Open CASCADE Technology 7.0.0

Upgrade from older OCCT versions

April 4, 2016

Contents

1 Introduction

This document provides technical details on changes made in particular versions of OCCT. It can help to upgrade user applications based on previous versions of OCCT to newer ones.

1.1 Precautions

Back-up your code before the upgrade. We strongly recommend using version control system during the upgrade process and saving one or several commits at each step of upgrade, until the overall result is verified. This will facilitate identification and correction of possible problems that can occur at the intermediate steps of upgrade. It is advisable to document each step carefully to be able to repeat it if necessary.

1.2 Disclaimer

This document describes known issues that have been encountered during porting of OCCT and some applications and approaches that have helped to resolve these issues in known cases. It does not pretend to cover all possible migration issues that can appear in your application. Take this document with discretion; apply your expertise and knowledge of your application to ensure the correct result.

The automatic upgrade tool is provided as is, without warranty of any kind, and we explicitly disclaim any liability for possible errors that may appear due to use of this tool. It is your responsibility to ensure that the changes you made in your code are correct. When you upgrade the code by an automatic script, make sure to carefully review the introduced changes at each step before committing them.

2 Upgrade to OCCT 6.5.0

Porting of user applications from an earlier OCCT version to version 6.5 requires taking into account the following major changes:

- If you are not comfortable with dependence on Intel TBB, FreeImage, or Gl2Ps libraries, you will need to (re)build OCCT with these dependencies disabled.
- The low-level format version of OCAF binary and XML persistence has been incremented. Hence, the files saved by OCCT 6.5 to OCAF binary or XML format will not be readable by previous versions of OCCT.
- The *BRepMesh* triangulation algorithm has been seriously revised and now tries hard to fulfill the requested deflection and angular tolerance parameters. If you experience any problems with performance or triangulation quality (in particular, display of shapes in shading mode), consider revising the values of these parameters used in your application.
- If you were using method *ToPixMap()* of class *V3d_View* to get a buffer for passing to Windows API functions (e.g. *BitBlt*), this will not work anymore. You will need to use method *Image_PixMap::AccessBuffer()* to get the raw buffer data that can be further passed to WinAPI functions.
- As the processing of message gravity parameter in *Message* package has been improved, some application messages (especially the ones generated by IGES or STEP translators) can be suppressed or new messages appear in the application. Use relevant message level parameter to tune this behavior.

3 Upgrade to OCCT 6.5.1

Porting of user applications from an earlier OCCT version to version 6.5.1 requires taking into account the following major changes:

- Method *Graphic3d_Structure::Groups()* now returns *Graphic3d_SequenceOfGroup*. If this method has been used, the application code should be updated to iterate another collection type or, if *Graphic3d_HSetOfGroup* is required, to fill its own collection:

```
const Graphic3d_SequenceOfGroup& aGroupsSeq = theStructure.Groups();
Handle(Graphic3d_HSetOfGroup) aGroupSet = new Graphic3d_HSetOfGroup();
Standard_Integer aLen = aGroupsSeq.Length();
for (Standard_Integer aGr = 1; aGr <= aLen; ++aGr)
{
 aGroupSet->Add (aGroupsSeq.Value (aGr));
}
```

- All occurrences of *Select3D_Projector* in the application code (if any) should be replaced with *Handle(Select3-D_Projector)*.
- The code of inheritors of *Select3D_SensitiveEntity* should be updated if they override *Matches()* (this is probable, if clipping planes are used).
- Constructor for *V3d_Plane* has been changed, so the extra argument should be removed if used in the application. It is necessary to add a new plane using method *V3d_Viewer::AddPlane()* if *V3d_Viewer* has been used to manage clipping planes list (this does not affect clipping planes representation). Please, have a look at the source code for new DRAWEXE *vclipplane* command in *ViewerTest_ObjectsCommands.cxx, VClipPlane* to see how clipping planes can be managed in the application.

4 Upgrade to OCCT 6.5.2

Porting of user applications from an earlier OCCT version to version 6.5.2 requires taking into account the following major changes:

- Any code that has been generated by WOK from CDL generic classes *Tcollection_DataMap* and *Tcollection-_IndexedDataMap* needs to be regenerated by WOK to take into account the change in the interface of these classes.
- The enumerations *CDF_StoreStatus* and *CDF_RetrievableStatus* have been replaced by the enumerations *PCDM_StoreStatus* and *PCDM_ReaderStatus*. Correspondingly, the methods *Open, Save* and *SaveAs* of the class *TDocStd_Application* have changed their return value. Any code, which uses these enumerations, needs to be updated.
- *BRepLib_MakeFace* has been modified to accept tolerance value for resolution of degenerated edges. This tolerance parameter has no default value to ensure that the client code takes care of passing a meaningful value, not just *Precision::Confusion*, so some porting overheads are expected.
- If the callback mechanism in call_togl_redraw function was used in the application code, it is necessary to revise it to take into account the new callback execution and provide a check of reason value of Aspect_-GraphicCallbackStruct in callback methods to confirm that the callback code is executed at the right moment. Now the callbacks are executed before redrawing the underlayer, before redrawing the overlayer and at the end of redrawing. The information about the moment when the callback is invoked is provided with the reason value in form of an additional bit flag *(OCC_PRE_REDRAW, OCC_PRE_OVERLAY)*. The state of OpenGl changed in callback methods will not be restored automatically, which might lead to unwanted behavior in redrawing procedure.
- The print method used in the application code might need to be revised to take into account the ability to choose between print algorithms: tile and stretch. The stretch algorithm will be selected by default during porting.
- It is recommended to *BRepMesh_DiscretFactory* users, to check *BRepMesh_DiscretFactory::SetDefault()* return value to determine plugin availability / validity. *BRepMesh_DiscretFactory::Discret()* method now returns handle instead of pointer. The code should be updated in the following manner:

```
Handle(BRepMesh_DiscretRoot) aMeshAlgo = BRepMesh_DiscretFactory::Get().Discret (theShape, theDeflection,
      theAngularToler);
 if (!aMeshAlgo.IsNull())  {}
```

- The default state of *BRepMesh* parallelization has been turned off. The user should switch this flag explicitly:
  - by using methods *BRepMesh_IncrementalMesh::SetParallel(Standard_True)* for each *BRepMesh_-IncrementalMesh* instance before *Perform()*;
  - by calling *BRepMesh_IncrementalMesh::SetParallelDefault(Standard_True)* when *BRepMesh_Discret-Factory* is used to retrieve the meshing tool (this also affects auto-triangulation in *AIS*).

5 Upgrade to OCCT 6.5.3

Porting of user applications from an earlier OCCT version to version 6.5.3 requires taking into account the following major changes:

- As a result of code clean-up and redesign of *TKOpenGl* driver, some obsolete functions and rendering primitives *(TriangleMesh, TriangleSet, Bezier, Polyline, Polygon, PolygonHoles, QuadrangleMesh* and *QuadrangleSet*) have been removed. Instead, the application developers should use primitive arrays that provide the same functionality but are hardware-accelerated. The details can be found in OCCT Visualization User's Guide, "Primitive Arrays" chapter.
- Applications should not call *AIS_InteractiveObject::SetPolygonOffsets()* method for an instance of *AIS_TexturedShape* class after it has been added to *AIS_InteractiveContext*. More generally, modification of *Graphic3d_AspectFillArea3d* parameters for the computed groups of any *AIS_InteractiveObject* subclass that uses texture mapping should be avoided, because this results in broken texture mapping (see issue 23118). It is still possible to apply non-default polygon offsets to *AIS_TexturedShape* by calling *SetPolygonOffsets()* before displaying the shape.
- The applications that might have used internal functions provided by *TKOpenGl* or removed primitives will need to be updated.
- In connection with the implementation of Z-layers it might be necessary to revise the application code or revise the custom direct descendant classes of *Graphic3d_GraphicDriver* and *Graphic3d_StructureManager* to use the Z-layer feature.
- Global variables *Standard_PI* and *PI* have been eliminated (use macro *M_PI* instead).
- Method *HashCode()* has been removed from class *Standard_Transient*. It is advisable to use global function *HashCode()* for Handle objects instead.
- Declaration of operators new/delete for classes has become consistent and is encapsulated in macros.
- Memory management has been changed to use standard heap *(MMGT_OPT=0)* and reentrant mode *(MMGT_REENTRANT=1)* by default.
- Map classes in *NCollection* package now accept one more argument defining a hash tool.

6 Upgrade to OCCT 6.5.4

Porting of user applications from an earlier OCCT version to version 6.5.4 requires taking into account the following major changes:

- The code using obsolete classes *Aspect_PixMap, Xw_PixMap* and *WNT_PixMap* should be rewritten implementing class *Image_PixMap*, which is now retrieved by *ToPixMap* methods as argument. A sample code using *ToPixMap* is given below:

```
#include <Image_AlienPixMap.hxx>
void dump (Handle(V3d_View)& theView3D)
{
  Standard_Integer aWndSizeX = 0;
  Standard_Integer aWndSizeY = 0;
  theView3D->Window()->Size (aWndSizeX, aWndSizeY);
  Image_AlienPixMap aPixMap;
  theView3D->ToPixMap (aPixMap, aWndSizeX, aWndSizeY);
  aPixMap.Save ("c:\\image.png");
}
```

- Now OpenGL resources related to Interactive Objects are automatically freed when the last view (window) is removed from graphical driver. To avoid presentation data loss, the application should replace an old view with a new one in the proper order: first the new view is created and activated and only then the old one is detached and removed.
- It is recommended to use *NCollection* containers with hasher parameter (introduced in 6.5.3) instead of global definition *IsEqual()/HashCode()* as well as to use explicit namespaces to avoid name collision.

7 Upgrade to OCCT 6.6.0

Porting of user applications from an earlier OCCT version to version 6.6.0 requires taking into account the following major changes:

- Due to the changes in the implementation of Boolean Operations, the order of sub-shapes resulting from the same operation performed with OCCT 6.5.x and OCCT 6.6.0 can be different. It is necessary to introduce the corresponding changes in the applications for which the order of sub-shapes resulting from a Boolean operation is important. It is strongly recommended to use identification methods not relying on the order of sub-shapes (e.g. OCAF naming).
- If you need to use OCCT on Mac OS X with X11 (without Cocoa), build OCCT with defined pre-processor macro *CSF_MAC_USE_GLX11*. XLib front-end (previously the only way for unofficial OCCT builds on Mac OS X) is now disabled by default on this platform. If your application has no support for Cocoa framework you may build OCCT with XLib front-end adding *MACOSX_USE_GLX* macro to compiler options (you may check the appropriate option in WOK configuration GUI and in CMake configuration). Notice that XQuartz (XLib implementation for Mac OS X) now is an optional component and does not provide a sufficient level of integrity with native (Cocoa-based) applications in the system. It is not possible to build OCCT with both XLib and Cocoa at the same time due to symbols conflict in OpenGL functions.
- Animation mode and degeneration presentation mode (simplified presentation for animation) and associated methods have been removed from 3D viewer functionality. Correspondingly, the code using methods *SetAnimationModeOn(), SetAnimationModeOff(), AnimationModeIsOn(), AnimationMode(), Tumble(), SetDegenerateModeOn(), SetDegenerateModeOff()* and *DegenerateModeIsOn()* of classes *V3d_View* and *Visual3d_View* will need to be removed or redesigned. Please, notice that Hidden Line Removal presentation was not affected; however, the old code that used methods *V3d_View::SetDegenerateModeOn* or *V3d_View::SetDegenerateModeOff* to control HLR presentation should be updated to use *V3d_View::SetComputedMode* method instead.
- Calls of *Graphic3d_Group::BeginPrimitives()* and *Graphic3d_Group::EndPrimitives()* should be removed from the application code.
- Application functionality for drawing 2D graphics that was formerly based on *TKV2d* API should be migrated to *TKV3d* API. The following changes are recommended for this migration:
  - A 2D view can be implemented as a *V3d_View* instance belonging to *V3d_Viewer* managed by *AIS_InteractiveContext* instance. To turn *V3d_View* into a 2D view, the necessary view orientation should be set up at the view initialization stage using *V3d_View::SetProj()* method, and view rotation methods simply should not be called.
  - Any 2D graphic entity (formerly represented with *AIS2D_InteractiveObject*) should become a class derived from *AIS_InteractiveObject* base. These entities should be manipulated in a view using *AIS_InteractiveContext* class API.
  - All drawing code should be put into *Compute()* virtual method of a custom interactive object class and use API of *Graphic3d* package. In particular, all geometry should be drawn using class hierarchy derived from *Graphic3d_ArrayOfPrimitives*. Normally, the Z coordinate for 2D geometry should be constant, unless the application implements some advanced 2D drawing techniques like e.g. multiple "Z layers" of drawings.
  - Interactive selection of 2D presentations should be set up inside *ComputeSelection()* virtual method of a custom interactive object class, using standard sensitive entities from *Select3D* package and standard or custom entity owners derived from *SelectMgr_EntityOwner* base. Please refer to the Visualization User's Guide for further details concerning OCCT 3D visualization and selection classes. See also *Viewer2D* OCCT sample application, which shows how 2D drawing can be implemented using TKV3d API.
- Run-time graphic driver library loading mechanism based on *CSF_GraphicShr* environment variable usage has been replaced by explicit linking against *TKOpenGl* library. The code sample below shows how the graphic driver should be created and initialized in the application code:

```
// initialize a new viewer with OpenGl graphic driver
Handle(Graphic3d_GraphicDriver) aGraphicDriver =
new OpenGl_GraphicDriver ("TKOpenGl");
  aGraphicDriver->Begin (new Aspect_DisplayConnection());
  TCollection_ExtendedString aNameOfViewer ("Visu3D");
  Handle(V3d_Viewer) aViewer
= new V3d_Viewer (aGraphicDriver, aNameOfViewer.ToExtString());
  aViewer->Init();

// create a new window or a wrapper over the existing window,
// provided by a 3rd-party framework (Qt, MFC, C# or Cocoa)
#if defined(_WIN32) || defined(__WIN32__)
  Aspect_Handle aWindowHandle = (Aspect_Handle )winId();
  Handle(WNT_Window) aWindow = new WNT_Window (winId());
#elif defined(__APPLE__) && !defined(MACOSX_USE_GLX)
  NSView* aViewHandle = (NSView* )winId();
  Handle(Cocoa_Window) aWindow = new Cocoa_Window (aViewHandle);
#else
 Aspect_Handle aWindowHandle = (Aspect_Handle )winId();
  Handle(Xw_Window) aWindow =
     new Xw_Window (aGraphicDriver->GetDisplayConnection(), aWindowHandle);
#endif // WNT

// setup the window for a new view
  Handle(V3d_View) aView = aViewer->CreateView();
  aView->SetWindow (aWindow);
```

- The following changes should be made in the application-specific implementations of texture aspect:
  - *Graphic3d_TextureRoot* inheritors now should return texture image by overloading of *Graphic3d_-TextureRoot::GetImage()* method instead of the old logic.
  - Now you can decide if the application should store the image copy as a field of property or reload it dynamically each time (to optimize the memory usage). The default implementation (which loads the image content from the provided file path) does not hold an extra copy since it will be uploaded to the graphic memory when first used.
  - Notice that the image itself should be created within *Image_PixMap* class from *AlienImage* package, while *Image_Image* class is no more supported and will be removed in the next OCCT release.

8 Upgrade to OCCT 6.7.0

Porting of user applications from an earlier OCCT version to version 6.7.0 requires taking into account the following major changes.

8.1 Object-level clipping and capping algorithm.

- It might be necessary to revise and port code related to management of view-level clipping to use *Graphic3d-_ClipPlane* instead of *V3d_Plane* instances. Please note that *V3d_Plane* class has been preserved – as previously, it can be used as plane representation. Another approach to represent *Graphic3d_ClipPlane* in a view is to use custom presentable object.
- The list of arguments of *Select3D_SensitiveEntity::Matches()* method for picking detection has changed. Since now, for correct selection clipping, the implementations should perform a depth clipping check and return (as output argument) minimum depth value found at the detected part of sensitive. Please refer to CDL / Doxygen documentation to find descriptive hints and snippets.
- *Select3D_SensitiveEntity::ComputeDepth()* abstract method has been removed. Custom implementations should provide depth checks by method *Matches()* instead – all data required for it is available within a scope of single method.
- It might be necessary to revise the code of custom sensitive entities and port *Matches()* and *ComputeDepth()* methods to ensure proper selection clipping. Please note that obsolete signature of *Matches* is not used anymore by the selector. If your class inheriting *Select3D_SensitiveEntity* redefines the method with old signature the code should not compile as the return type has been changed. This is done to prevent override of removed methods.

8.2 Redesign of markers presentation

- Due to the redesign of *Graphic3d_AspectMarker3d* class the code of custom markers initialization should be updated. Notice that you can reuse old markers definition code as *TColStd_HArray1OfByte*; however, *Image_PixMap* is now the preferred way (and supports full-color images on modern hardware).
- Logics and arguments of methods *AIS_InteractiveContext::Erase()* and *AIS_InteractiveContext::EraseAll()* have been changed. Now these methods do not remove resources from *Graphic3d_Structure*; they simply change the visibility flag in it. Therefore, the code that deletes and reomputes resources should be revised.
- *Graphic3d_Group::MarkerSet()* has been removed. *Graphic3d_Group::AddPrimitiveArray()* should be used instead to specify marker(s) array.

8.3 Default views are not created automatically

As the obsolete methods *Init(), DefaultOrthographicView()* and *DefaultPerspectiveView()* have been removed from *V3d_Viewer* class, the two default views are no longer created automatically. It is obligatory to create *V3d_View* instances explicitly, either directly by operator new or by calling *V3d_Viewer::CreateView()*.

The call *V3d_Viewer::SetDefaultLights()* should also be done explicitly at the application level, if the application prefers to use the default light source configuration. Otherwise, the application itself should set up the light sources to obtain a correct 3D scene.

8.4 Improved dimensions implementation

- It might be necessary to revise and port code related to management of *AIS_LengthDimension, AIS_Angle-Dimension* and *AIS_DiameterDimension* presentations. There is no more need to compute value of dimension and pass it as string to constructor argument. The value is computed internally. The custom value can be set with *SetCustomValue()* method.

- The definition of units and general aspect properties is now provided by *Prs3d_DimensionUnits* and *Prs3d_-DimensionApsect* classes.
- It might be also necessary to revise code of your application related to usage of *AIS_DimensionDisplayMode enumeration*. If it used for specifying the selection mode, then it should be replaced by a more appropriate enumeration *AIS_DimensionSelectionMode*.

8.5 NCollection_Set replaced by List collection

It might be necessary to revise your application code, which uses non-ordered *Graphic3d_SetOfHClipPlane* collection type and replace its occurrences by ordered *Graphic3d_SequenceOfHClipPlane* collection type.

9 Upgrade to OCCT 6.8.0

Porting of user applications from an earlier OCCT version to version 6.8.0 requires taking into account the following major changes.

9.1 Changes in NCollection classes

Method *Assign()* in *NCollection* classes does not allow any more copying between different collection types. Such copying should be done manually.

List and map classes in *NCollection* package now require that their items be copy-constructible, but do not require items to have default constructor. Thus the code using *NCollection* classes for non-copy-constructible objects needs be updated. One option is to provide copy constructor; another possibility is to use Handle or other smart pointer.

9.2 3D View Camera

If *ViewMapping* and *ViewOrientation* were used directly, this functionality has to be ported to the new camera model. The following methods should be considered as an alternative to the obsolete *Visual3d* services (all points and directions are supposed to be in world coordinates):

- *Graphic3d_Camera::ViewDimensions()* or *V3d_View::Size()/ZSize()* – returns view width, height and depth (or "Z size"). Since the view is symmetric now, you can easily compute top, bottom, left and right limits. *Graphic3d_Camera::ZNear()/ZFar()* can be used to obtain the near and far clipping distances with respect to the eye.
- *Graphic3d_Camera::Up()* or *V3d_View::Up()* – returns Y direction of the view.
- *Graphic3d_Camera::Direction()* returns the reverse view normal directed from the eye, *V3d_View::Proj()* returns the old-style view normal.
- *Graphic3d_Camera::Eye()* or *V3d_View::Eye()* – returns the camera position (same as projection reference point in old implementation).
- *Graphic3d_Camera::Center()* or *V3d_View::At()* – returns the point the camera looks at (or view reference point according to old terminology).

The current perspective model is not fully backward compatible, so the old perspective-related functionality needs to be reviewed.

Please revise application-specific custom presentations to provide proper bounding box. Otherwise object might become erroneously clipped by automatic *ZFit* or frustum culling algorithms enabled by default.

9.3 Redesign of Connected Interactive Objects

The new implementation of connected Interactive Objects makes it necessary to take the following steps if you use connected Interactive Objects in your application.

- Use new *PrsMgr_PresentableObject* transformation API.
- Call *RemoveChild()* from the original object after connect if you need the original object and *AIS_ConnectedInteractive* to move independently.
- Access instances of objects connected to *AIS_MultiplyConnectedInteractive* with *Children()* method.
- For *PrsMgr_PresentableObject* transformation:
  - *SetLocation (TopLoc_Location) -> SetLocalTransformation (gp_Trsf)*
  - *Location -> LocalTransformation*
  - *HasLocation -> HasTransformation*
  - *ResetLocation -> ResetTransformation*

9.4 Support of UNICODE Characters

Support of UNICODE characters introduced in OCCT breaks backward compatibility with applications, which currently use filenames in extended ASCII encoding bound to the current locale. Such applications should be updated to convert such strings to UTF-8 format.

The conversion from UTF-8 to wchar_t is made using little-endian approach. Thus, this code will not work correctly on big-endian platforms. It is needed to complete this in the way similar as it is done for binary persistence (see the macro *DO_INVERSE* in *FSD_FileHeader.hxx).*

9.5 Elimination of Projection Shift Concept

It might be necessary to revise the application code, which deals with *Center()* method of *V3d_View*.

This method was used to pan a *V3d* view by virtually moving the screen center with respect to the projection ray passed through Eye and At points. There is no more need to derive the panning from the Center parameter to get a camera-like eye position and look at the coordinates. *Eye()* and *At()* now return these coordinates directly. When porting code dealing with *Center()*, the parameters *Eye()* and *At()* can be adjusted instead. Also *V3d_View::SetCenter(Xpix, Ypix)* method can be used instead of *V3d_View::Center(X, Y)* to center the view at the given point. However, if the center coordinates X and Y come from older OCCT releases, calling *V3d_View::Panning(-X, -Y)* can be recommended to compensate missing projection shift effect.

There are several changes introduced to *Graphic3d_Camera*. The internal data structure of the camera is based on *Standard_Real* data types to avoid redundant application-level conversions and precision errors. The transformation matrices now can be evaluated both for *Standard_Real* and *Standard_ShortReal* value types. *ZNear* and *ZFar* planes can be either negative or positive for orthographic camera projection, providing a trade-off between the camera distance and the range of *ZNear* or *ZFar* to reduce difference of exponents of values composing the orientation matrix - to avoid calculation errors. The negative values can be specified to avoid Z-clipping if the reference system of camera goes inside of the model when decreasing camera distance.

The auto z fit mode, since now, has a parameter defining Z-range margin (the one which is usually passed as argument to *ZFitAll()* method). The methods *SetAutoZFitMode(), AutoZFitScaleFactor()* and *ZFitAll()* from class *V3d_View* deal with the new parameter.

The class *Select3D_Projector* now supports both orientation and projection transformation matrices, which can be naturally set for the projector. The definition of projector was revised in *StdSelect_ViewerSelector3d*: perspective and orthographic projection parameters are handled properly. Orthographic projector is based only on direction of projection - no more *Center* property. This makes it possible to avoid unnecessary re-projection of sensitive while panning, zooming or moving along the projection ray of the view. These operations do not affect the orthographic projection.

10 Upgrade to OCCT 6.9.0

Porting of user applications from an earlier OCCT version to version 6.9.0 requires taking into account the following major changes.

10.1 Changes in Selection

Selection mechanism of 3D Viewer has been redesigned to use 3-level BVH tree traverse directly in 3D space instead of projection onto 2D screen space (updated on each rotation). This architectural redesign may require appropriate changes at application level in case if custom Interactive Objects are used.

Standard selection

Usage of standard OCCT selection entities would require only minor updates.

Custom Interactive Objects should implement new virtual method *SelectMgr_SelectableObject::BoundingBox().*

Now the method *SelectMgr_Selection::Sensitive()* does not return *SelectBasics_SensitiveEntity*. It returns an instance of *SelectMgr_SensitiveEntity*, which belongs to a different class hierarchy (thus *DownCast()* will fail). To access base sensitive it is necessary to use method *SelectMgr_SensitiveEntity::BaseSensitive()*. For example:

```
Handle(SelectMgr_Selection) aSelection = anInteractiveObject->Selection (aMode);
for (aSelection->Init(); aSelection->More(); aSelection->Next())
{
   Handle(SelectBasics_SensitiveEntity) anEntity = aSelection->Sensitive()->BaseSensitive();
}
```

Custom sensitive entities

Custom sensitive entities require more complex changes, since the selection algorithm has been redesigned and requires different output from the entities.

The method *SelectBasics_SensitiveEntity::Matches()* of the base class should be overridden following the new signature:

Standard_Boolean Matches (SelectBasics_SelectingVolumeManager& theMgr, SelectBasics_PickResult& thePickResult), where *theMgr* contains information about the currently selected frustum or set of frustums (see *SelectMgr_RectangularFrustum, SelectMgr_TrangularFrustum, SelectMgr_TriangularFrustumSet)* and *SelectBasics_PickResult* is an output parameter, containing information about the depth of the detected entity and distance to its center of geometry.

In the overridden method it is necessary to implement an algorithm of overlap and inclusion detection (the active mode is returned by *theMgr.IsOverlapAllowed()*) with triangular and rectangular frustums.

The depth and distance to the center of geometry must be calculated for the 3D projection of user-picked screen point in the world space. You may use already implemented overlap and inclusion detection methods for different primitives from *SelectMgr_RectangularFrustum* and *SelectMgr_TriangularFrustum*, including triangle, point, axis-aligned box, line segment and planar polygon.

Here is an example of overlap/inclusion test for a box:

```
if (!theMgr.IsOverlapAllowed()) // check for inclusion
{
  Standard_Boolean isInside = Standard_True;
  return theMgr.Overlaps (myBox.CornerMin(), myBox.CornerMax(), &isInside) && isInside;
}

Standard_Real aDepth;
if (!theMgr.Overlaps (myBox, aDepth)) // check for overlap
{
  return Standard_False;
}

thePickResult =
SelectBasics_PickResult (aDepth, theMgr.DistToGeometryCenter (myCenter3d));
```

The interface of *SelectBasics_SensitiveEntity* now contains four new pure virtual functions that should be implemented by each custom sensitive:

- *BoundingBox()* – returns a bounding box of the entity;
- *Clear()* – clears up all the resources and memory allocated for complex sensitive entities;
- *BVH()* – builds a BVH tree for complex sensitive entities, if it is needed;
- *NbSubElements()* – returns atomic sub-entities of a complex sensitive entity, which will be used as primitives for BVH building. If the entity is simple and no BVH is required, this method returns 1.

Each sensitive entity now has its own tolerance, which can be overridden by method *SelectBasics_SensitiveEntity-::SetSensitivityFactor()* called from constructor.

10.2 Changes in Adaptor3d_Curve class

All classes inheriting *Adaptor3d_Curve* (directly or indirectly) must be updated in application code to use new signature of methods *Intervals()* and *NbIntervals()*. Note that no compiler warning will be generated if this is not done.

10.3 Changes in V3d_View class

The methods *V3d_View::Convert* and *V3d_View::ConvertWithProj()* have ceased to return point on the active grid. It might be necessary to revise the code of your application so that *V3d_View::ConvertToGrid()* was called explicitly for the values returned by *V3d_View::Convert* to get analogous coordinates on the grid. The methods *V3d_-View::Convert* and *V3d_View::ConvertWithProj* convert point into reference plane of the view corresponding to the intersection with the projection plane of the eye/view point vector.

11 Upgrade to OCCT 7.0.0

Porting of user applications from an earlier OCCT version to version 7.0.0 requires taking into account the following major changes.

11.1 Removal of legacy persistence

Legacy persistence for shapes and OCAF data based on *Storage_Schema* (toolkits *TKPShape*, *TKPLCAF*, *TKPCAF*, *TKShapeShcema, TLStdLSchema, TKStdSchema*, and *TKXCAFSchema*) has been removed in OCCT 7.0.0. The applications that used these data persistence tools need to be updated to use other persistence mechanisms.

Note

> For compatibility with previous versions, possibility to read standard OCAF data (TKLCAF, TKCAF) from the files stored in old format is preserved (toolkits *TKStdL*, *TKStd*).

The existing data files in standard formats can be converted using OCCT 6.9.1 or a previous version, as follows.

CSFDB files

Files in CSFDB format (usually with extension .csfdb) contain OCCT shape data that can be converted to BRep format. The easiest way to do that is to use ImportExport sample provided with OCCT 6.9.0 (or earlier):

- Start ImportExport sample;
- Select File / New;
- Select File / Import / CSFDB... and specify the file to be converted;
- Drag the mouse with the right button pressed across the view to select all shapes by the rectangle;
- Select File / Export / BREP... and specify the location and name for the resulting file

OCAF and XCAF documents

Files containing OCAF data saved in the old format usually have extensions *.std, .sgd* or *.dxc* (XDE documents). These files can be converted to XML or binary OCAF formats using DRAW Test Harness commands. Note that if the file contains only attributes defined in TKLCAF and TKCAF, this action can be performed in OCCT 7.0; otherwise OCCT 6.9.1 or earlier should be used.

For that, start *DRAWEXE* and perform the following commands:

- To convert ∗*.std* and ∗*.sgd* file formats to binary format ∗*.cbf* (The created document should be in *BinOcaf* format instead of *MDTV-Standard*):

```
Draw[]> pload ALL
Draw[]> Open [path to *.std or *.sgd file] Doc
Draw[]> Format Doc BinOcaf
Draw[]> SaveAs Doc [path to the new file]
```

- To convert ∗*.dxc* file format to binary format ∗*.xbf* (The created document should be in *BinXCAF* format instead of *MDTV-XCAF*):

```
Draw[]> pload ALL
Draw[]> XOpen [path to *.dxc file] Doc
Draw[]> Format Doc BinXCAF
Draw[]> XSave Doc [path to the new file]
```

On Windows, it is necessary to replace back slashes in the file path by direct slashes or pairs of back slashes.

Use *XmlOcaf* or *XmlXCAF* instead of *BinOcaf* and *BinXCAF*, respectively, to save in XML format instead of binary one.

11.2 Removal of CDL and WOK

OCCT code has been completely refactored in version 7.0 to get rid of obsolete technologies used since its inception: CDL (Cas.Cade Definition Language) and WOK (Workshop Organization Kit).

C++ code previously generated by WOK from CDL declarations is now included directly in OCCT sources.

This modification did not change names, API, and behavior of existing OCCT classes, thus in general the code based on OCCT 6.x should compile and work fine with OCCT 7.0. However, due to redesign of basic mechanisms (CDL generic classes, Handles and RTTI) using C++ templates, some changes may be necessary in the code when porting to OCCT 7.0, as described below.

WOK is not necessary anymore for building OCCT from sources, though it still can be used in a traditional way – auxiliary files required for that are preserved. The recommended method for building OCCT 7.x is CMake, see occt_dev_guides__building_cmake. The alternative solution is to use project files generated by OCCT legacy tool **genproj**, see occt_dev_guides__building_msvc, occt_dev_guides__building_code_blocks, and occt_dev_guides_-_building_xcode.

11.2.1 Automatic upgrade

Most of typical changes required for upgrading code for OCCT 7.0 can be done automatically using the *upgrade* tool included in OCCT 7.0. This tool is a Tcl script, thus Tcl should be available on your workstation to run it.

Example:

```
$ tclsh
% source <path_to_occt>/adm/upgrade.tcl
% upgrade -recurse -all -src=<path_to_your_sources>
```

On Windows, the helper batch script *upgrade.bat* can be used, provided that Tcl is either available in *PATH*, or configured via *custom.bat* script (for instance, if you use OCCT installed from Windows installer package). Start it from the command prompt:

```
cmd> <path_to_occt>\upgrade.bat -recurse -all -inc=<path_to_occt>\inc -src=<path_to_your_sources> [options]
```

Run the upgrade tool without arguments to see the list of available options.

The upgrade tool performs the following changes in the code.

1. Replaces macro *DEFINE_STANDARD_RTTI* by *DEFINE_STANDARD_RTTIEXT*, with second argument indicating base class for the main argument class (if inheritance is recognized by the script):

   ```
   DEFINE_STANDARD_RTTI(Class) -> DEFINE_STANDARD_RTTIEXT(Class, Base)
   ```

   Note

   If macro *DEFINE_STANDARD_RTTI* with two arguments (used in intermediate development versions of OCCT 7.0) is found, the script will convert it to either *DEFINE_STANDARD_RTTIEXT* or *DEFINE-_STANDARD_RTTI_INLINE*. The former case is used if current file is header and source file with the same name is found in the same folder. In this case, macro *IMPLEMENT_STANDARD_RTTI* is injected in the corresponding source file. The latter variant defines all methods for RTTI as inline, and does not require *IMPLEMENT_STANDARD_RTTIEXT* macro.

2. Replaces forward declarations of collection classes previously generated from CDL generics (defined in *T-Collection* package) by inclusion of the corresponding header:

   ```
   class TColStd_Array1OfReal; -> #include <TColStd_Array1OfReal.hxx>
   ```

3. Replaces underscored names of *Handle* classes by usage of a macro:

   ```
   Handle_Class -> Handle(Class)
   ```

This change is not applied if the source or header file is recognized as containing the definition of Qt class with signals or slots, to avoid possible compilation errors of MOC files caused by inability of MOC to recognize macros (see `http://doc.qt.io/qt-4.8/signalsandslots.html`). The file is considered as defining a Qt object if it contains strings *Q_OBJECT* and either *slots:* or *signals:*.

4. Removes forward declarations of classes with names *Handle(C)* or *Handle_C*, replacing them either by forward declaration of its argument class, or (for files defining Qt objects) *#include* statement for a header with the name of the argument class and extension .hxx:

   ```
   class Handle(TColStd_HArray1OfReal); -> #include <TColStd_HArray1OfReal.hxx>
   ```

5. Removes *#includes* of files *Handle_...hxx* that have disappeared in OCCT 7.0:

   ```
   #include <Handle_Geom_Curve.hxx> ->
   ```

6. Removes *typedef* statements that use *Handle* macro to generate the name:

   ```
   typedef NCollection_Handle<Message_Msg> Handle(Message_Msg); ->
   ```

7. Converts C-style casts applied to Handles into calls to *DownCast()* method:

   ```
   ((Handle(A)&)b)      -> Handle(A)::DownCast(b)
   (Handle(A)&)b        -> Handle(A)::DownCast(b)
   (*((Handle(A)*)&b))  -> Handle(A)::DownCast(b)
   *((Handle(A)*)&b)    -> Handle(A)::DownCast(b)
   (*(Handle(A)*)&b)    -> Handle(A)::DownCast(b)
   ```

8. Moves *Handle()* macro out of namespace scope:

   ```
   Namespace::Handle(Class) -> Handle(Namespace::Class)
   ```

9. Converts local variables of reference type, which are initialized by a temporary object returned by call to *DownCast()*, to the variables of non-reference type (to avoid using references to destroyed memory):

   ```
   const Handle(A)& a = Handle(B)::DownCast (b); -> Handle(A) a (Handle(B)::DownCast (b));
   ```

10. Adds *#include* for all classes used as argument to macro *STANDARD_TYPE()*, except for already included ones;

11. Removes uses of obsolete macros *IMPLEMENT_DOWNCAST* and *IMPLEMENT_STANDARD_...*, except *IMPLEMENT_STANDARD_RTTIEXT*.

    Note

    If you plan to keep compatibility of your code with older versions of OCCT, add option *-compat* to avoid this change. See also **Preserving compatibility with OCCT 6.x** (p. 23).

As long as the upgrade routine runs, some information messages are sent to the standard output. In some cases the warnings or errors like the following may appear:

```
Error in {HEADER_FILE}: Macro DEFINE_STANDARD_RTTI used for class {CLASS_NAME} whose declaration is not
      found in this file, cannot fix
```

Be sure to check carefully all reported errors and warnings, as the corresponding code will likely require manual corrections. In some cases these messages may help you to detect errors in your code, for instance, cases where *DEFINE_STANDARD_RTTI* macro is used with incorrect class name as an argument.

11.2.2 Possible compiler errors

Some situations requiring upgrade cannot be detected and / or handled by the automatic procedure. If you get compiler errors or warnings when trying to build the upgraded code, you will need to fix them manually. The following paragraphs list known situations of this kind.

Missing header files

The use of handle objects (construction, comparison using operators == or !=, use of function *STANDRAD_TYPE()* and method *DownCast()*) now requires the type of the object pointed by Handle to be completely known at compile time. Thus it may be necessary to include header of the corresponding class to make the code compilable.

For example, the following lines will fail to compile if *Geom_Line.hxx* is not included:

```
Handle(Geom_Line) aLine = 0;
if (aLine != aCurve) {...}
if (aCurve->IsKind(STANDARD_TYPE(Geom_Line)) {...}
aLine = Handle(Geom_Line)::DownCast (aCurve);
```

Note that it is not necessary to include header of the class to declare Handle to it. However, if you define a class *B* that uses Handle(*A*) in its fields, or contains a method returning Handle(*A*), it is advisable to have header defining *A* included in the header of *B*. This will eliminate the need to include the header *A* in each source file where class *B* is used.

Ambiguity of calls to overloaded functions

This issue appears in the compilers that do not support default arguments in template functions (known cases are Visual C++ 10 and 11): the compiler reports an ambiguity error if a handle is used in the argument of a call to the function that has two or move overloaded versions, accepting handles to different types. The problem is that operator *const handle<T2>&* is defined for any type *T2*, thus the compiler cannot make the right choice.

Example:

```
void func (const Handle(Geom_Curve)&);
void func (const Handle(Geom_Surface)&);

Handle(Geom_TrimmedCurve) aCurve = new Geom_TrimmedCurve (...);
func (aCurve); // ambiguity error in VC++ 10
```

Note that this problem can be avoided in many cases if macro *OCCT_HANDLE_NOCAST* is used, see **below** (p. 22).

To resolve this ambiguity, change your code so that argument type should correspond exactly to the function signature. In some cases this can be done by using the relevant type for the corresponding variable, like in the example above:

```
Handle(Geom_Curve) aCurve = new Geom_TrimmedCurve (...);
```

Other variants consist in assigning the argument to a local variable of the correct type and using the direct cast or constructor:

```
const Handle(Geom_Curve)& aGCurve (aTrimmedCurve);
func (aGCurve); // OK - argument has exact type
func (static_cast(aCurve)); // OK - direct cast
func (Handle(Geom_Curve)(aCurve)); // OK - temporary handle is constructed
```

Another possibility consists in defining additional template variant of the overloaded function causing ambiguity, and using *SFINAE* to resolve the ambiguity. This technique can be illustrated by the definition of the template variant of method *IGESData_IGESWriter::Send()*.

Lack of implicit cast to base type

As the cast of a handle to the reference to another handle to the base type has become a user-defined operation, the conversions that require this cast together with another user-defined cast will not be resolved automatically by the compiler.

For example:

```
Handle(Geom_Geometry) aC = GC_MakeLine (p, v); // compiler error
```

The problem is that the class *GCE2d_MakeSegment* has a user-defined conversion to *const Handle(Geom_-TrimmedCurve)&,* which is not the same as the type of the local variable *aC*.

To resolve this, use method *Value()*:

```
Handle(Geom_Geometry) aC = GC_MakeLine (p, v).Value(); // ok
```

or use variable of the appropriate type:

```
Handle(Geom_TrimmedCurve) aC = GC_MakeLine (p, v); // ok
```

With GCC compiler, similar problem appears when const handle to derived type is used to construct handle to base type via assignment (and in some cases in return statement), for instance:

```
const Handle(Geom_Line) aLine;
Handle(Geom_Curve) c1 = aLine; // GCC error
Handle(Geom_Curve) c2 (aLine); // ok
```

This problem is specific to GCC and it does not appear if macro *OCCT_HANDLE_NOCAST* is used, see **below** (p. 22).

Incorrect use of STANDARD_TYPE and Handle macros

You might need to clean your code from incorrect use of macros *STANDARD_TYPE*() and *Handle*().

1. Explicit definitions of static functions with names generated by macro *STANDARD_TYPE()*, which are artifacts of old implementation of RTTI, should be removed.

   Example:

   ```
   const Handle(Standard_Type)& STANDARD_TYPE(math_GlobOptMin)
   {
     static Handle(Standard_Type) _atype = new Standard_Type ("math_GlobOptMin", sizeof (math_GlobOptMin));
     return _atype;
   }
   ```

2. Incorrect location of closing parenthesis of *Handle()* macro that was not detectable in OCCT 6.x will cause a compiler error and must be corrected.

   Example (note misplaced closing parenthesis):

   ```
   aBSpline = Handle( Geom2d_BSplineCurve::DownCast(BS->Copy()) );
   ```

Use of class Standard_AncestorIterator

Class *Standard_AncestorIterator* has been removed; use method *Parent()* of *Standard_Type* class to parse the inheritance chain.

Absence of cast to Standard_Transient∗

Handles in OCCT 7.0 do not have the operator of conversion to *Standard_Transient∗,* which was present in earlier versions. This is done to prevent possible unintended errors like this:

```
Handle(Geom_Line) aLine = ...;
Handle(Geom_Surface) aSurf = ...;
...
if (aLine == aSurf) {...} // will cause a compiler error in OCCT 7.0, but not OCCT 6.x
```

The places where this implicit cast has been used should be corrected manually. The typical situation is when Handle is passed to stream:

```
Handle(Geom_Line) aLine = ...;
os << aLine; // in OCCT 6.9.0, resolves to operator << (void*)
```

Call method *get()* explicitly to output the address of the Handle.

Method DownCast for non-base types

Method *DownCast()* in OCCT 7.0 is made templated; if it is used with argument which is not a base class, "deprecated" compiler warning is generated. This is done to prevent possible unintended errors like this:

```
Handle(Geom_Surface) aSurf = ;
Handle(Geom_Line) aLine =
  Handle(Geom_Line)::DownCast (aSurf); // will cause a compiler warning in OCCT 7.0, but not OCCT 6.x
```

The places where this cast has been used should be corrected manually.

If down casting is used in a template context where argument can have the same or unrelated type so that *DownCast()* may be not available in all cases, use C++ *dynamic_cast<>* instead, e.g.:

```
template <class T>
bool CheckLine (const Handle(T) theArg)
{
  Handle(Geom_Line) aLine = dynamic_cast<Geom_Line> (theArg.get());
  ...
}
```

11.2.3 Possible runtime problems

Here is the list of known possible problems at run time after the upgrade to OCCT 7.0.

References to temporary objects

In previous versions, the compiler was able to detect the situation when a local variable of a "reference to a Handle" type is initialized by temporary object, and ensured that lifetime of that object is longer than that of the variable. In OCCT 7.0 with default options, it will not work if types of the temporary object and variable are different (due to involvement of user-defined type cast), thus such temporary object will be destroyed immediately.

This problem does not appear if macro *OCCT_HANDLE_NOCAST* is used during compilation, see below.

Example:

```
// note that DownCast() returns new temporary object!
const Handle(Geom_BoundedCurve)& aBC =
Handle(Geom_TrimmedCurve)::DownCast(aCurve);
aBC->Transform (T); // access violation in OCCT 7.0
```

11.2.4 Option to avoid cast of handle to reference to base type

In OCCT 6.x and earlier versions the handle classes formed a hierarchy echoing hierarchy of corresponding object classes. This automatically enabled possibility to use handle to derived class in all contexts where handle to base class was needed, e.g. pass it in function by reference without copying:

```
Standard_Boolean GetCurve (Handle(Geom_Curve)& theCurve);
....
Handle(Geom_Line) aLine;
if (GetCurve (aLine)) {
  // use aLine, unsafe
}
```

This feature was used in multiple places in OCCT and dependent projects. However it is potentially unsafe: in the above example no checks are done at compile time or at run time to ensure that argument handle is assigned a type compatible with the type of handle passed as argument. If object of incompatible type (e.g. Geom_Circle) is assigned to *theCurve*, the behavior will be unpredictable.

For compatibility with existing code, by default OCCT 7.0 keeps this possibility, providing operators of type cast to handle to base type. Besides being unsafe, in specific situations this feature may cause compile-time or run-time errors as described above.

In order to provide safer behavior, this feature can be disabled by defining a compile-time macro *OCCT_HANDLE_NOCAST*. When it is defined, constructors and assignment operators are defined (instead of type cast operators) to convert from handle to defived type to handle to base type. This implies creation of temporary objects and hence may be more expensive at run time in some circumstances, however this way is more standard, safer, and in general recommended.

The code that relies on possibility of casting to base should be amended so that handle of argument type is always used in function call, and to use DownCast() to safely convert the result to desired type. For instance, the code from the example below can be changed as follows:

```
Handle(Geom_Line) aLine;
Handle(Geom_Curve) aCurve;
if (GetCurve (aCure) && !(aLine = Handle(Geom_Line)::DownCast (aCurve)).IsNull()) {
  // use aLine safely
}
```

11.2.5 Preserving compatibility with OCCT 6.x

If you like to preserve the compatibility of your application code with OCCT versions 6.x even after the upgrade to 7.0, consider the following suggestions:

1. If your code used sequences of macros *IMPLEMENT_STANDARD_...* generated by WOK, replace them by single macro *IMPLEMENT_STANDARD_RTTIEXT*
2. When running automatic upgrade tool, add option *-compat*.
3. Define macros *DEFINE_STANDARD_RTTIEXT* and *DEFINE_STANDARD_RTTI_INLINE* when building with previous versions of OCCT, resolving to *DEFINE_STANDARD_RTTI* with single argument

   Example:

   ```
   #if OCC_VERSION_HEX < 0x070000
     #define DEFINE_STANDARD_RTTIEXT(C1,C2) DEFINE_STANDARD_RTTI(C1)
     #define DEFINE_STANDARD_RTTI_INLINE(C1,C2) DEFINE_STANDARD_RTTI(C1)
   #endif
   ```

11.2.6 Applications based on CDL and WOK

If your application is essentially based on CDL, and you need to upgrade it to OCCT 7.0, you will very likely need to convert your application code to non-CDL form. This is a non-trivial effort; the required actions would depend strongly on the structure of the code and used CDL features.

The upgrade script and sources of a specialized WOK version used for OCCT code upgrade can be found in WOK Git repository in branch `CR0_700_2`.

`Contact us` if you need more help.

11.3 Separation of BSpline cache

Implementation of NURBS curves and surfaces has been revised: the cache of polynomial coefficients, which is used to accelerate calculate values of B-spline, has been separated from data objects *Geom2d_BSplineCurve, Geom_BSplineCurve* and *Geom_BSplineSurface* into the dedicated classes *BSplCLib_Cache* and *BSplSLib_-Cache*.

The benefits of this change are:

- Reduced memory footprint of OCCT shapes (up to 20% on some cases)
- Possibility to evaluate the same B-Spline concurrently in parallel threads without data races and mutex locks

The drawback is that direct evaluation of B-Splines using methods of curves and surfaces becomes slower due to the absence of cache. The slow-down can be avoided by using adaptor classes *Geom2dAdaptor_Curve, Geom-Adaptor_Curve* and *GeomAdaptor_Surface*, which now use cache when the curve or surface is a B-spline.

OCCT algorithms have been changed to use adaptors for B-spline calculations instead of direct methods for curves and surfaces. The same changes (use of adaptors instead of direct call to curve and surface methods) should be implemented in relevant places in the applications based on OCCT to get the maximum performance.

11.4 Structural result of Boolean operations

The result of Boolean operations became structured according to the structure of the input shapes. Therefore it may impact old applications that always iterate on direct children of the result compound assuming to obtain solids as iteration items, regardless of the structure of the input shapes. In order to get always solids as iteration items it is recommended to use TopExp_Explorer instead of TopoDS_Iterator.

11.5 BRepExtrema_ExtCC finds one solution only

Extrema computation between non-analytical curves in shape-shape distance calculation algorithm has been changed in order to return only one solution. So, if e.g. two edges are created on parallel b-spline curves the algorithm BRepExtrema_DistShapeShape will return only one solution instead of enormous number of solutions. There is no way to get algorithm working in old manner.

11.6 Removal of SortTools package

Package *SortTools* has been removed. The code that used the tools provided by that package should be corrected manually. The recommended approach is to use sorting algorithms provided by STL.

For instance:

```
#include <SortTools_StraightInsertionSortOfReal.hxx>
#include <SortTools_ShellSortOfReal.hxx>
#include <TCollection_CompareOfReal.hxx>
...
TCollection_Array1OfReal aValues = ...;
...
TCollection_CompareOfReal aCompReal;
SortTools_StraightInsertionSortOfReal::Sort(aValues, aCompReal);
```

can be replaced by:

```
#include <algorithm>
...
TCollection_Array1OfReal aValues = ...;
...
std::stable_sort (aValues.begin(), aValues.end());
```

11.7 On-screen objects and ColorScale

The old mechanism for rendering Underlay and Overlay on-screen 2D objects based on *Visual3d_Layer* and immediate drawing model (uncached and thus slow) has been removed. Classes *Aspect_Clayer2d, OpenGl_GraphicDriver_Layer, Visual3d_Layer, Visual3d_LayerItem, V3d_LayerMgr* and *V3d_LayerMgrPointer* have been deleted.

General AIS interactive objects with transformation persistence flag *Graphic3d_TMF_2d* can be used as a replacement of *Visual3d_LayerItem*. The anchor point specified for transformation persistence defines the window corner of (or center in case of (0, 0) point). To keep on-screen 2D objects on top of the main screen, they can be assigned to the appropriate Z-layer. Predefined Z-layers *Graphic3d_ZLayerId_TopOSD* and *Graphic3d_ZLayerId_BotOSD* are intended to replace Underlay and Overlay layers within the old API.

ColorScale object previously implemented using *Visual3d_LayerItem* has been moved to a new class *AIS_ColorScale*, with width and height specified explicitly. The property of *V3d_View* storing the global *ColorScale* object has been removed with associated methods *V3d_View::ColorScaleDisplay(), V3d_View::ColorScaleErase(), V3d_View::ColorScaleIsDisplayed()* and *V3d_View::ColorScale()* as well as the classes *V3d_ColorScale, V3d_ColorScaleLayerItem* and *Aspect_ColorScale*. Here is an example of creating *ColorScale* using the updated API:

```
Handle(AIS_ColorScale) aCS = new AIS_ColorScale();
// configuring
Standard_Integer aWidth, aHeight;
aView->Window()->Size (aWidth, aHeight);
aCS->SetSize              (aWidth, aHeight);
aCS->SetRange             (0.0, 10.0);
aCS->SetNumberOfIntervals (10);
```

```
// displaying
aCS->SetZLayer (Graphic3d_ZLayerId_TopOSD);
aCS->SetTransformPersistence (Graphic3d_TMF_2d, gp_Pnt (-1,-1,0));
aCS->SetToUpdate();
theContextAIS->Display (aCS);
```

To see how 2d objects are implemented in OCCT you can call Draw commands *vcolorscale, vlayerline* or *vdrawtext* (with *-2d* option). Draw command *vcolorscale* now requires the name of *ColorScale* object as argument. To display this object use command *vdisplay*. For example:

```
pload VISUALIZATION
vinit
vcolorscale cs -demo
pload MODELING
box b 100 100 100
vdisplay b
vsetdispmode 1
vfit
vlayerline 0 300 300 300 10
vdrawtext t "2D-TEXT" -2d -pos 0 150 0 -color red
```

Here is a small example in C++ illustrating how to display a custom AIS object in 2d:

```
Handle(AIS_InteractiveContext) aContext = ...;
Handle(AIS_InteractiveObject) anObj =...; // create an AIS object
anObj->SetZLayer(Graphic3d_ZLayerId_TopOSD); // display object in overlay
anObj->SetTransformPersistence (Graphic3d_TMF_2d, gp_Pnt (-1,-1,0)); // set 2d flag, coordinate origin is
       set to down-left corner
aContext->Display (anObj); // display the object
```

11.8 UserDraw and Visual3d

Visual3d package

Package *Visual3d* implementing the intermediate layer between high-level *V3d* classes and low-level OpenGl classes for views and graphic structures management has been dropped.

The *OpenGl_View* inherits from the new class *Graphic3d_CView*. *Graphic3d_CView* is an interface class that declares abstract methods for managing displayed structures, display properties and a base layer code that implements computation and management of HLR (or more broadly speaking view-depended) structures.

In the new implementation it takes place of the eliminated *Visual3d_View*. As before the instance of *Graphic3d_CView* is still completely managed by *V3d_View* classes. It can be accessed through *V3d_View* interface but normally it should not be required as all its methods are completely wrapped.

In more details, a concrete specialization of *Graphic3d_CView* is created and returned by the graphical driver on request. Right after the creation the views are directly used for setting rendering properties and adding graphical structures to be displayed.

The rendering of graphics is possible after mapping a window and activating the view. The direct setting of properties obsoletes the use of intermediate structures with display parameter like *Visual3d_ContextView*, etc. This means that the whole package *Visual3d* becomes redundant.

The functionality previously provided by *Visual3d* package has been redesigned in the following way :

- The management of display of structures has been moved from *Visual3d_ViewManager* into *Graphic3d_StructureManager*.
- The class *Visual3d_View* has been removed. The management of computed structures has been moved into the base layer of *Graphi3d_CView*.
- All intermediate structures for storing view parameters, e.g. *Visual3d_ContextView*, have been removed. The settings are now kept by instances of *Graphic3d_CView*.
- The intermediate class *Visual3d_Light* has been removed. All light properties are stored in *Graphic3d_CLight* structure, which is directly accessed by instances of *V3d_Light* classes.
- All necessary enumerations have been moved into *Graphic3d* package.

Custom OpenGL rendering and UserDraw

Old APIs based on global callback functions for creating *UserDraw* objects and for performing custom OpenGL rendering within the view have been dropped. *UserDraw* callbacks are no more required since *OpenGl_Group* now inherits *Graphic3d_Group* and thus can be accessed directly from *AIS_InteractiveObject*:

```
//! Class implementing custom OpenGL element.
class UserDrawElement : public OpenGl_Element {};

//! Implementation of virtual method AIS_InteractiveObject::Compute().
void UserDrawObject::Compute (const Handle(PrsMgr_PresentationManager3d)& thePrsMgr,
                              const Handle(Prs3d_Presentation)& thePrs,
                              const Standard_Integer theMode)
{
  Graphic3d_Vec4 aBndMin (myCoords[0], myCoords[1], myCoords[2], 1.0f);
  Graphic3d_Vec4 aBndMax (myCoords[3], myCoords[4], myCoords[5], 1.0f);

  // casting to OpenGl_Group should be always true as far as application uses OpenGl_GraphicDriver for
       rendering
  Handle(OpenGl_Group) aGroup = Handle(OpenGl_Group)::DownCast (thePrs->NewGroup());
  aGroup->SetMinMaxValues (aBndMin.x(), aBndMin.y(), aBndMin.z(),
                           aBndMax.x(), aBndMax.y(), aBndMax.z());
  UserDrawElement* anElem = new UserDrawElement (this);
  aGroup->AddElement(anElem);

  // invalidate bounding box of the scene
  thePrsMgr->StructureManager()->Update (thePrsMgr->StructureManager()->UpdateMode());
}
```

To perform a custom OpenGL code within the view, it is necessary to inherit from class *OpenGl_View*. See the following code sample:

```
//! Custom view.
class UserView : public OpenGl_View
{
public:
  //! Override rendering into the view.
  virtual void render (Graphic3d_Camera::Projection theProjection,
                       OpenGl_FrameBuffer*          theReadDrawFbo,
                       const Standard_Boolean       theToDrawImmediate)
  {
    OpenGl_View::render (theProjection, theReadDrawFbo, theToDrawImmediate);
    if (theToDrawImmediate)
    {
      return;
    }

    // perform custom drawing
    const Handle(OpenGl_Context)& aCtx = myWorkspace->GetGlContext();
    GLfloat aVerts[3] = { 0.0f, 0,0f, 0,0f };
    aCtx->core20->glEnableClientState(GL_VERTEX_ARRAY);
    aCtx->core20->glVertexPointer(3, GL_FLOAT, 0, aVerts);
    aCtx->core20->glDrawArrays(GL_POINTS, 0, 1);
    aCtx->core20->glDisableClientState(GL_VERTEX_ARRAY);
  }

};

//! Custom driver for creating UserView.
class UserDriver : public OpenGl_GraphicDriver
{
public:
  //! Create instance of own view.
  virtual Handle(Graphic3d_CView) CreateView (const Handle(Graphic3d_StructureManager)& theMgr)
      Standard_OVERRIDE
  {
    Handle(UserView) aView = new UserView (theMgr, this, myCaps, myDeviceLostFlag, &myStateCounter);
    myMapOfView.Add (aView);
    for (TColStd_SequenceOfInteger::Iterator aLayerIt (myLayerSeq); aLayerIt.More(); aLayerIt.Next())
    {
      const Graphic3d_ZLayerId        aLayerID  = aLayerIt.Value();
      const Graphic3d_ZLayerSettings& aSettings = myMapOfZLayerSettings.Find (aLayerID);
      aView->AddZLayer          (aLayerID);
      aView->SetZLayerSettings (aLayerID, aSettings);
    }
    return aView;
  }
};
```

11.9 Deprecation of Local Context

The conception of Local Context has been deprecated. The related classes, e.g. *AIS_LocalContext*, and methods (*AIS_InteractiveContext::OpenLocalContext()* and others) will be removed in a future OCCT release.

The main functionality provided by Local Context - selection of object subparts - can be now used within Neutral Point without opening any Local Context.

11.10 Separation of visualization part from TKCAF

Visualization CAF attributes have been moved into a new toolkit *TKVCAF*. If your application uses the classes from *TPrsStd* package then add link to *TKVCAF* library.

Version numbers of *BinOCAF* and *XmlOCAF* formats are incremented; new files cannot be read by earlier versions of OCCT.

Before loading the OCAF files saved by previous versions and containing *TPrsStd_AISPresentation* attribute it is necessary to define the environment variable *CSF_MIGRATION_TYPES*, pointing to file *src/StdResources/-MigrationSheet.txt*. When using documents loaded from a file, make sure to call method *TPrsStd_AISViewer::New()* prior to accessing *TPrsStd_AISPresentation* attributes in this document as that method creates them.

11.11 Correction of interpretation of Euler angles in gp_Quaternion

Conversion of *gp_Quaternion* to and from intrinsic Tait-Bryan angles (including *gp_YawPitchRoll*) is fixed.

Before that fix the sequence of rotation axes was opposite to the intended; e.g. *gp_YawPitchRoll* (equivalent to *gp-_Intrinsic_ZYX*) actually defined intrinsic rotations around X, then Y, then Z. Now the rotations are made in correct order.

Applications that use *gp_Quaternion* to convert Yaw-Pitch-Roll angles (or other intrinsic Tait-Bryan sequences) may need to be updated to take this change into account.

11.12 Zoom Persistent Selection

Zoom persistent selection introduces a new structure *Graphic3d_TransformPers* for transform persistence methods and parameters and a new class *Graphic3d_WorldViewProjState* for referring camera transformation state. You might need to update your code to deal with the new classes if you were using the related features. Please, keep in mind the following:

- *Graphic3d_Camera::ModelViewState* has been renamed to *Graphic3d_Camera::WorldViewState*.
- Transformation matrix utilities from *OpenGl_Utils* namespace have been moved to *Graphic3d_TransformUtils* and *Graphic3d_TransformUtils.hxx* header respectively.
- Matrix stack utilities from *OpenGl_Utils* namespace have been moved to *OpenGl_MatrixStack* class and *OpenGl_MatrixStack.hxx* header respectively.
- *OpenGl_View* methods *Begin/EndTransformPersistence* have been removed. Please, use *Graphic3d_-TransformPers::Apply()* instead to apply persistence to perspective and world-view projection matrices.