30" x="0.51" y="0.4" width="0.4" height="0.4" scale="1" enabled="0" >
    <RDBSHMBuffer key="0x8000"/>
    <Texture file="hudDigit_1.rgb"/>
</Symbol>
```

Above definition creates a special texture whose contents are read from a SHM buffer with key 0x8000. This texture will be referenced as state 0. It can be made visible as follows:

```
<Symbol name="scrSymbol" > <Overlay id="30" state="0"/> </Symbol>
```

The SHM buffer must be created by the user application; IG itself is reader only. To publish a buffer to the IG the user application should set the RDB_SHM_BUFFER_FLAG_IG flag in the RDB_SHM_BUFFER_INFO_t structure,
once the IG is finished it will release the buffer to the user application with the RDB_SHM_BUFFER_FLAG_NONE flag.

The IG expects buffers to be made available with the RDB_SHM_BUFFER_FLAG_IG flag each frame and may not keep copies of old images in its own memory. To ensure that the IG can lock an image each frame the shared
memory segment should contain at least two buffers so one can be updated by the user application while the other is read by the IG.

Using double buffering the layout of the buffers should be as follows, with each RDB_IMAGE_t followed by the pixel data of the texture as either RDB_PIX_FORMAT_RGB8 or RDB_PIX_FORMAT_RGBA8:

```
RDB_SHM_HDR_t
#array of buffer headers
  RDB_SHM_BUFFER_INFO_t
  RDB_SHM_BUFFER_INFO_t

#contents of buffer 0
RDB_MSG_HDR_t
  RDB_MSG_ENTRY_HDR_t
    RDB_IMAGE_t

#contents of buffer 1
RDB_MSG_HDR_t
  RDB_MSG_ENTRY_HDR_t
    RDB_IMAGE_t
```

Sizes and pixel formats for each buffer should not change once an image is send to avoid visual glitches and jitter.

How can I assign a symbol to a dedicated channel in multi-channel applications

Starting from VTD 1.2.3, symbols can be assigned to individual channels. For this, the respective instance of the IG has to be started with a channel mask (see Setup of channel masks).

Each symbol that shall appear on a sub-set of the available channels has to be provided with the *channel* attribute. Examples:

```
<Symbol name="sym01" channel="0xffffffff" > <Text data="Symbol with channel mask 0xffffffff" colorRGB="0xffff00" size="20.0"/>
<Symbol name="sym02" channel="0x00000001" > <Text data="Symbol with channel mask 0x00000001" colorRGB="0xff0000" size="20.0"/>
<Symbol name="sym03" channel="0x00000002" > <Text data="Symbol with channel mask 0x00000002" colorRGB="0x0000ff" size="20.0"/>
<Symbol name="sym04" channel="0x00000003" > <Text data="Symbol with channel mask 0x00000003" colorRGB="0x00ff00" size="20.0"/>
```

If, for example, the channel mask is *0x2*, only symbols *sym01*, *sym03* and *sym04* of the list above will be shown in this channel.

Symbols which are to be shown on top of each other are flickering. What can I do?

For the concurrent display of symbols in the same areas of the IG output window, the user has to define the drawing sequence explicitly. For this, the `<Symbol>` tag has been extended with the attribute *layer*. The layer is a numeric ID and determines the sequence of drawing. The higher the number, the later the symbol will be drawn.

Example:

```
<Symbol name="Symbol1" layer="1">
  <Overlay id="0" state="0"/>
  <PosScreen x="0.5" y="0.5"/>
  <RectSize width="0.1" height="0.1"/>
</Symbol>

<Symbol name="Symbol2" layer="2">
  <Overlay id="0" state="4"/>
  <PosScreen x="0.5" y="0.5"/>
  <RectSize width="0.1" height="0.1"/>
</Symbol>
```

Symbols in HDR

By default only Screen space Symbols have color, other Symbols appear black in HDR, they do not receive or emit light and only have fixed colors that are lost during the HDR range conversion. In order to see Symbols with their original colors they have to be rendered independently and merged with the result of the HDR Pipeline. The configuration below is intended to extend a simple HDR pipeline configuration, placeholders exist where the existing pipeline configuration has to be inserted or modified (the HDR pipeline is most likely located in Data/Setups/[SetupName]/Config/ImageGenerator/IGbase.xml):

```
<Components>
    ...
    <PostProcessing ... />

    <!-- Add this step afte the definition of the PostProcessing component and before the VCarHDRRenderer component.
         It generates an additional depth image we need in the last step -->
    <PostProcessingPipelineConfigurator name="SceneRenderer" >
        <Pipeline>
            <Step type="PPSTextureRect" name="OriginalScene" generateDepthTexture="1" depthTextureFormat="GL_DEPTH_COMPONENT24
                <Inputs>
                    <NodeInput inputNo="0" type="scene"/>
                </Inputs>
                <RTT size="viewport" bufferFormat="GL_RGB32F" />
                <SAQ lowerLeftX="0.5" lowerLeftY="0" width="0.5" height="0.5" isForDebug="1" enableSRGB="0" />
            </Step>

        </Pipeline>

    </PostProcessingPipelineConfigurator>

    <!-- The VCarHDRRenderer already defined in the configuration file has to be placed here
         rename or remove the originalSceneBufferFormat parameter and add a textureInputStep refering to the above step.
    -->
    <VCarHDRRenderer name="MyHDRRenderer"
                     ...
                     textureInputStep="OriginalScene"
                     __originalSceneBufferFormat="GL_RGB32F"
                     ...
                     />

    <!-- After the VCarHDRRenderer add the following to render the HUD and merge its outputs with that of the VCarHDRRenderer
    <PostProcessingPipelineConfigurator name="HUD Renderer" >
        <Pipeline>
            <Step type="HUD3DStep" name="RenderHUD" clearColor="1 1 1 0"  generateDepthTexture="1" depthTextureFormat="GL_DEPTH

                <RTT size="viewport" bufferFormat="GL_RGBA8" />
                <SAQ lowerLeftX="0.5" lowerLeftY="0" width="0.5" height="0.5" isForDebug="0" enableSRGB="0" />
            </Step>

            <Step type="PPSTextureRect" name="Merger" >
                <Inputs>
                    <TextureInput inputNo="0" sourceStep="FrameBufferStep"/>
                    <TextureInput inputNo="1" sourceStep="RenderHUD"/>
                    <TextureInput inputNo="2" sourceStep="OriginalScene" outputSlot="depth" />
                    <TextureInput inputNo="3" sourceStep="RenderHUD" outputSlot="depth" />
                </Inputs>
                <RTT size="viewport" bufferFormat="GL_RGB16" />
                <SAQ lowerLeftX="0" lowerLeftY="0" width="1" height="1"  isForDebug="0" enableSRGB="1" />
                <Program vertexShader="alphaBlendVert.glsl" fragmentShader="alphaDepthFrag.glsl">
                        <Uniform type="sampler2D" name="img1" value="0" />
                        <Uniform type="sampler2D" name="img2" value="1" />
                        <Uniform type="sampler2D" name="dimg1" value="2"/>
                        <Uniform type="sampler2D" name="dimg2" value="3"/>
                </Program>
            </Step>
        </Pipeline>
    </PostProcessingPipelineConfigurator>
```

The post processing now performs follwing steps:

1. Generate Textures for both color and depth of the scene
2. Perform HDR transformations on the scenes color texture
3. Generate Textures for both color and depth of the in scene Symbols (HUD3DStep)
4. Blend the HUD elements onto the scene

By adding these steps we have to accept a performance penalty of more than one millisecond per frame (depending mostly on image resolution). A simplified version without depth ordering could be performed with two thirds of the additional time.

For existing post processing pipelines containing more than just HDR different modifications can be necessary (cube map rendering, distortions, etc.). The Symbols plug-in provides the HUD3DStep and HUD3DCubeStep to mirror PPSTextureRect and PPSCubeMap.

How can I render the same frame from multiple eyepoints (e.g. for stereo images via Ethernet)

VTD allows you to define a camera list via SCP which will be used for rendering each simulation frame multiple times (once for each camera). Here's an example for rendering a scene from three camera perspectives, the first two being in the color range, the third one being a depth image. The example is an SCP script which is to be run by the scpGenerator. Note the important part `<Set target="list"/>` which makes sure that a camera definition is added to a list instead of being applied immediately. If a camera list is defined, no other camera definitions will be accepted or used.

```
         0 "<SimCtrl> <Stop/> <LoadScenario filename="../../../Projects/Current/Scenarios/traffic_demo.xml" /> <Start mode="pr
        +0 "<Camera name="1stCam" type="color"> <PosTether player="Own" distance="10.0" azimuth="0.6" elevation="0.6"/> <ViewP
        +0 "<Camera name="2ndCam" type="color"> <PosTether player="Own" distance="20.0" azimuth="0.8" elevation="0.6"/> <ViewP
        +0 "<Camera name="3rdCam" type="depth"> <PosTether player="Own" distance="2.0" azimuth="0.9" elevation="0.6"/> <ViewPl
```

Camera lists should be setup immediately at the beginning or - preferred solution - between `<Init>` and `<Start>`. So the perfect sequence for setting up camera lists is:

1. Initialize the simulation
2. Wait for `<InitDone>`
3. Configure your camera list
4. Send `<Start>`

After you have stopped the simulation, i.e. after issuing a `<Stop>` command, you will have to define the camera lists again, i.e. at point no. 1 of the above list.

If image transfer via SHM or network is enabled in the corresponding setup files, then a separate image will be streamed for each camera perspective on the same port (e.g. RDBimg).

The pedestrian animation looks "odd"

If you are using the DI-Guy pedestrians and you see them moving in the database but without animation, you may most probably have forgotten to activate the animation in the traffic module (via the general simulation settings). For the complete configuration of the DI-Guy pedestrians, please check

How can I influence the display of EGO speed and simulation time?

EGO speed and simulation time are debug options and have to be configured in the TaskControl configuration file (typically *Data/Setups/Current/Config /TaskControl/taskControl.xml*):

Example:

```
<TaskControl>
    ...
    <Debug ...
      egoSpeed="true"
      simTime="true" />
</TaskControl>
```

The same configuration settings may be performed in the *TaskControl* panel of the GUI.

The display of the debug information is realized as a symbol. Therefore, the user may modify display characteristics (e.g. color, size, position) during run-time via SCP commands. The name of the symbol is **igHudInfo**. Here's an example illustrating the modification of some parameters of the debug text:

```
<Symbol name="igHudInfo">
    <Text colorRGB="0xff0000" size="50"/>
    <PosScreen x="0.5" y="0.8"/>
</Symbol>
```

Why can't I see the mouse-cursor in the IG window?

In the standard configuration, the mouse cursor is hidden when it is located within the IG window (name: *MainRS*). The display option may be altered by

1. pressing the 'c'-key while the mouse cursor is still in the IG window
2. editing the script *Data/Setups/Current/Scripts/configureDisplay.sh* and setting the attribute *enableCursor* within the tag `<Graphics/>` to the value `1`.

If you can't find the script at the location indicated above, then it is located in *Data/Setups/* ***Common*** */Scripts/configureDisplay.sh*. Unless you really want to change the display of the mouse cursor for all setups, you should make a local copy of the script and place it in your current setup at the location given above.

Since the script is used for creating the IG configuration file *Data/Setups/Current/Config/ImageGenerator/AutoCfgDisplay.xml* all changes to it will only be applied after re-configuration of the IG (i.e. after re-loading the components or changing the setup so that the IG has to be re-started upon INIT).

How can I configure the appearance of the sky dome?

The sky dome is configured in the file `VTD/Runtime/Core/ImageGenerator/data/Sky.xml`

If you want e.g. change the lighting conditions for various times-of-day, then enter the entries for the corresponding sun elevation (-1 = midnight, 1 = noon) and the respective cloud state (off, clear etc.).

Synchronization and Triggering

Synchronization

The image generator will usually run synchronously to the graphics card. This feature is influenced by the following setting in the file `IGbase.xml`:

```
    <EnvVariable value="1" var="__GL_SYNC_TO_VBLANK" />
```

If you want the image generator to run as fast as possible, just set the above variable to `0`. Note that this may cause tearing of the image since the image generation is no longer synchronized to the double-buffering of the graphics car. Depending on the general sync settings of the simulation (i.e. if it is triggered by the image generator), also your simulation may now run at a different speed.

Triggering

The image generator will always display the last received information when performing an internal update. Per default, it is in continuous rendering mode.

If you want the image generator to render images only upon request, there are two possibilities:

1. rendering upon receipt of a taskControl message
2. rendering upon trigger via SHM

In both cases, make sure that the rest of your simulation (i.e the taskControl) is **NOT** triggered by the IG (i.e. the sync source is not "extern"). Otherwise a deadlock will occur.

For the rendering **upon receipt of a taskControl message**, you have to do the following:

- turn off the synchronization to the gfx card (see above)
- in the taskControl configuration file, set `<TaskControl> ... <ImageGenerator ... explicitTrigger="true" ... /> ... </TaskControl>`

For rendering **upon trigger via SHM**, you have to do the following:

- turn off the synchronization to the graphics card
- in the image generator configuration file (`IGbase.xml`), set `<TAKATA ... updateIGCommandsByShm="1" ... />`
- create a single-buffered RDB shared memory with id `RDB_SHM_ID_CONTROL_GENERATOR_IN`
- in the buffer, place an RDB message containing only the package `RDB_PKG_ID_SYNC`
- in this package, set the value for the member `cmdMask` to `RDB_SYNC_CMD_RENDER_SINGLE_FRAME`
- set the flags of the buffer to `RDB_SHM_BUFFER_FLAG_IG`

Repeat the last three steps each time you want the IG to render an image. The following example shall illustrate this technique. It issues trigger commands roughly every 33ms, provided that the IG is rendering images in-between. Otherwise the routine will wait until the IG has finished rendering.

```
// ShmWriter.cpp : Sample implementation of a connection to the IG's control
// SHM for triggering individual render frames.
//
// (c) 2015 by VIRES Simulationstechnologie GmbH
// Provided AS IS without any warranty!
//

#include <stdlib.h>
#include <stdio.h>
#include <sys/shm.h>
#include <string.h>
#include <unistd.h>
#include "RDBHandler.hh"

// forward declarations of methods

/**
* method for writing the commands to the SHM
*/
void openShm();
int  writeTriggerToShm();

/**
* some global variables, considered "members" of this example
*/
unsigned int mShmKey       = RDB_SHM_ID_CONTROL_GENERATOR_IN;   // key of the SHM segment
unsigned int mFrameNo      = 0;
double       mFrameTime    = 0.030;                             // frame time is 30ms
void*        mShmPtr       = 0;                                 // pointer to the SHM segment
size_t       mShmTotalSize = 64 * 1024;                         // 64kB total size of SHM segment
bool         mVerbose      = false;                             // run in verbose mode?
Framework::RDBHandler mRdbHandler;                              // use the RDBHandler helper routines to handle
                                                                // the memory and message management

/**
* information about usage of the software
* this method will exit the program
*/
void usage()
{
    printf("usage: shmWriter [-k:key]\n\n");
    printf("       -k:key        SHM key that is to be addressed\n");
    printf("       -v            run in verbose mode\n");
    exit(1);
}

/**
* validate the arguments given in the command line
*/
void ValidateArgs(int argc, char **argv)
{
    for( int i = 1; i < argc; i++)
    {
        if ((argv[i][0] == '-') || (argv[i][0] == '/'))
        {
            switch (tolower(argv[i][1]))
            {
                case 'k':        // shared memory key
                    if ( strlen( argv[i] ) > 3 )
                        mShmKey = atoi( &argv[i][3] );
                    break;

                case 'v':       // verbose mode
                    mVerbose = true;
                    break;
```

```
                default:
                    usage();
                    break;
            }
        }
    }

    fprintf( stderr, "ValidateArgs: key = 0x%x\n", mShmKey );
}

/**
* main program with high frequency loop for checking the shared memory;
* does nothing else
*/

int main(int argc, char* argv[])
{
    // Parse the command line
    //
    ValidateArgs(argc, argv);

    // first: open the shared memory (try to attach without creating a new segment)

    fprintf( stderr, "attaching to shared memory....\n" );

    while ( !mShmPtr )
    {
        openShm();
        usleep( 1000 );     // do not overload the CPU
    }

    fprintf( stderr, "...attached! Triggering now...\n" );

    // now write the trigger to the SHM for the time being
    while ( 1 )
    {
        writeTriggerToShm();

        usleep( ( unsigned int ) ( 1.e6 * mFrameTime ) );    // wait for frame time (not very precise!)
    }
}

/**
* open the shared memory segment; create if necessary
*/
void openShm()
{
    // do not open twice!
    if ( mShmPtr )
        return;

    int shmid = 0;
    int flag  = IPC_CREAT | 0777;

    // does the memory already exist?
    if ( ( shmid = shmget( mShmKey, 0, 0 ) ) < 0 )
    {
        // not yet there, so let's create the segment
        if ( ( shmid = shmget( mShmKey, mShmTotalSize, flag ) ) < 0 )
        {
            perror("openShm: shmget()");
            mShmPtr = 0;
            return;
        }
    }

    // now attach to the segment
    if ( ( mShmPtr = (char *)shmat( shmid, (char *)0, 0 ) ) == (char *) -1 )
    {
        perror("openShm: shmat()");
        mShmPtr = 0;
    }

    if ( !mShmPtr )
        return;

    struct shmid_ds sInfo;

    if ( ( shmid = shmctl( shmid, IPC_STAT, &sInfo ) ) < 0 )
        perror( "openShm: shmctl()" );
    else
        mShmTotalSize = sInfo.shm_segsz;

    // allocate a single buffer within the shared memory segment
```

```
    mRdbHandler.shmConfigure( mShmPtr, 1, mShmTotalSize );
}

int writeTriggerToShm()
{
    if ( !mShmPtr )
        return 0;

    // get access to the administration information of the first RDB buffer in SHM
    RDB_SHM_BUFFER_INFO_t* info = mRdbHandler.shmBufferGetInfo( 0 );

    if ( !info )
        return 0;

    // is the buffer ready for write?
    if ( info->flags )      // is the buffer accessible (flags == 0)?
        return 0;

    // clear the buffer before writing to it (otherwise messages will accumulate)
    if ( !mRdbHandler.shmBufferClear( 0, true ) )   // true = clearing will be forced; not recommended!
        return 0;

    fprintf( stderr, "writeTriggerToShm: sending single trigger\n" );

    // increase the frame number
    mFrameNo++;

    // create a message containing the sync information
    RDB_SYNC_t* syncData = ( RDB_SYNC_t* ) mRdbHandler.addPackage( mFrameNo * mFrameTime, mFrameNo, RDB_PKG_ID_SYNC );

    if ( !syncData )
        return 0;

    syncData->mask    = 0x0;
    syncData->cmdMask = RDB_SYNC_CMD_RENDER_SINGLE_FRAME;

    // set some information concerning the RDB buffer itself
    info->id    = 1;
    info->flags = RDB_SHM_BUFFER_FLAG_IG;

    // now copy the symc message to the first RDB buffer in SHM
    mRdbHandler.mapMsgToShm( 0 );

    return 1;
}
```

Note: the class `RDBHandler` is located in the VTD distribution at `Develop/Framework/RDBHandler`

The source code of the above example is also available at the following link:

sampleShmReaderWriterRDB.tgz

The programming example may be used as follows:

- download the archive (see above)
- unpack the archive
- go to the sub-dir `RDBShmSample/`
- execute `./compile.sh`
- start the application that consumes the data in its shared memory segment (i.e. VTD's image generator).
- execute the compiled programming example as follows:
    - `./shmWriter`

Overview of synchronization and trigger modes

Please consult FAQ - Overview of trigger and image generation modes

How can I limit the frame-rate of the ImageGenerator

In cases when you don't want the IG to run as fast as possible or as fast as your XServer allows, you may limit the maximum frame rate to a custom value. The configuration has to be done in the `<Graphics>` section of the display configuration. The corresponding attribute is `targetFramerate`.

Example:

```
<Graphics smallFeatureCullPixelSize="6"  lodscale="1" gamma="1 1 1" enableCursor="0" enableDatabasePagerThread="0" drawThreadCl
appCullThreadCPUAffinity="0" enableTransparencyAntialiasing="1" enableMultisampling="1"configurationProfile="HDR" isLocalLight
schedulerStabilization="1" targetFramerate="30">
```

Special Notes for the VIL Setup

If in a VIL setup, de-activate the sensor task for replay and preparation mode. Otherwise the viewing angles will flicker. Also, VIL setup is supported with 32bit IG only.

Light Source Configuration

Light sources are configured in BaseLights.xml which can be found in Runtime/Core/IG64/data/ . Each <LightCluster> in the XML defines a group of light sources, e.g. headlights
of a car. lightSourceGeomRadiance parameter of the <LightCluster> defines the color and intensity of the geometries that represent the given light sources. In other words, this
parameter is used to determine the color and intensity of the light source on the image, but not its effect on the environment.

Each <LightSource> section defines a light source within a light cluster. The ambient, diffuse, specular, intensity, constantAttenuation, linearAttenuation and quadraticAttenuation
parameters are used to define the effect of the light source on the environment.

The light clusters that are defined in the XML are assigned to the scene objects in <LightAssignments> section. Note that an assignment that is found in <LightAssignments> is applied
only if its description is compatible with the currently used material database. E.g. in HDR mode, only assignments with description="HDR" will be applied.

How to modify the appearance of streetlight light (Release 1.4 onwards)

Streetlights are also defined as a LightCluster. Therefore if you want to change, for example the colour of your streetlight, it is best to create a ProjectLights.xml in Data/Projects/Current/Config/ImageGenerator

High Dynamic Range Rendering

How to define the intensity of oncoming vehicle's headlights?

The intensity of oncoming vehicle's headlights is defined by the parameter *headlightLightGeomRadiance* in the configuration file *Data/Setups/Current/Config /ImageGenerator/IGbase.xml*

How to configure rendering as RGB or RedClear?

Please adjust the file *Data/Setups/Current/Config/ImageGenerator/IGbase.xml* as follows:

For Redclear

```
<VCarHDRRenderer name="MyHDRRenderer" writeVideoToShm="1" videoReadbackThreadAffinity="-1" numRenderBuffer="-1"
                 originalSceneBufferFormat="GL_RGB32F"
                 floatBufferFormat="GL_LUMINANCE32F"
                 floatBufferScaleFactor="1"
                 outputToFloatTexture="BLOOM_REDCLEAR"
                 outputToFrameBuffer="REINHARD"
                 enableSRGBOnFrameBuffer="1"
                 frameBufferScaleFactor="0.00024414" />
```

For RGB

```
<VCarHDRRenderer name="MyHDRRenderer" writeVideoToShm="1" videoReadbackThreadAffinity="-1" numRenderBuffer="-1"
                 originalSceneBufferFormat="GL_RGB32F"
                 floatBufferFormat="GL_RGB32F"
                 floatBufferScaleFactor="1"
                 outputToFloatTexture="BLOOM_LINEAR"
                 outputToFrameBuffer="REINHARD"
                 enableSRGBOnFrameBuffer="1"
                 frameBufferScaleFactor="0.00024414" />
```

Note: the attribute *name* **must always** be *MyHDRRenderer*

How to configure video stream

Video streaming in HDR mode is described in a separate chapter. Please refer to HDR video transfer.

What must be the Light Source intensity factors?

In HDR rendering, all intensities must have the same value, example:

ambientIntensity=1
diffuseIntensity=1
specularIntensity=1

If intensity is 1.0, each light source will illuminate the scene with exactly the amount of light that is measured and stored in the IES file.

Which car models are supported in HDR rendering?

The headlights of the following VTD cars have been adapted for HDR rendering. More will follow (list as of December 09, 2011):

- Mercedes_S_2009
- Smart
- Truck
- AudiA4
- AudiA3SB
- MAN_TGL
- Audi A6
- VW Golf 2010
- VW Passat Variant 11

Light Intensity (HDR version of IG only!)

The HDR vIG delivers via shared memory a data set (array) which contains *spectral RGB* values. The *luminance* in *cd/m²* can be computed from these values

as follows:

```
L = r*0.265068 + g* 0.67023428 + b *0.06409157
```

Top View

How do I change the top view camera resolution?

Resolution is set in in ProjectIGConfig1.xml und ProjectIGConfig2.xml files.

1- Cubemap resolution

Defined in:
`<Step type="PPSCubeMap" name="CubeInputLeft" >`
`<Step type="PPSCubeMap" name="CubeInputRight" >`

with following line:

```
<RTT size="1024" bufferFormat="GL_RGB8" />
```

"size" parameter must be changed to change the cube map resolution.

2- Top View Camera resolution:

Defined in:
`<Step type="PPSTextureRect" name="LensLeft" >`
`<Step type="PPSTextureRect" name="LensRight" >`

with following line:

```
<RTT sizeMode="viewport" bufferFormat="GL_RGB8" />
```

This line must be replaced as follows:

```
<RTT sizeMode="explicit" width="1280" height="800" bufferFormat="GL_RGB8" />
```

Now width and height can be adjusted.

3- Shared Memory buffer Resolution:
Defined in:
`<Step type="PPSTextureRect" name="FinalImage">`

with following line:

```
<RTT sizeMode="explicit" width="1280" height="1600" bufferFormat="GL_RGB8" />
```

This buffer contains the outputs of two cameras hence must have twice the height. "width" must be the same as top view camera widths and "height" must be twice as large. Moreover, u_inputImageHeight parameter must be the same as the top view camera height (**not** SHM buffer height)

How do I change position and orientation of top view cameras?

These are set in ProjectIGConfig1.xml und ProjectIGConfig2.xml.

Position and orientation is determined in

```
<Step type="PPSCubeMap" name="CubeInputLeft">
<Step type="PPSCubeMap" name="CubeInputRight">
```

in

`<CameraConfiguration>` with "xyz" and "hpr" parameters. The parameters are relative to the eye position.

FAQ

RDBInterface

RDB shared memory layout concept

Any communication to an external process via network or shared memory is handled by the RDBInterface component RDBInterface.

Custom shared memory keys

Define your own shared memory keys within the config files:

```
    <RDBInterface name="MyRDBInterface" printDebugInfo="0" ignoreShmFlags="0">
        <ShmLayoutImage key="0x0616a" freeFlag="0x00" releaseFlag="0x0D" />
        <ShmLayoutLightmap key="0x0666a"/>
        <ShmLayoutIGControl key="0x0555a"/>
        <ShmLayoutCustomImage key="0x0444a"/>
        <ShmLayoutGeneric key="0x0333a"/>
        <ShmLayoutIGFrameTrigger key="0x0222a"/>
        <ShmLayoutOptiXW key="0x0111a"/>
        <ShmLayoutOptiXR key="0x0000a"/>
    </RDBInterface>
```

Material System

Material system concept for LDR, HDR and HCS

All materials for the different setups of LDR, HDR and HCS are managed by the Material System.

Headless Mode

IG without window

Supported with VTD.2.0.3. In order to start the ImageGenerator without a window, add the attribute `pbuffer="1"` to the `RenderSurface` of the IGConfig. This definition is normally located in the setup within *Config/ImageGenerator/AutoCfgDisplay.xml* or overwritten with *Scripts/configureDisplay.sh*. In this way the IG renders to the pixelbuffer instead of the framebuffer and no window is created.

```
<IGconfig>
   <SystemConfig>
     <Graphics smallFeatureCullPixelSize="3.9"  lodscale="0.9" gamma="1 1 1"  enableCursor="0" enableDatabasePagerThread="0" d
       <RenderSurface name="mainRS" x="0" y="600" width="4096" height="4096" depthBits="24" sampleBuffers="0" samples="0" bord
       <Camera name="cam1" renderSurface="mainRS" viewPortX="0" viewPortY="0" viewPortWidth="4096" viewPortHeight="4096">
         <SymmetricPerspectiveAngles  fovX="45.00" fovY="45.00" near="1" far="1500" offsetHPR="0 0 0" offsetXYZ="0 0.0 0"/>
       </Camera>
     </Graphics>
   </SystemConfig>
 </IGconfig>
```

Rendering to Shared Memory

The shared memory writing is already implemented and enabled with the postprocessing pipeline, which should be added to the `IGbase.xml` in the setup:

```
    <RDBInterface name="MyRDBInterface" printDebugInfo="0" ignoreShmFlags="0" />

    <PostProcessing name="MyPostProcessing" />

    <PostProcessingPipelineConfigurator name="MyPostProcessingPipelineConfigurator">
      <Pipeline hideSceneFromDefaultView="1" >
    <Step type="PPSTextureRect" name="OriginalScene" generateDepthTexture="1" >
      <Inputs>
        <NodeInput enabled="1" renderLights="1" inputNo="0" type="scene" />
      </Inputs>
      <RTT  sizeMode="viewport"  bufferFormat="GL_RGB8" />
      <SAQ lowerLeftX="0" lowerLeftY="0" width="1" height="1" isForDebug="0" enableSRGB="0" />
      <Readback readbackThreadAffinity="3" >
            <Image outputSlot="color" />
            <Image outputSlot="depth" />
          </Readback>
    </Step>
      </Pipeline>
    </PostProcessingPipelineConfigurator>
```

Writing to shared memory via Post Processing is only supported for a single view configuration. The cpu id for the writing thread is defined by `readbackThreadAffinity="3"`.

Performance Monitoring

OpenSceneGraph Mode

You may toggle between different performance statistics modes provided by OpenSceneGraph by pressing the key "s" while locating the cursor in the IG render window.

v-IG Mode

The IG's internal mode for performance monitoring is - per default - configured in the file `Data/Setups/Common/Config/ImageGenerator/CommonIGConfig.xml`

Example:

```
<Statistics name="MyPerfStats" componentPerfStatsFrameCount="0" graphics="1" appDetails="1" frameRate="1" camDetails="1" print
    <FrameTime history="500" />
    <ComponentDeletionTime history="500"  />
    <ApplicationTime history="500" />
    <GuiProcessingTime history="500"  />
    <PreUpdateCallsTime history="500" />
    <PreFrameCallsTime history="500" />
    <PostFrameCallsTime history="500" />
    <SceneGraphUpdateTime history="500" />
    <SceneGraphCullTimeView0 history="500" />
    <SceneGraphDrawTimeView0 history="500" />
    <Drawables history="500" />
</Statistics>
```

In order to enable the statistics, make sure the entry is active and the attribute *showOnScreen* is set to *1*.

The tags and their attributes are

- *Statistics*: top level tag
  - *name*: must be "MyPerfStats"
  - *componentPerfStatsFrameCount*: enables IGComponent statistics for preUpdate, preFrame and postFrame calls. Collected over the given number of frames and printed to console.
  - *graphics*: Boolean value that enables statistics collected from OpenSceneGraph, including times spend during event, update and rendering traversal as well as GPU usage.
  - *appDetails*: ???. Boolean value, may be unused.
  - *frameRate*: Boolean value, show frame rate using OpenSceneGraph statistics.
  - *camDetails*: Boolean value, collect statistics on visible geometry each frame.
  - *printToConsoleFrameTime*: Boolean value, prints average frame rate to console. Based on FrameTime data.
  - *showOnScreen*: show statistics on-screen if set to "1"
- *FrameTime*: Time between two ImageGenerator::synch() calls.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *ComponentDeletionTime*: Time spend deleting IGComponents.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *ApplicationTime*: Time spend between synch() and frame() calls, should currently be near zero for vigcar.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *GuiProcessingTime*: Time spend uploading already queued resources and processing UI events.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *PreUpdateCallsTime*: Time spend in IGComponent::preUpdate() calls.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *PreFrameCallsTime*: Time spend in IGComponent::preFrame() calls.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *PostFrameCallsTime*: Time spend in IGComponent::postFrame() calls.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *SceneGraphUpdateTime*: Time spend in the OpenSceneGraph update traversal.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *SceneGraphCullTimeView0*: Time spend in the cull traversal of view 0
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *SceneGraphDrawTimeView0*: Time spend in the render traversal of view 0
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.
- *Drawables*: Number of visible drawables.
  - *history*: internally held number of measurement points
  - *enableGraph*: show graph for this entry.

Image Formats

The following list provides a mapping between RDB_PIX_FORMAT definitions and the better-known OpenGL or OSC definitions

tbd.

RDB_PIX_FORMAT	OpenGL format	Description
RDB_PIX_FORMAT_RED8	GL_R8	1 Channel red, 8 bit [0, 255]
RDB_PIX_FORMAT_RED16	GL_R16	1 Channel red, 16 bit [0, 65535]
RDB_PIX_FORMAT_RED24	-	1 Channel red, 24 bit [0, 16777215]
RDB_PIX_FORMAT_RED32	GL_R32UI	1 Channel red, 32 bit [0, 4294967295]]
RDB_PIX_FORMAT_RED16F	GL_R16F	1 Channel red, half precision float
RDB_PIX_FORMAT_RED32F	GL_R32F, GL_LUMINANCE32F_ARB	1 Channel red, 32 bit single precision float (IEEE 754)
RDB_PIX_FORMAT_RG8	GL_RG8	2 Channel red, green 8 bit each channel [0, 255]
RDB_PIX_FORMAT_RG16	GL_RG16	2 Channel red, green 16 bit each channel [0, 65535]
RDB_PIX_FORMAT_RG32	GL_RG32UI	2 Channel red, green 32 bit each channel [0, 4294967295]]
RDB_PIX_FORMAT_RG16F	GL_RG16F	2 Channel red, green half precision float
RDB_PIX_FORMAT_RG32F	GL_RG32F	2 Channel red, green single precision float
RDB_PIX_FORMAT_DEPTH8	-	1 Channel depth, 8 bit [0, 255]
RDB_PIX_FORMAT_DEPTH16	GL_DEPTH_COMPONENT16	1 Channel depth, 16 bit [0, 65535]
RDB_PIX_FORMAT_DEPTH24	GL_DEPTH_COMPONENT24	1 Channel depth, 24 bit [0, 16777215]
RDB_PIX_FORMAT_DEPTH32	GL_DEPTH_COMPONENT32	1 Channel depth, 32 bit [0, 4294967295]
RDB_PIX_FORMAT_RGB8	GL_RGB8	3 Channel red, green, blue , 8 bit each channel [0, 255]
RDB_PIX_FORMAT_RGB16	GL_RGB16UI	3 Channel red, green, blue , 16 bit each channel [0, 65535]
RDB_PIX_FORMAT_RGB32	GL_RGB32UI	3 Channel red, green, blue , 32 bit each channel [0, 4294967295]
RDB_PIX_FORMAT_RGB16F	GL_RGB16F	3 Channel red, green, blue, half precision float each
RDB_PIX_FORMAT_RGB32F	GL_RGB32F_ARB	3 Channel red, green, blue single precision float each

RDB_PIX_FORMAT_RGBA8	GL_RGBA8	4 Channel red, green, blue, alpha , 8 bit each channel [0, 255]
RDB_PIX_FORMAT_RGBA16	GL_RGBA16	4 Channel red, green, blue, alpha, 16 bit each channel [0, 65535]
RDB_PIX_FORMAT_RGBA32	GL_RGBA32UI	4 Channel red, green, blue, alpha, 32 bit each channel [0, 4294967295]
RDB_PIX_FORMAT_RGBA32F	GL_RGBA32F_ARB	4 Channel red, green, blue, alpha, single precision float each
RDB_PIX_FORMAT_RGBA16F	GL_RGBA16F	4 Channel red, green, blue, alpha, half precision float each
RDB_PIX_FORMAT_RGB_32	-	tbd.
RDB_PIX_FORMAT_RGBA_32	-	tbd.
RDB_PIX_FORMAT_R3_G2_B2	GL_R3_G3_B2	3 Channel red, green, blue. Red, green 3 bit each [0, 7], blue 2 bit [0, 3]
RDB_PIX_FORMAT_R3_G2_B2_A8	GL_R3_G3_B2_A8	4 Channel red, green, blue, alpha. Red, green 3 bit each [0, 7], blue 2 bit [0, 3], alpha 8 [0, 255]
RDB_PIX_FORMAT_R5_G6_B5	GL_RGB565	3 Channel red, green, blue. Red, blue 5 bit each [0, 31], green 6 bit [0, 63]
RDB_PIX_FORMAT_R5_G6_B5_A16	-	4 Channel red, green, blue, alpha. Red, blue 5 bit each [0, 31], green 6 bit [0, 63], alpha 16 [0, 65535]
RDB_PIX_FORMAT_RGB8_A24	-	4 Channel red, green, blue, alpha. Red, green and blue 8 bit each [0, 255], alpha 24 bit [0, 16777215]
RDB_PIX_FORMAT_CUSTOM_01	-	User defined format
RDB_PIX_FORMAT_CUSTOM_02	-	User defined format
RDB_PIX_FORMAT_CUSTOM_03	-	User defined format

The shared memory read-back implemented in the IG PostProcessing Component may not use the corresponding OpenGL/RDB_PIX_FORMAT to write images. Deviations are listed below:

OpenGL	RDB_PIX_FORMAT
GL_LUMINANCE16F_ARB	RDB_PIX_FORMAT_RED32F
GL_DEPTH_COMPONENT24	RDB_PIX_FORMAT_DEPTH32
GL_RGB16F_ARB	RDB_PIX_FORMAT_RGB32F
GL_RGB16AF_ARB	RDB_PIX_FORMAT_RGBA32F

Formats currently supported by the IG Post Processing shared memory read-back:

- GL_R16
- GL_R32F
- GL_LUMINANCE32F_ARB
- GL_RGB8
- GL_RGB32F
- GL_RGBA8
- GL_RGBA16
- GL_RGBA32F
- GL_DEPTH_COMPONENT24
- GL_DEPTH_COMPONENT32

Formats currently used by network read-back:

- RDB_PIX_FORMAT_DEPTH32
- RDB_PIX_FORMAT_RGB8

Formats currently correctly used by the shared memory RDBVideoTexture symbol:

- RDB_PIX_FORMAT_RGB (see RDB_PIX_FORMAT_RGB8)
- RDB_PIX_FORMAT_RGB8
- RDB_PIX_FORMAT_RGBA8
- RDB_PIX_FORMAT_BW_8 (see RDB_PIX_FORMAT_RED8)

Lightmap update

- RDB_PIX_FORMAT_RGBA8 handled as if it was RDB_PIX_FORMAT_RED32F, multiplied with the image header colour field.

Output formats supported for OptiX

OptiX type	RDB_PIX_FORMAT
RT_FORMAT_FLOAT (half)	RDB_PIX_FORMAT_RED16F
RT_FORMAT_FLOAT (single)	RDB_PIX_FORMAT_RED32F
RT_FORMAT_FLOAT2 (half)	RDB_PIX_FORMAT_RG16F
RT_FORMAT_FLOAT2 (single)	RDB_PIX_FORMAT_RG32F
RT_FORMAT_FLOAT3 (half)	RDB_PIX_FORMAT_RGB16F
RT_FORMAT_FLOAT3 (single)	RDB_PIX_FORMAT_RGB32F
RT_FORMAT_FLOAT4 (half)	RDB_PIX_FORMAT_RGBA16F
RT_FORMAT_FLOAT4 (single)	RDB_PIX_FORMAT_RGBA32F
RT_FORMAT_UNSIGNED_BYTE	RDB_PIX_FORMAT_RED8

RT_FORMAT_BYTE	RDB_PIX_FORMAT_RED8
RT_FORMAT_UNSIGNED_BYTE2	RDB_PIX_FORMAT_RG8
RT_FORMAT_BYTE2	RDB_PIX_FORMAT_RG8
RT_FORMAT_UNSIGNED_BYTE3	RDB_PIX_FORMAT_RGB8
RT_FORMAT_BYTE3	RDB_PIX_FORMAT_RGB8
RT_FORMAT_UNSIGNED_SHORT	RDB_PIX_FORMAT_RED16
RT_FORMAT_SHORT	RDB_PIX_FORMAT_RED16
RT_FORMAT_UNSIGNED_SHORT2	RDB_PIX_FORMAT_RG16
RT_FORMAT_SHORT2	RDB_PIX_FORMAT_RG16
RT_FORMAT_UNSIGNED_SHORT3	RDB_PIX_FORMAT_RGB16
RT_FORMAT_SHORT3	RDB_PIX_FORMAT_RGB16
RT_FORMAT_UNSIGNED_SHORT4	RDB_PIX_FORMAT_RGBA16
RT_FORMAT_SHORT4	RDB_PIX_FORMAT_RGBA16
RT_FORMAT_UNSIGNED_INT	RDB_PIX_FORMAT_RED32
RT_FORMAT_INT	RDB_PIX_FORMAT_RED32
RT_FORMAT_UNSIGNED_INT2	RDB_PIX_FORMAT_RG32
RT_FORMAT_INT2	RDB_PIX_FORMAT_RG32
RT_FORMAT_UNSIGNED_INT3	RDB_PIX_FORMAT_RGB32
RT_FORMAT_INT3	RDB_PIX_FORMAT_RGB32
RT_FORMAT_UNSIGNED_INT4	RDB_PIX_FORMAT_RGBA32
RT_FORMAT_INT4	RDB_PIX_FORMAT_RGBA32
RT_FORMAT_USER	RDB_PIX_FORMAT_CUSTOM_01

viewingAngles.png (282 KB) Marius Dupuis, 05.08.2012 08:46
guiSnapshot01.png (26.7 KB) Marius Dupuis, 29.10.2012 13:27
ProjectLights.xml - Example for bright green streetlights (607 Bytes) Esther Hekele, 05.05.2014 14:52
FiguresManualO3.png (56.7 KB) Marius Dupuis, 27.07.2014 09:51
dualIGDualPlayer.tgz (14 KB) Marius Dupuis, 02.01.2015 10:49
crossing8DualExt.xml (7.33 KB) Marius Dupuis, 02.01.2015 11:15
vigDefaultConfigurationFiles.png (62.2 KB) Daniel Wiesenhuetter, 16.08.2018 13:33
vigMultiConfigurationFiles.png (88 KB) Daniel Wiesenhuetter, 16.08.2018 13:33
DualIGDualPlayer.tgz (11.6 KB) Stefan Fink, 25.10.2018 16:32