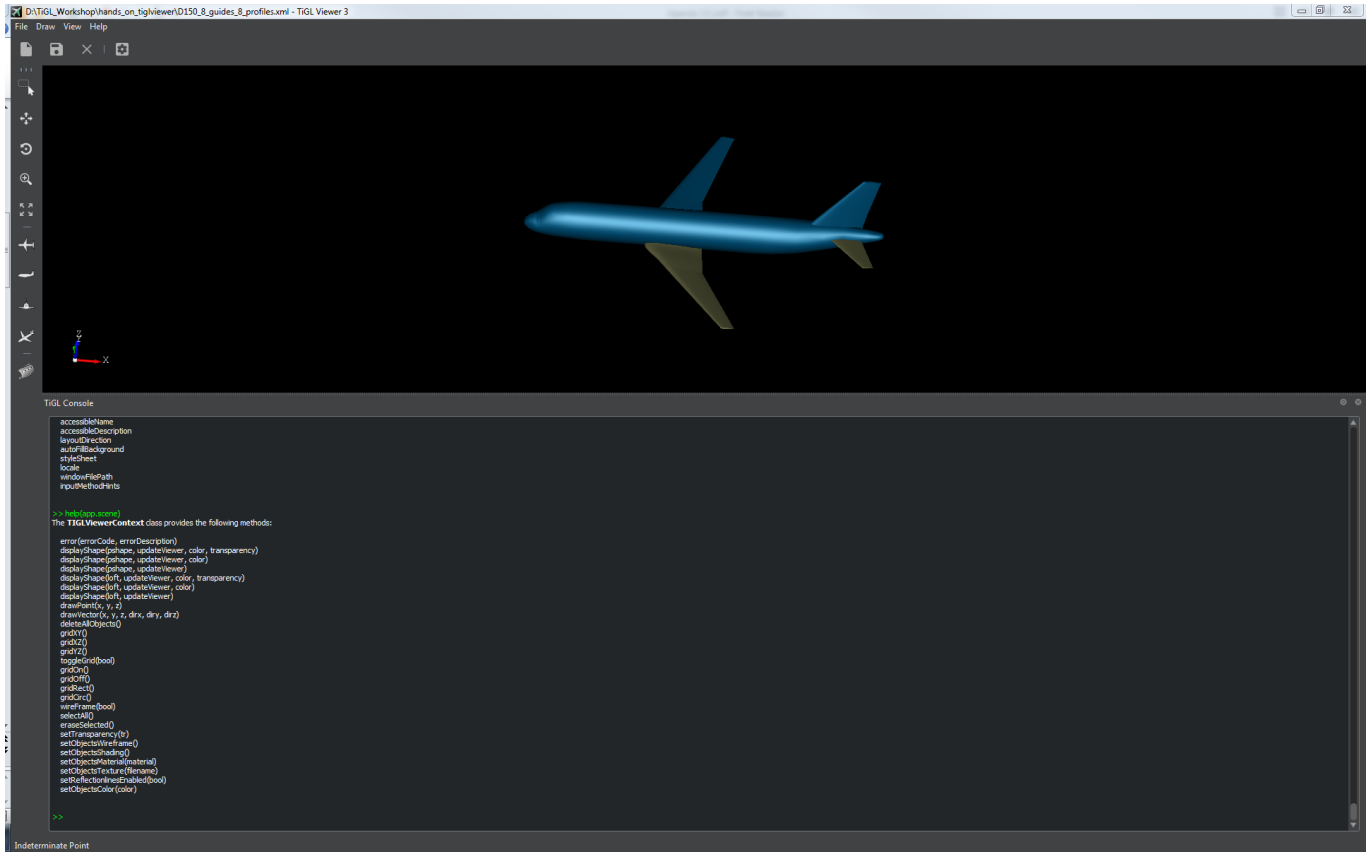
1. TiGL Workshop – TiGL Viewer Scripting API + Hands On

Jan Kleinert, Martin Siggel

September 11./12. 2018, Cologne

Contents

Introduction to the TiGL Viewer Scripting API



Hands On

1. CAD Export with customized export options
2. Display shapes by UID and make screenshots
3. Make a video with camera movement

Final Remarks

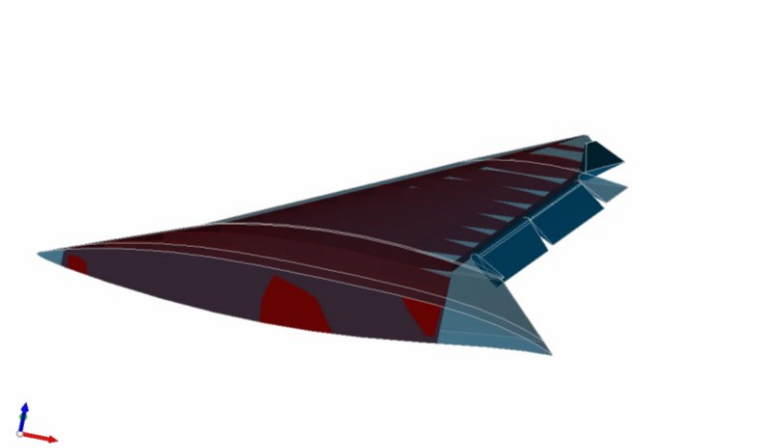
Introduction to the TiGL Viewer Scripting API

Introduction to the TiGL Viewer API

- The TiGL Viewer comes with a separate scripting API based on **JavaScript**

It can be used for

- Manipulating the visualization and creating screenshots and videos
- Drawing (sub-) shapes, points and vectors
- Accessing some of the basic TiGL API-functions



How to use the TiGL Viewer Scripting API

There are four ways to use the TiGL Viewer Scripting API

1. Open the Console in TiGL Viewer (**Alt+C** or **View → Display → Console**) and enter commands

2. Open the Console in TiGL Viewer (**Alt+C** or **View → Display → Console**) and enter:

```
>> app.openScript("myAwesomeTiGLViewerScript.js")
```

3. Press **File → Open Script** and select a javascript file

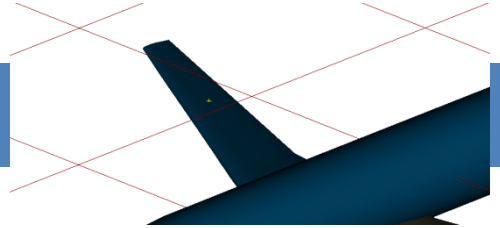
4. Directly start TiGL Viewer from a terminal/command-prompt with a javascript file:

```
$ D:\tiglhome\tiglviewer-3.exe --script myAwesomeTiGLViewerScript.js
```


Some examples how to use the TiGL Viewer Scripting API:

- Compute and draw point on wing:

```
>> p = tigl.wingGetUpperPoint(1, 3, 0.5, 0.5);
>> drawPoint(p);
```

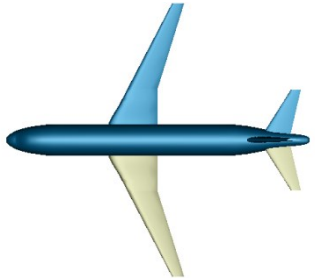


- Draw the first wing:

```
>> uid = tigl.wingGetUID(1);
>> drawShape(tigl.getShape(uid));
```

- Show top view and fit screen:

```
>> app.viewer.viewTop();
>> app.viewer.fitAll();
```



- Make a screenshot

```
>> app.viewer.makeScreenshot('image.png'); //PNG image, white background
>> app.viewer.makeScreenshot('image.jpg', false); //JPEG image, current background
```

Hint: *Type* `help(tigl)` *to get a list of available TiGL functions.*
Hint: *Type* `help` *to display* ***this information*** *in the Console.*

tigl, app, viewer and scene

- The API consists of **four javascript objects** and a few **global definitions** *(e.g. a Point3d class)*

`tigl`

- A selection of the basic TiGL API functions

`app`

- Open CPACS files or TiGLViewer Scripts, close the application, get the `viewer` and `scene` objects
- Derived from Qt's `QMainWindow` class

`app.viewer`

- Change the camera position, zoom, pan, make screenshots…
- Derived from Qt's `QWidget` class

`app.scene`

- Display objects, turn grid on and off, change the objects appearance, draw points and vectors, …

The TiGL Interface `tigl` class provides the following methods:

getWingCount()
getVersion()
componentGetHashCode(componentUID)
exportComponent(uid, filename, deflection)
exportConfiguration(filename, fuseAllShapes, deflection)
exportFusedWingFuselageIGES(filename)
exportIGES(filename)
exportSTEP(filename)
exportFusedSTEP(filename)
exportMeshedFuselageSTL(fuselageIndex, filename, deflection)
exportMeshedFuselageVTKByIndex(fuselageIndex, filename, deflection)
exportMeshedFuselageVTKByUID(fuselageUID, filename, deflection)
setExportOptions(exporter, optionName, optionValue)
fuselageGetUID(fuselageIndex)
fuselageGetCircumference(fuselageIndex, segmentIndex, eta)
fuselageGetPoint(fuselageIndex, segmentIndex, eta, zeta)
fuselageGetSegmentUID(fuselageIndex, segmentIndex)
fuselageGetSegmentVolume(fuselageIndex, segmentIndex)
getFuselageCount()
fuselageGetSegmentCount(fuselageIndex)
wingGetUpperPoint(wingIndex, segmentIndex, eta, xsi)
wingGetUID(wingIndex)
wingGetLowerPoint(wingIndex, segmentIndex, eta, xsi)
wingGetUpperPointAtDirection(wingIndex, segmentIndex, eta, xsi, dirx, diry, dirz)
wingGetLowerPointAtDirection(wingIndex, segmentIndex, eta, xsi, dirx, diry, dirz)
wingGetChordPoint(wingIndex, segmentIndex, eta, xsi)
wingGetChordNormal(wingIndex, segmentIndex, eta, xsi)
wingComponentSegmentGetPoint(compSegUID, eta, xsi)
wingGetSegmentCount(wingIndex)
wingGetSegmentUID(wingIndex, segmentIndex)
wingGetSpan(wingUID)
wingGetSegmentVolume(wingIndex, segmentIndex)
getErrorString(errorCode)
getShape(uid)

The following properties are defined:

version

```
>> help(tigl)
```

The TiGL Viewer Application class app provides the following methods:

windowInitialized()
openFile(fileName)
openScript(scriptFileName)
saveFile(fileName)
closeConfiguration()
getViewer()
getScene()
getDocument()

The following properties are defined:

viewer
scene

+ some additional Qt related functions and properties

```
>> help(app)
```


The TiGL Viewer Application class `app.viewer` provides the following methods:

initialized()
selectionChanged()
mouseMoved(X, Y, Z)
pointClicked(X, Y, Z)
sendStatus(aMessage)
error(errorCode, errorDescription)
idle()
fitExtents()
fitAll()
fitArea()
zoom()
zoomIn()
zoomOut()
zoom(scale)
pan()
globalPan()
rotation()
selecting()
hiddenLineOn()
hiddenLineOff()

setBackgroundGradient(r, g, b)
setBackgroundColor(r, g, b)
setBGImage(QString)
viewFront()
viewBack()
viewTop()
viewBottom()
viewLeft()
viewRight()
viewAxo()
viewTopFront()
viewGrid()
viewReset()
setLookAtPosition(x, y, z)
setCameraPosition(x, y, z)
setCameraPosition(elevationAngleDegree, azimuthAngleDegree)
setCameraUpVector(x, y, z)
setReset()
setTransparency()
setTransparency(int)

setObjectsColor()
setObjectsMaterial()
setObjectsTexture()
makeScreenshot(filename, whiteBGEnabled, width, height, quality)
makeScreenshot(filename, whiteBGEnabled, width, height)
makeScreenshot(filename, whiteBGEnabled, width)
makeScreenshot(filename, whiteBGEnabled)
makeScreenshot(filename)

+ some additional Qt related functions and properties

```
>> help(app.viewer)
```

The TiGL Viewer Application class app.scene provides the following methods:

error(errorCode, errorDescription)
displayShape(pshape, updateViewer, color, transparency)
displayShape(pshape, updateViewer, color)
displayShape(pshape, updateViewer)
displayShape(loft, updateViewer, color, transparency)
displayShape(loft, updateViewer, color)
displayShape(loft, updateViewer)
drawPoint(x, y, z)
drawVector(x, y, z, dirx, diry, dirz)
deleteAllObjects()
gridXY()
gridXZ()
gridYZ()
toggleGrid(bool)
gridOn()
gridOff()
gridRect()
gridCirc()

wireFrame(bool)
selectAll()
eraseSelected()
setTransparency(tr)
setObjectsWireframe()
setObjectsShading()
setObjectsMaterial(material)
setObjectsTexture(filename)
setReflectionlinesEnabled(bool)
setObjectsColor(color)

```
>> help(app.scene)
```

CAD Exports

The standard TiGL API functions can be used for CAD exports

- **New in TiGL 3:** You can set options for the exporter using

```
setExportOptions(exporter, optionName, optionValue)
```

- `exporter` is one of “`brep`“, “`collada`“, “`iges`“, “`step`“, “`stl`“, “`vtk`“
- `optionName` is one of “`IsDefault`”, “`ApplySymmetries`”, “`IncludeFarfield`”, …
- `optionValue` depends on the optionName (can be bool, int, float)

- The different exporters have different additional options and different default values.

Default values for the CAD export options

	BREP	Collada	IGES	STEP	STL	VTK
ApplySymmetries	false	true	false	false	true	true
IncludeFarfield	false	false	true	true	false	false
ShapeGroupMode	WHOLE_SHAPE	WHOLE_SHAPE	NAMED_COMPOUNDS	NAMED_COMPOUNDS	WHOLE_SHAPE	WHOLE_SHAPE
IGES5.3	x	x	true	x	x	x
WriteNormals	x	x	x	x	x	true
MultiplePieces	x	x	x	x	x	false
WriteMetaData	x	x	x	x	x	true

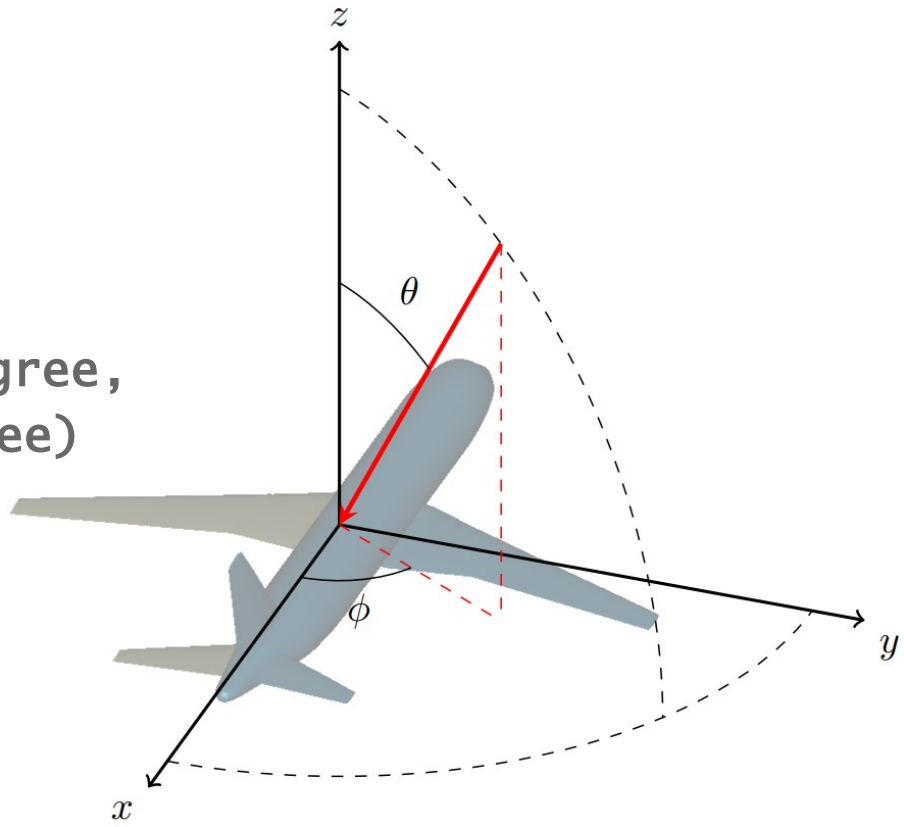
Camera Positioning

Camera Positioning can be manipulated via

```
app.viewer.setLookAtPosition(x,y,z)
app.viewer.setCameraUpVector(x,y,z)
```

and either

```
app.viewer.setCameraPosition(x, y, z)
app.viewer.setCameraPosition(elevationAngleDegree,
                             azimuthAngleDegree)
```



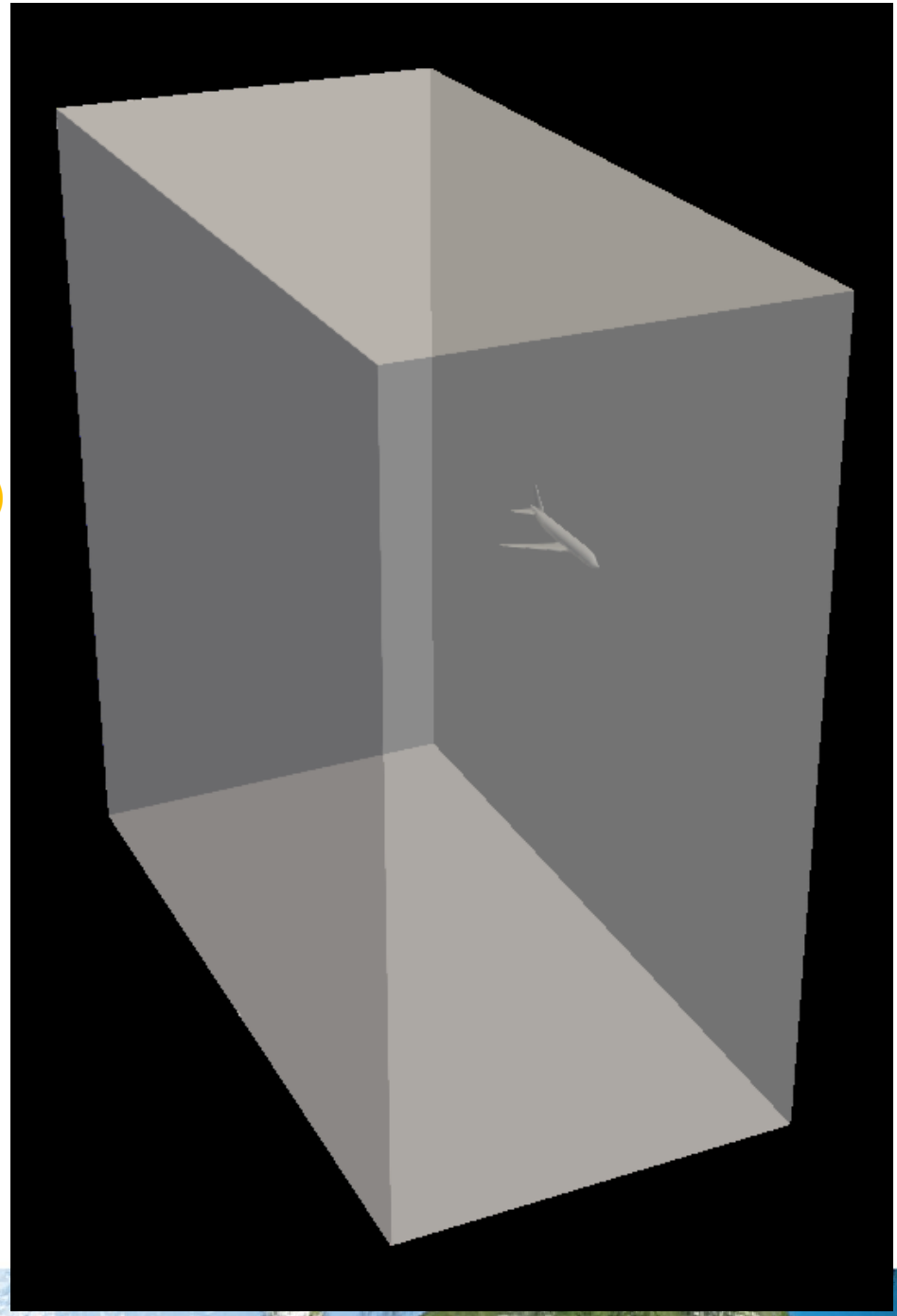
- Setting Camera Position using **spherical coordinates** keeps the distance to the LookAtPosition constant
- Create animations using a for-loop:

```
for (i = 0; i < numFrames; i += 1) {
   // reset theta and phi here
   app.viewer.setCameraPosition(theta, phi)
   app.viewer.makeScreenshot("frame_" + i + ".jpg");
}
```

Hands On

Task 1: Customized VTK export

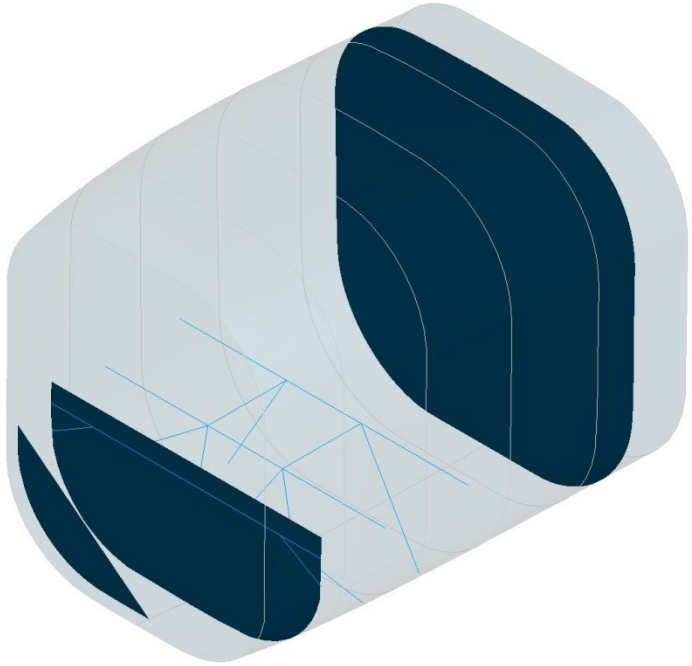
1. Write a script that
   a. Opens the file “D150_farField.xml”,
      (get the file from here: https://goo.gl/tMHMU3)
   b. Sets the options for the VTK-export, so that the far field is included and symmetries are not applied,
   c. Exports the configuration to a vtk file using a maximum deflection of 0.1 for the triangulation,
   d. Closes TiGLViewer.



Hint:

- Start TiGlViewer from the console in the directory of the input file using the “--script” option.

Task 2: Display subshapes and make a screenshot



2. Write a script that
   a. Opens the CPACS file “fuselage_structure-v3.xml”,
      (get the file from here: https://goo.gl/vm4HXp)
   b. Sets the transparency of all objects to 90
   c. Displays the shapes with the following uids

"cargoCrossBeam1"
"cargoCrossBeam2"
"cargoCrossBeam3"

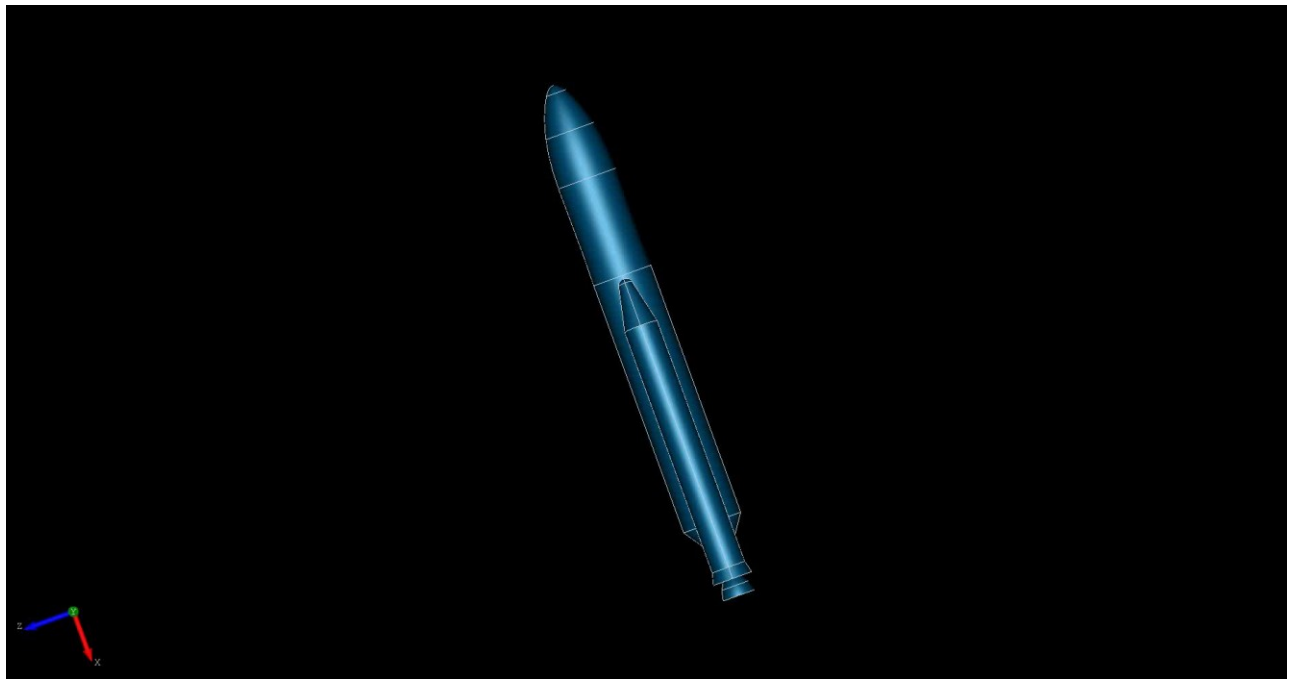
"cargoCrossBeamStrut1"
"cargoCrossBeamStrut2"
"cargoCrossBeamStrut3"
"cargoCrossBeamStrut4"
"cargoCrossBeamStrut5"
"cargoCrossBeamStrut6"
"cargoCrossBeamStrut7"
"cargoCrossBeamStrut8"

"half_bulkhead"
"holed_bulkhead„
"simple_bulkhead"

"longFloorBeam1"
"longFloorBeam2"

   d. Makes a screenshot

Task 3: Make an animated video



3. Replicate the Ariane-5 video from this slide. To do so,

   a. Unzip the file ariane.zip from the exercise directory.
   b. Fill in the missing code in the ariane-movie.js script *(search for "EXERCISE" in the comments)*

   c. (OPTIONAL) If you have FFMPEG (https://www.ffmpeg.org/) installed, open a terminal/command-prompt and type

```
ffmpeg -r 25 -f image2 -s 1874x816 -I image_04d.jpg -vcodec libx264 -crf 15
-pix_fmt yuv420p ariane.mp4
```

(see https://goo.gl/SW8mZR)

Final Remarks

Final Remarks

- The main purpose of the TiGL Viewer Scripting API is quick debugging and customized visualization
- We add new functionality every now and then as new applications come to our minds

- **Outlook:**
  - Manipulate individual objects in the scene
  - Change material and color of the objects

- If you use the TiGL Viewer Scripting API and find that useful functions are missing, please let us know!

Questions

jan.kleinert@dlr.de
martin.siggel@dlr.de