Object Libraries

Visualization User's Guide

Version 6.3 / September 2008

NOTICE FOR USERS:

This User Guide is a general instruction for Open CASCADE study. It may be incomplete and even contain occasional mistakes, particularly in examples, samples, etc. Open CASCADE S.A.S. bears no responsibility for such mistakes. If you find any mistakes or imperfections in this document, or if you have suggestions for improving this document, please, contact us and contribute your share to the development of Open CASCADE Technology: bugmaster@opencascade.com

Tour Opus 12
77, Esplanade du Général de Gaulle
92914 PARIS LA DEFENSE
FRANCE

TABLE OF CONTENTS

1. Introduction

This manual explains how to use Open CASCADE Visualization. It provides basic documentation on setting up and using Visualization. For advanced information on Visualization and its applications, see our offerings on our web site at www.opencascade.org/support/training.html

Visualization in Open CASCADE is based on the separation of:

- on the one hand - the data which stores the geometry and topology of the entities you want to display and select, and
- on the other - its display and selection.

1.1. Open CASCADE Visualization and the Organization of this guide

Display is managed through the Presentation component, and selection through the Selection component.

To make management of these functionalities in 3D more intuitive and consequently, more transparent, **Application Interactive Services** have been created. **AIS** use the notion of the *interactive object*, a displayable and selectable entity, which represents an element from the application data. As a result, in 3D, you, the user, have no need to be familiar with any functions underlying AIS unless you want to create your own interactive objects or filters.

If, however, you require types of interactive objects and filters other than those provided, you will need to know the mechanics of presentable and selectable objects, specifically how to implement their virtual functions. To do this requires familiarity with such fundamental concepts as the sensitive primitive and the presentable object.

The packages used to display 3D objects are the following:

- AIS
- StdPrs
- V3d
- Prs3d
- Graphic3d

If you are concerned with 2D visualization, you must familiarize yourself with the fundamental concepts of presentation as outlined in the section on this subject in chapter 1, Fundamental Concepts. In brief, the packages used to display 2D objects are the following:

- PrsMgr
- PresentationManager2d
- V2d
- Graphic2d.

Figure 1 below presents a schematic overview of the relations between the key concepts and packages in visualization.

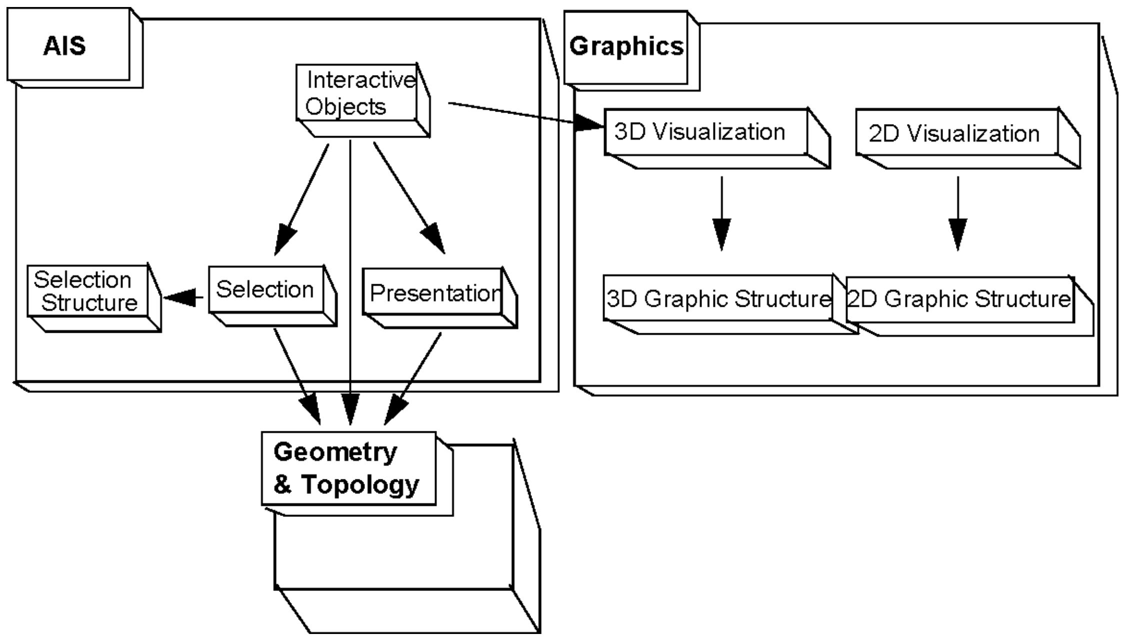
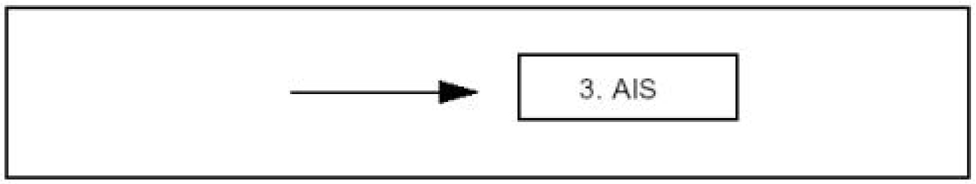


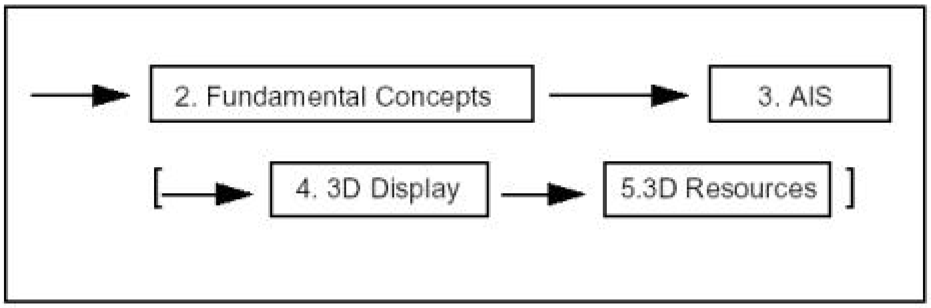
Figure 1. Key concepts and packages in visualization

To answer different needs of CASCADE users, this user's guide offers the following three paths in reading it.

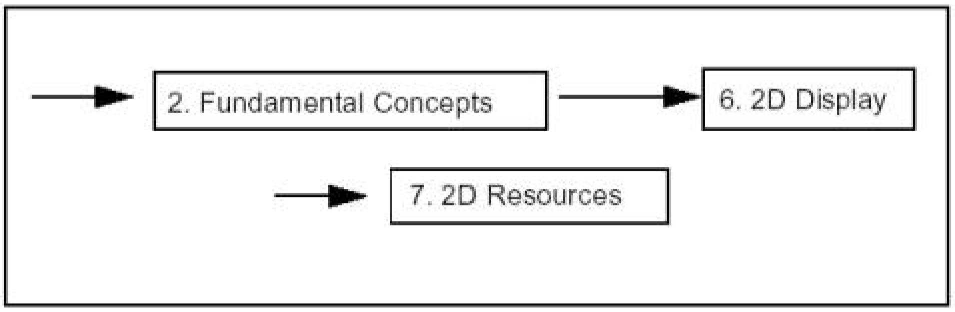
- If the 3D services proposed in AIS meet your requirements, you need only read chapter 3, *AIS: Application Interactive Services*.



- If the services provided do not satisfy your requirements - if for example, you need a filter on another type of entity... - you should read chapter 2 *Fundamental Concepts*, chapter 3 *AIS: Application Interactive Services*, and possibly chapters 4 and 5 *3D Display* and *3D Resources*. You may want to begin with the chapter presenting AIS.



- If your display will be in 2D, you should read chapter 1 *Fundamental Concepts*, chapter 6 *2D Display* and chapter 7 *2D Resources*.



2. Fundamental Concepts

2. 1 Presentation

In Open CASCADE, display services are separated from the data, which they represent which is generated by applicative algorithms. This division allows you to modify a geometric or topological algorithm and its resulting objects without modifying the services by which it is displayed.

2. 1. 1 Structure of the Presentation

Displaying an object on the screen involves three kinds of entity:

- a presentable object, the *AIS_InteractiveObject*
- a viewer
- an interactive context, the *AIS_InteractiveContext*.

The presentable object

The purpose of a presentable object is to supply the Graphic2d or Graphic3d structure of the object to be presented. On the first display request, it creates this structure by calling the appropriate algorithm and retaining this framework for further display.

Standard presentation algorithms are provided in the StdPrs and Prs3d packages. You can, however, write specific presentation algorithms of your own, provided that they create presentations made of structures from the Graphic2d or Graphic3d packages. You can also create several presentations of a single presentable object: one for each visualization mode supported by your application.

Each object to be presented individually must be presentable or associated with a presentable object.

The viewer

The viewer allows you to interactively manipulate views of the object. When you zoom, translate or rotate a view, the viewer operates on the graphic structure created by the presentable object and not on the data model of the application. Creating Graphic2d and Graphic3d structures in your presentation algorithms allows you to use the 2D and 3D viewers provided in Open CASCADE.

The Interactive Context

(see chapter 2, AIS: Application Interactive Services) The interactive context controls the entire presentation process from a common high-level API. When the application requests the display of an object, the interactive context requests the graphic structure from the presentable object and sends it to the viewer for display.

Presentation packages

Presentation involves at least the AIS, PrsMgr, StdPrs, V3d and V2d packages. Additional packages such as Prs3d, Graphic3d and Graphic2d may be used if you need to implement your own presentation algorithms.

AIS

See chapter 2, **AIS: Application Interactive Services** The *AIS* package provides all classes to implement presentable and selectable 3D objects.

PrsMgr

The *PrsMgr* package provides all the classes needed to implement the presentation process: the *Presentation* and *PresentableObject* abstract classes and the *PresentationManager2d* and *PresentationManager3d* concrete classes.

StdPrs

The *StdPrs* package provides ready-to-use standard presentation algorithms of points, curves and shapes of the geometry and topology toolkits.

V2d and V3d

The *V2d* and *V3d* packages provide the services supported by the 2D and 3D viewers.

Prs3d

The *Prs3d* package provides some generic presentation algorithms such as wireframe, shading and hidden line removal associated with a Drawer class which controls the attributes of the presentation to be created in terms of color, line type, thickness, and so on.

Graphic2d and Graphic3d

The *Graphic2d* and *Graphic3d* packages provide resources to create 2D and 3D graphic structures (refer to Chapters for more information).

2. 1. 2 A Basic Example: How to display a 3D object

Example

```
Void Standard_Real dx = ...; //Parameters
Void Standard_Real dy = ...; //to build a wedge
Void Standard_Real dz = ...;

Handle(V3d_Viewer)aViewer = ...;
Handle(AIS_InteractiveContext)aContext;
aContext = new AIS_InteractiveContext(aViewer);

BRepAPI_MakeWedge w(dx, dy, dz, Ctx);
TopoDS_Solid & = w.Solid();
Handle(AIS_Shape) anAis = new AIS_Shape(S);
```

```
//creation of the presentable object
aContext -> Display(anAis);
//Display the presentable object in the 3d viewer.
```

The shape is created using the *BRepAPI_MakeWedge* command. An AIS_Shape is then created from the shape. When calling the *Display (PresentableObject)* command, the interactive context calls the Compute method to calculate the presentation data and transfer it to the viewer. See Figure 2 below.

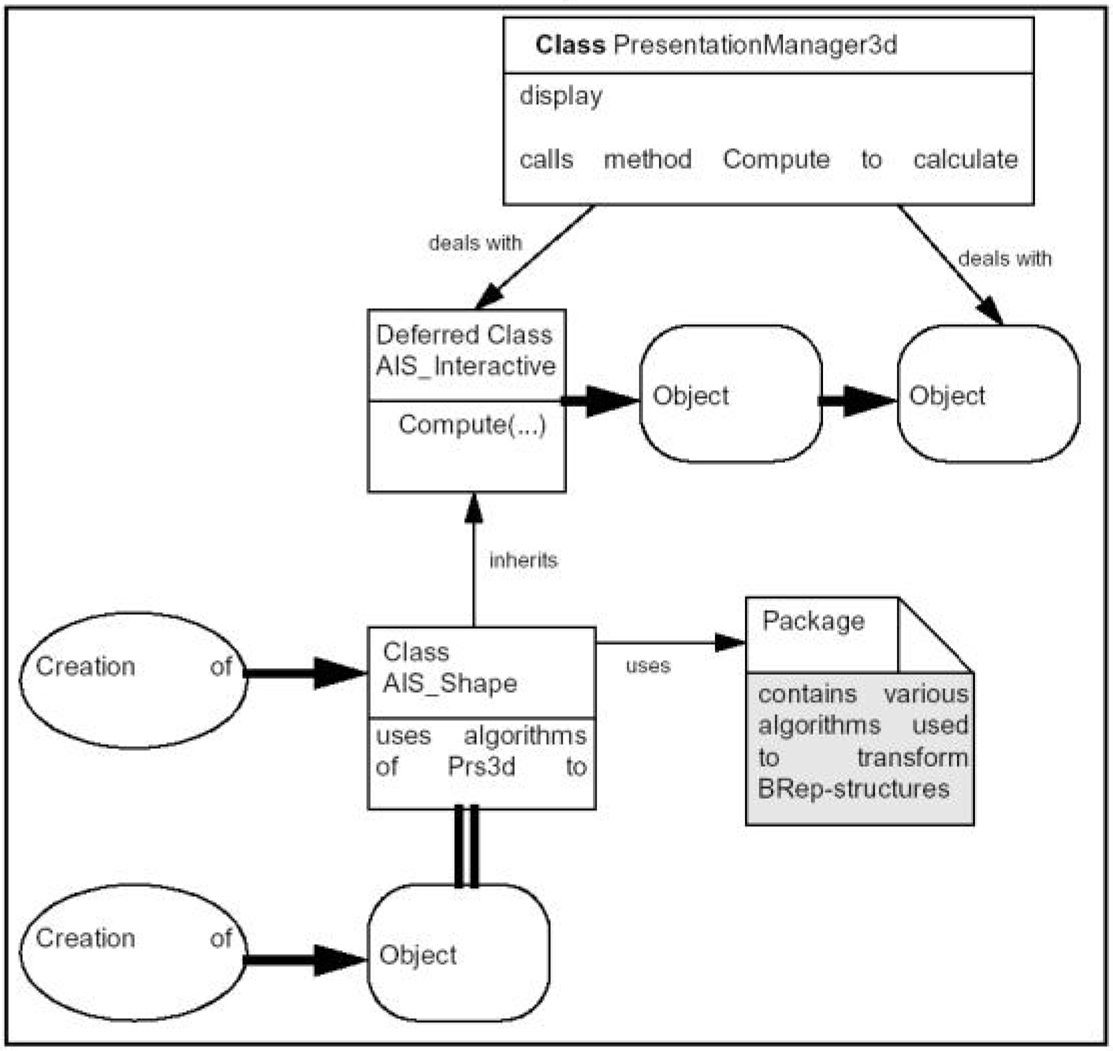

Class PresentationManager3d
display
calls method Compute to calculate
deals with
deals with
Deferred Class AIS_Interactive
Compute(...)
Object
Object
inherits
Creation of
Class AIS_Shape
uses algorithms of Prs3d to
uses
Package
contains various algorithms used to transform BRep-structures
Creation of
Object

Figure 2. Processes involved in displaying a presentable shape

2. 2 Selection

This chapter deals with the process used for selecting entities, which are displayed in the 2D space of the selection view.

2. 2. 1 The Selection Principle

Objects, which may be selected graphically, are displayed as sets of sensitive primitives, which provide sensitive zones in 2D graphic space. These zones are sorted according to their position on the screen when starting the selection process.

The position of the mouse is also associated with a sensitive zone. When moving within the window where objects are displayed, the areas touched by the zone of the mouse are analyzed. The owners of these areas are then highlighted or signaled by other means such as the name of the object highlighted in a list. That way, you are informed of the identity of the element detected.

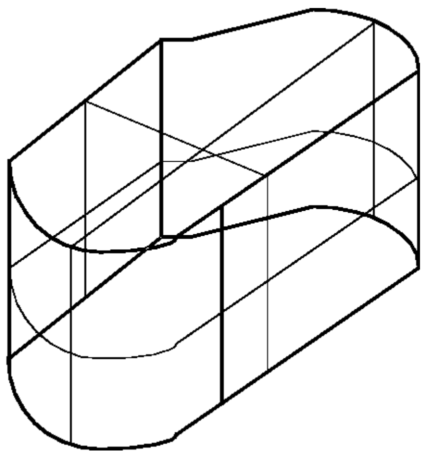


Figure 3. a model

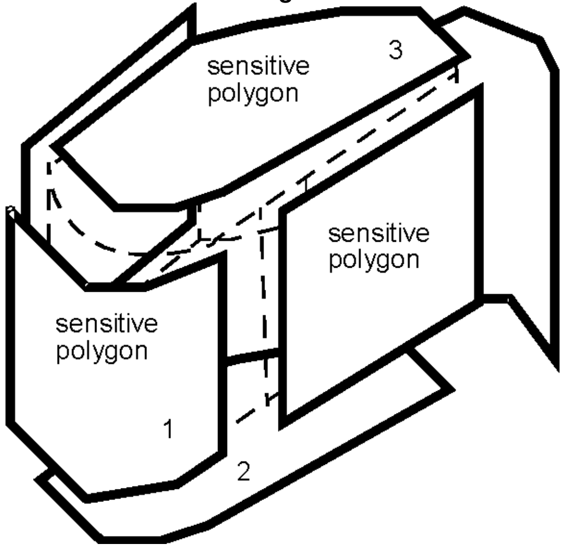


Figure 4. Modeling faces with sensitive primitives

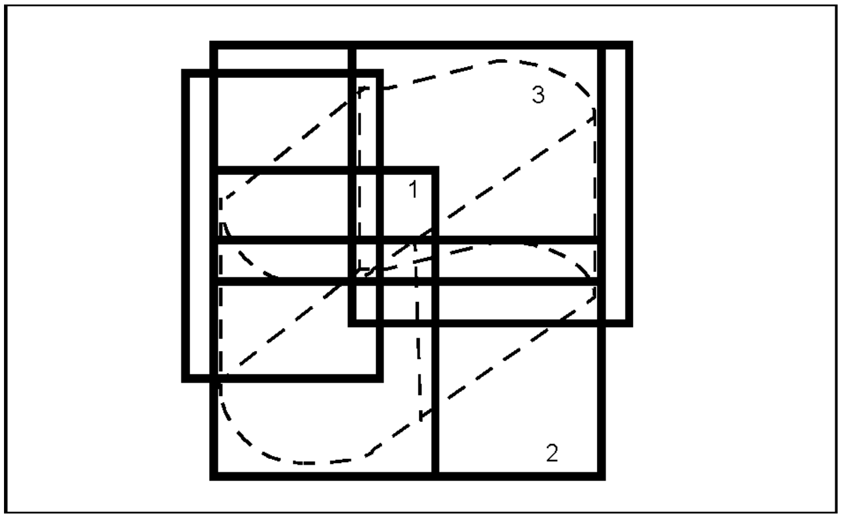


Figure 5. In dynamic selection, each sensitive polygon is represented by its bounding rectangle

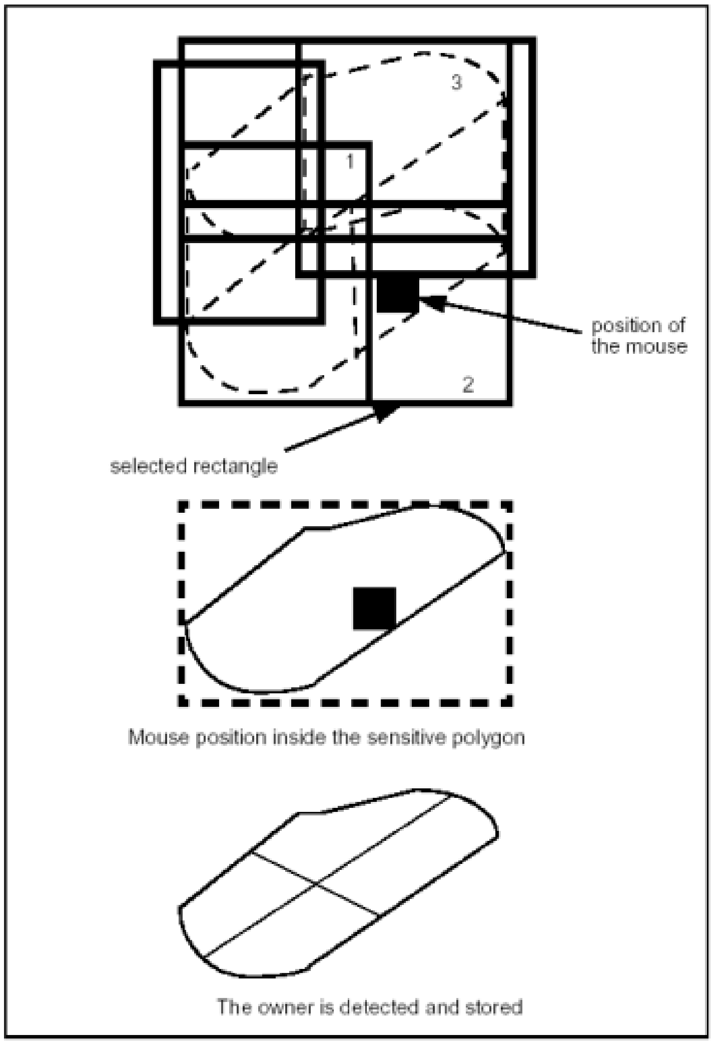

3
1
position of the mouse
2
selected rectangle
Mouse position inside the sensitive polygon
The owner is detected and stored

Figure 6. Reference to the sensitive primitive, then to the owner

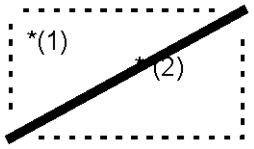
2. 2. 2 The Sensitive Primitive

The sensitive primitive - along with the entity owner - allow you to define what can be made selectable, and in so doing, provide the link between the applicative object and the sensitive zones defined by the 2D bounding boxes. For an object to be dynamically selectable, it has to be represented either as a sensitive primitive or a set of them. These give 2D boxes, which will be included in a sorting algorithm.

The use of 2D boxes allows a pre-selection of the detected entities. After pre-selection, the algorithm checks which sensitive primitives are actually detected. When detected, the primitives provide their owners' identity.

Example

The sensitive line segment below proposes a bounding box to the selector. During selection, positions 1 and 2 of the mouse detect the box but after sorting, only position 2 retains the line segment as selected by the algorithm.



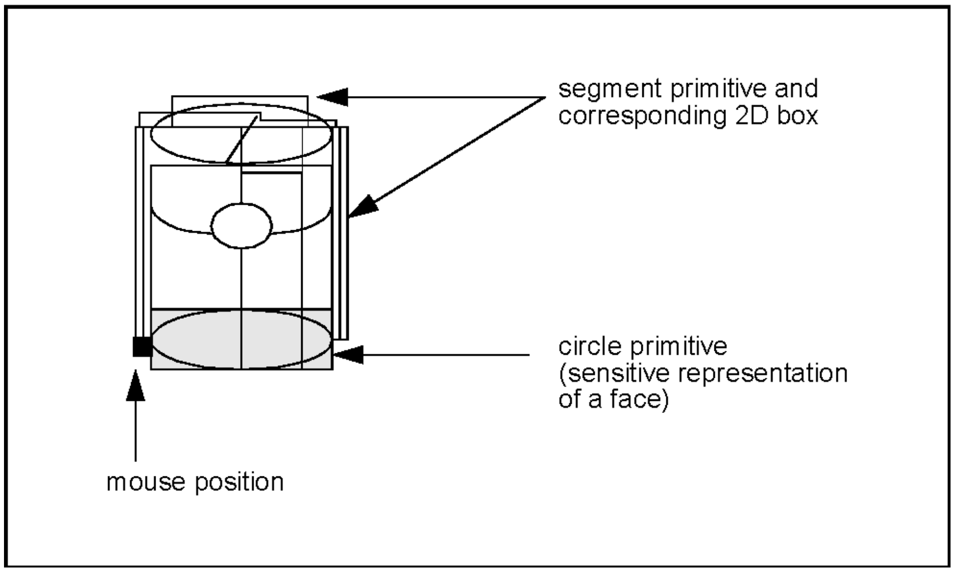

segment primitive and corresponding 2D box
circle primitive (sensitive representation of a face)
mouse position

Figure 7. Example of sensitive primitives

When the 2D box associated with the position of the mouse intersects the 2D box of a sensitive primitive, the owner of the sensitive primitive is called and its display is highlighted.

The notion of sensitive primitive is important for the developer when defining his own classes of sensitive primitives for the chosen selection modes. The classes must contain *Areas* and *Matches* functions. The former provides the list of 2D sensitive boxes representing the sensitive primitive at pre-selection and the latter determines if the detection of the primitive by the 2D boxes is valid.

2. 2. 3 The Principles of Dynamic Selection

Dynamic selection causes objects in a view to be automatically highlighted as the mouse cursor moves over them. This allows the user to be certain that the picked object is the correct one. Dynamic Selection is based on the following two concepts:

- a Selectable Object (see *AIS_InteractiveObject*)
- an Interactive Context

Selectable Object

A selectable object presents a given number of selection modes which can be redefined, and which will be activated or deactivated in the selection manager's selectors.

NOTE

The term, selection mode of a selectable object, can refer to the selection mode of the object itself or to that of one of its parts.

For each selection mode, a *SelectMgr_Selection* object class is included in the selectable object. (Each selection mode establishes a priority of selection for each class of selectable object defined.)

The notion of SELECTION is comparable to the notion of DISPLAY. Just as a display contains a set of graphic primitives that allow display of the entity in a specific display mode, a SELECTION contains a set of sensitive primitives, which allow detection of the entities they are associated with.

Interactive Context

See chapter 2, AIS: Application Interactive Services, Section 2.4

The interactive context is used to manage both selectable objects and selection processes.

Selection modes may be activated or de-activated for given selectable objects. Information is then provided about the status of activated/de-activated selection modes for a given object in a given selector.

Example

Let's consider the 3D selectable shape object, which corresponds to a topological shape.

For this class, seven selection modes can be defined:

mode 0 - selection of the shape itself

mode 1 - selection of vertices

mode 2 - selection of edges

mode 3 - selection of wires

mode 4 - selection of faces

mode 5 - selection of shells

mode 6 - selection of solids

mode 7 - selection of compounds

Selection 2 includes the sensitive primitives, which model all the edges of the shape. Each of these primitives contains a reference to the edge it represents.

The selections may be calculated before any activation and are graph independent as long as they are not activated in a given selector. Activation of selection mode 3 in a selector associated with a view V leads to the projection of the 3D sensitive primitives contained in the selection; then the 2D areas which represent the 2D bounding boxes of these primitives are provided to the sorting process of the selector containing all the detectable areas.

To deactivate selection mode 3 is to remove all those 2D areas.

Selection Packages

The selection packages are the following: *SelectBasics*, *Select2D*, *Select3D*, *SelectMgr*, *StdSelect*.

SelectBasics

The *SelectBasics* package contains the basic classes of the selection:

• the main definition of a sensitive primitive: *SensitiveEntity*

• the definition of a sensitive primitive owner: *EntityOwner*

• the algorithm used for sorting sensitive boxes: *SortAlgo*

Select2D

The *Select2D* package contains the basic classes of 2D sensitive primitives such as Points, Segments, and Circles, which inherit from *SensitiveEntity* from *SelectBasics* and used to represent 2D selectable objects from a dynamic selection viewpoint.

Select3D

The *Select3D* package contains all 3D standard sensitive primitives such as point, curve and face. All these classes inherit from 3D *SensitiveEntry* from *SelectBasics* with an additional method, which allows recovery of the bounding boxes in the 2D graphic selection space, if required. This package also includes the 3D-2D projector.

SelectMgr

The *SelectMgr* package is used to manage the whole dynamic selection process. It contains the *SelectableObject*, *Selection*, *SelectionManager*, and *ViewSelector* classes.

StdSelect

The *StdSelect* package provides standard uses of the classes described above and main tools used to prevent the developer from redefining the selection objects.

ViewSelector2D

Class defining a view selector for a view of the *V2d* package.

2. 2. 4 Methodology

Several operations must be performed prior to using dynamic selection:

1. Implement specific sensitive primitives if those defined in SELECT2D and SELECT3D are not sufficient. These primitives must inherit from *SensitiveEntity* from *SelectBasics* or from *SensitiveEntity* from *Select3D* when a projection from 3D to 2D is necessary.
2. Define all the owner types, which will be used, and the classes of selectable objects. ....i.e. the number of possible selection modes for these objects and the calculation of the decomposition of the object into sensitive primitives of all the primitives describing this mode. It is possible to define only one default selection mode for a selectable object if this object is to be selectable in a unique way.
3. Install the process, which provides the user with the identity of the owner of the detected entities in the selection loop.

When all these steps have been carried out, follow the procedure below:

1. Create an interactive context.
2. Create the selectable objects and calculate their various possible selections.

3. Load these selectable objects in the interactive context. The objects may be common to all the selectors, i.e. they will be seen by all the selectors in the selection manager, or local to one selector or more.
4. Activate or deactivate the objects' selection modes in the selector(s). When activating a selection mode in a selector for a given object, the manager sends the order to make the sensitive primitives in this selector selectable. If the primitives are to projected from 3D to 2D, the selector calls the specific method used to carry out this projection.

At this stage, the selection of selectable entities in the selectors is available.

The selection loop informs constantly the selectors with the position of the mouse and questions them about the detected entities.

2. 2. 5 Example of Use

Let's suppose you are creating an application that displays houses in a viewer of the V3d package and you want to select houses or parts of these houses (windows, doors, etc.) in the graphic window.

You define a selectable object called *House* and propose four possible selection modes for this object:

1 - selection of the house itself

2 - selection of the rooms

3 - selection of the walls

4 - selection of the doors.

You have to write the method, which calculates the four selections above, i.e. the sensitive primitives which are activated when the mode is.

You must define the class *Owner* specific to your application. This class will contain the reference to the house element it represents: wall, door or room. It inherits from *EntityOwner* from *SelectMgr*.

For example, let's consider a house with the following representation:

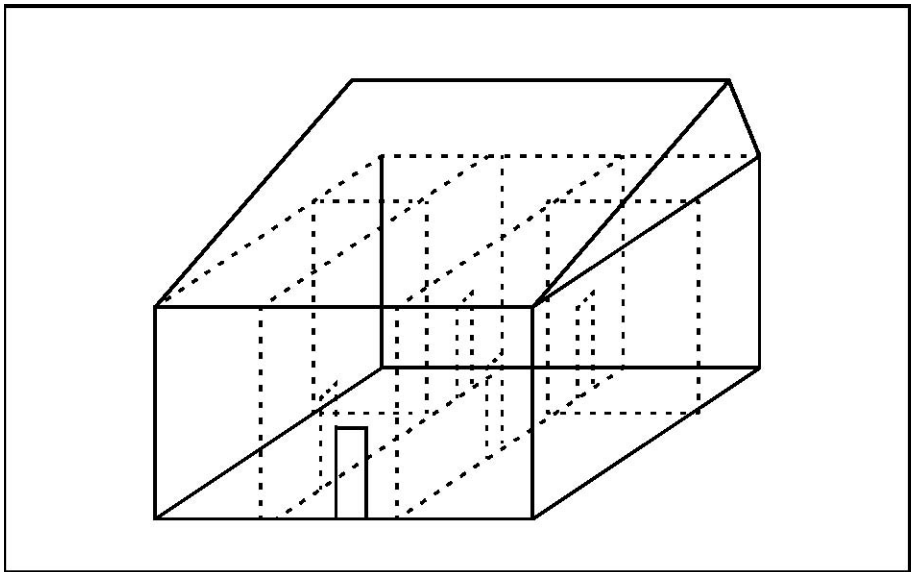


Figure 8. Selection of the rooms of a house

To build the selection, which corresponds to the mode “selection of the rooms” (selection 2 in the list of selection modes) use the following procedure:

Example

```
Void House::ComputeSelection
      (Const Handle(SelectMgr_Selection)& Sel,
       const Standard_Integer mode {
             switch(mode){
             case 0: //Selection of the rooms
             {
             for(Standard_Integer i = 1; i <= myNbRooms;
            i++)
             { //for every room, create an instance of the
            owner

               //along with the given room and its name.
            Handle(RoomOwner) aRoomOwner = new RoomOwner
            (Room(i), NameRoom(i)); //Room() returns a room
            and NameRoom() returns its name.
Handle(Select3d_SensitiveBox) aSensitiveBox;
aSensitiveBox = new Select3d_SensitiveBox
      (aRoomOwner, Xmin, Ymin, Zmin, Xmax, Ymax, Zmax);
             Sel -> Add(aSensitiveBox);
             }
             break;
             Case 1: ... //Selection of the doors
             } //Switch

) // ComputeSelection
```

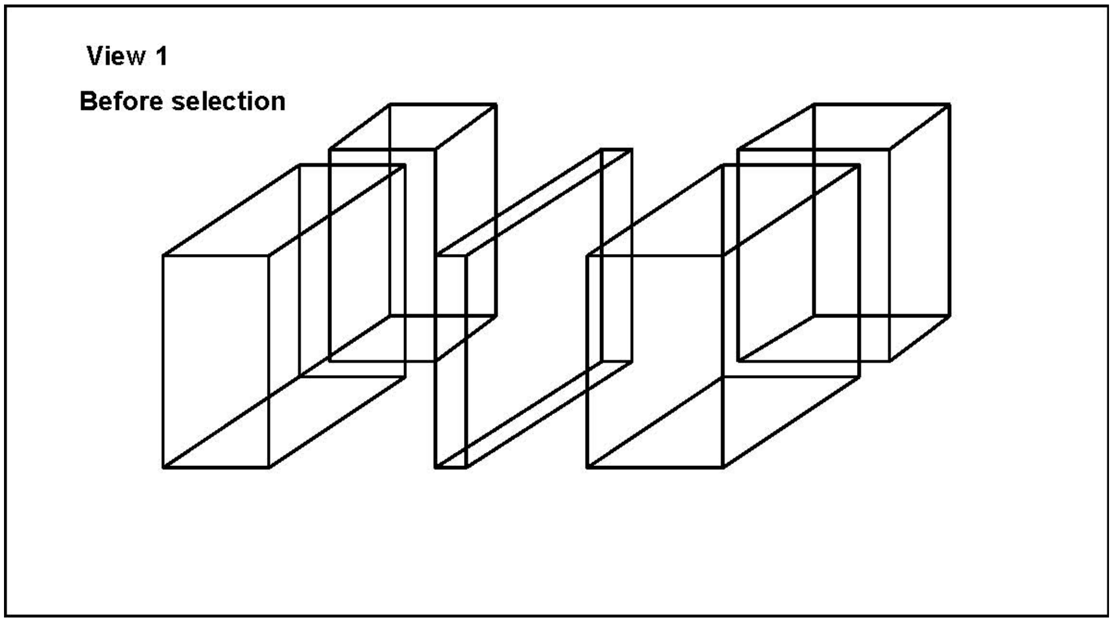


Figure 9. Activated sensitive boxes corresponding to selection mode 0 (selection of the rooms)

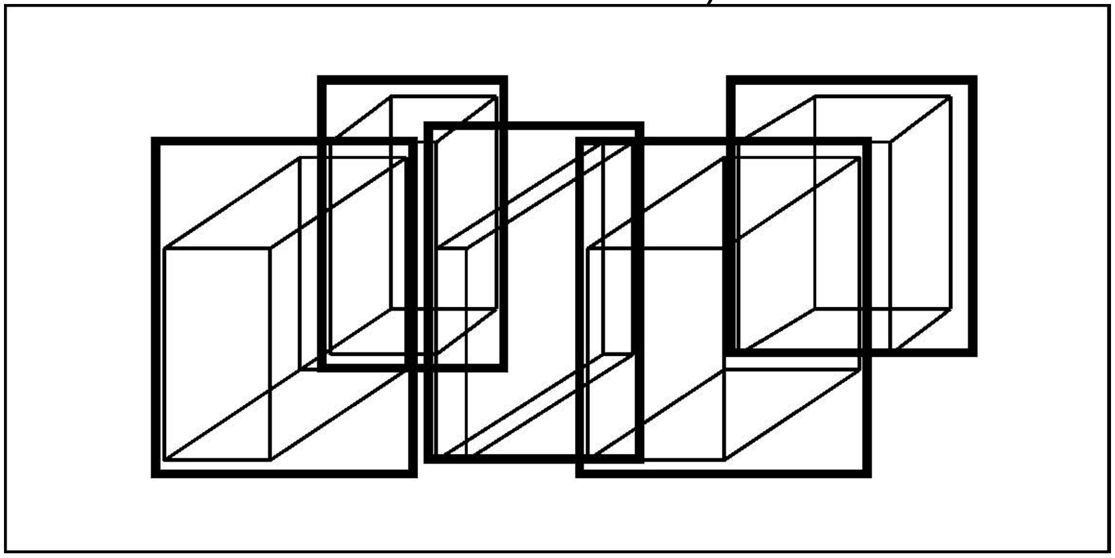
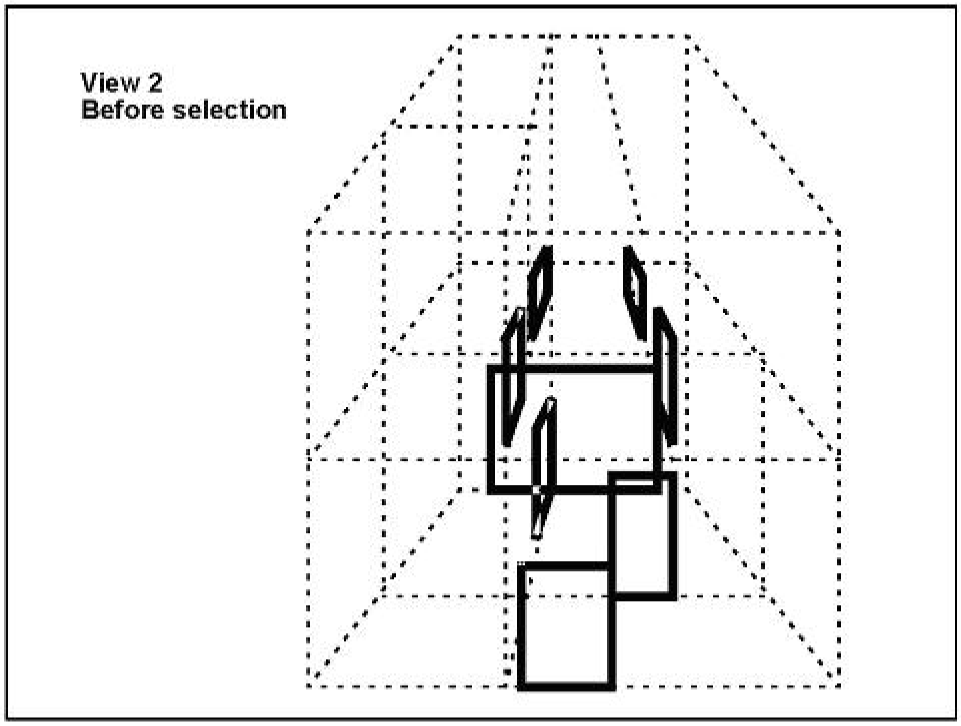


Figure 10. Activated sensitive rectangles in the selector during dynamic selection in view 1



(

Figure 11. Activated sensitive polygons corresponding to selection mode 1. (selection of the doors)

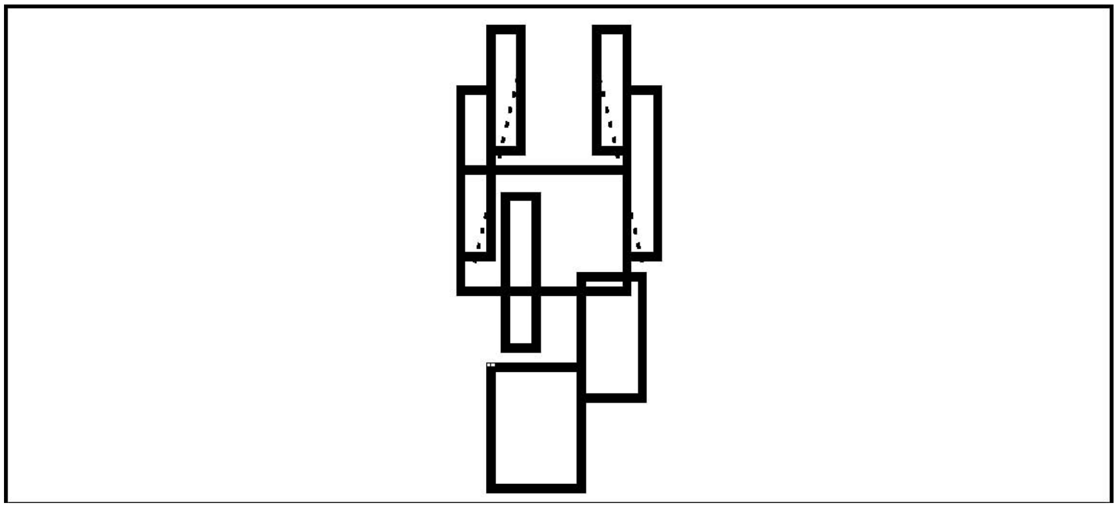


Figure 12. Sensitive rectangles in the selector during dynamic selection in view 2

3. AIS: Application Interactive Services

Application Interactive Services (**AIS**) offers a set of general services beyond those offered by basic Selection and Presentation packages such as **PrsMgr**, **SelectMgr** and **StdSelect**. These allow you to manage display and dynamic selection in a viewer simply and transparently. To use these services optimally, you should know various rules and conventions. Section I provides an overview of the important classes which you need to manipulate AIS well. Sections 2 and 3 explain in detail how to use them and how to implement them, as well as the rules and conventions to respect. The annexes offer various standard Interactive Objects in AIS, an example of an implementation of AIS and a reminder of how to manage presentation and selection.

3. 1 OVERVIEW

3. 1. 1 The Interactive Object

AIS_InteractiveObject

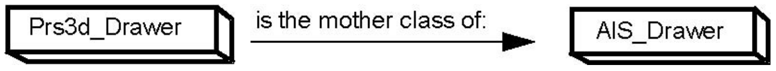
Entities, which are visualized and selected, are Interactive Objects. You can use classes of standard interactive objects for which all necessary functions have already been programmed, or you can implement your own classes of interactive objects, by respecting a certain number of rules and conventions described below.

3. 1. 2 Interactive Context/Local Context

AIS_InteractiveContext

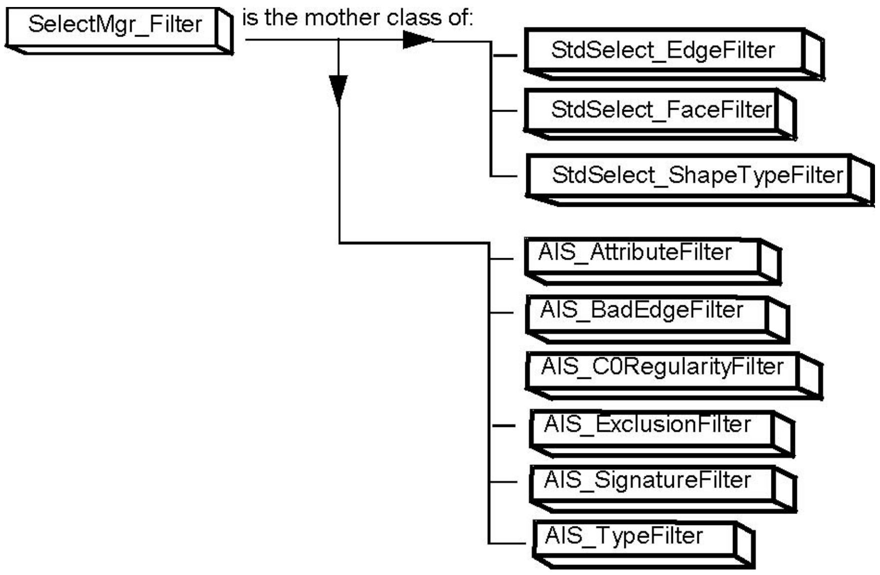
The central entity, which pilots visualizations and selections, is the Interactive Context. It is linked to a main viewer (and if need be, a trash bin viewer.) It has two operating modes: the Neutral Point and the local visualization and selection context. The neutral point, which is the default mode, allows you to easily visualize and select interactive objects, which have been loaded into the context. Opening Local Contexts allows you to prepare and use a temporary selection environment without disturbing the neutral point. A set of functions allows you to choose the interactive objects, which you want to act on, the selection modes, which you want to activate, and the temporary visualizations, which you will execute. When the operation is finished, you close the current local context and return to the state in which you were before opening it (neutral point or previous local context).

3. 1. 3 Graphic Attributes Manager or “Drawer”



An Interactive Object can have a certain number of graphic attributes which are specific to it (such as visualization mode, color and material) By the same token, the Interactive Context has a drawer which is valid by default for the objects it controls. When an interactive object is visualized, the required graphic attributes are first taken from its own Drawer if it has the ones required, or from the context drawer if it does not have them.

3. 1. 4 Selection Filters

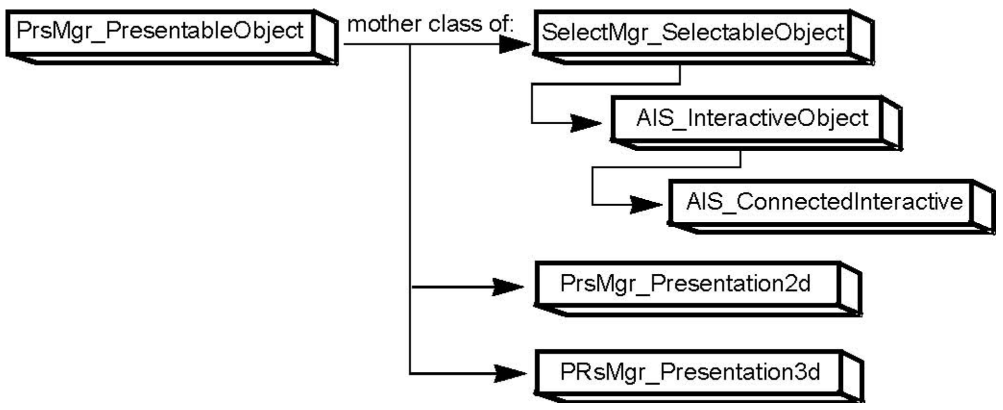


An important need in selection is the filtering of entities, which you want to select. Consequently there are FILTER entities, which allow you to refine the dynamic detection context, which you want to put into effect. Some of these filters can be used at the Neutral Point, others only in an open local context. A user will be able to program his own filters and load them into the interactive context.

3. 2 Rules and Conventions Governing Interactive Objects

An interactive object is a “virtual” entity, which can be presented and selected. It can also have its own visualization aspects such as color, material, and mode of visualization. In order to create and manipulate the interactive objects with ease, you must know the rules and conventions, which have been established. Several “virtual” functions must be implemented for these objects to have the behavior expected of them. A certain number of standard interactive objects, which respect the rules and conventions described below, have been implemented in AIS. The current list of them can be found in ANNEX I. The services that concern manipulation of presentations, selection and graphic attributes will be treated separately.

3. 2. 1 Presentations:



Conventions

- Either in 2D or in 3D, an interactive object can have as many presentations as its creator wants to give it.
- 3D presentations are managed by PresentationManager3D; 2D presentations by PresentationManager2D. As this is transparent in AIS, the user does not have to worry about it.
- A presentation is identified by an index and by the reference to the Presentation Manager which it depends on.
- By convention, the default mode of representation for the Interactive Object has index 0.

Virtual functions

Calculation of different presentations of an interactive object is done in the *Compute* functions inheriting from *PrsMgr_ PresentableObject::Compute* functions. They are automatically called by *PresentationManager* at a visualization or an update request.

If you are creating your own type of interactive object, you must implement the Compute function in one of the following ways:

- **For 2D:**

Example

```
void PackageName_ClassName::Compute
      (const Handle(PrsMgr_PresentationManager2d)&
                            aPresentationManager,
       const Handle(Graphic2d_)& aGraphicObject,
       const Standard_Integer aMode = 0);
```

- **For 3D:**

Example

```
void PackageName_ClassName::Compute
      (const Handle(PrsMgr_PresentationManager3d)&
                              aPresentationManager,
       const Handle(Prs3d_Presentation)& aPresentation,
       const Standard_Integer aMode = 0);
```

- **For hidden line mode3D parts (*):**

Example

```
void PackageName_ClassName::Compute
      (const Handle(Prs3d_Projector)& aProjector,
       const Handle(Prs3d_Presentation)& aPresentation);
```

WARNING (*)

As its call is automatically ordered by a view, this function requires explanation; the view has two states: degenerate mode (normal mode) and non-degenerate mode (Hidden line mode). When the latter is active, the view looks for all presentations displayed in normal mode, which have been signaled as accepting hidden line mode. An internal mechanism allows us to call the interactive object's own *Compute*, that is, projector, function. How do you declare that such and such a presentation will accept an "equivalent" in hidden line mode? By convention, it is the Interactive Object, which accepts or rejects the representation of hidden-line mode. You can make this declaration in one of two ways, either initially by using one of the values of the enumeration PrsMgr_TypeOfPresentation:

- PrsMgr_TOP_AllView,
- PrsMgr_TOP_ProjectorDependant

or later on, by using the function:

- • PrsMgr_PresentableObject::SetTypeOfPresentation

3. 2. 2 Important Specifics of AIS:

There are four types of interactive object in AIS:

- the "construction element" or Datum,
- the Relation (dimensions and constraints)

- the Object
- the None type (when the object is of an unknown type).

Inside these categories, additional characterization is available by means of a signature (an index.) By default, the interactive object has a NONE type and a signature of 0 (equivalent to NONE.) If you want to give a particular type and signature to your interactive object, you must redefine two virtual functions:

- `AIS_InteractiveObject::Type`
- `AIS_InteractiveObject::Signature.`

WARNING

Some signatures have already been used by “standard” objects delivered in AIS. (see the list of standard objects, Annex I.)

As will be seen below, the interactive context can have a default mode of representation for the set of interactive objects. This mode may not be accepted by a given class of objects. Consequently, a virtual function allowing you to get information about this class must be implemented:

- `AIS_InteractiveObject::AcceptDisplayMode.`

Services You Should Know

Display Mode: An object can have its own temporary display mode, which is different from that proposed by the interactive context. The functions to use are:

- `AIS_InteractiveContext::SetDisplayMode`
- `AIS_InteractiveContext::UnsetDisplayMode.`

You can, if you want, change the default index. To do this, implement the following virtual function:

- `AIS_InteractiveObject::DefaultDisplayMode.`

Hilight Mode: At dynamic detection, the presentation echoed by the Interactive Context, is by default the presentation already on the screen. You can always specify a highlight presentation mode, which is valid no matter what the active representation of the object. It makes no difference whether this choice is temporary or definitive. To do this, you use the following functions:

- `AIS_InteractiveObject::SetHilightMode`
- `AIS_InteractiveObject::UnSetHilightMode`

An example: For a shape - whether it is visualized in wireframe presentation or with shading - you want to systematically highlight the wireframe presentation. Consequently, you set the highlight mode to *0* in the constructor of the interactive object. You mustn’t forget to effect the implementation of this representation mode in the *Compute* functions.

Infinite Status: If you don’t want an object to be affected by a FitAll view, you must declare it infinite; you can cancel its “infinite” status in the same way.

- AIS_InteractiveObject::SetInfiniteState
- AIS_InteractiveObject::IsInfinite

Example

Let’s take the case of a class called IShape, representing an interactive object

```
myPk_IShape::myPK_IShape
      (const TopoDS_Shape& SH, PrsMgr_TypeOfPresentation
aType):

      AIS_InteractiveObject(aType),
      myShape(SH),
      myDrwr(new AIS_Drawer())
{
      SetHilightMode(0);
}
void myPk_IShape::Compute
      (const Handle(PrsMgr_PresentationManager3d) & PM,
       const Handle(Prs3d_Presentation)& P,
       const Standard_Integer TheMode)
{
      switch (TheMode){

      case 0:
            StdPrs_WFDeflectionShape::Add
      (P,myShape,myDrwr);
            //algo for calculation of wireframe
      presentation break;

      case 1:
            StdPrs_ShadedShape::Add (P,myShape,myDrwr);
            //algo for calculation of shading presentation.
            break;
      }
}
void myPk_IsShape::Compute
      (const Handle(Prs3d_Projector)& Prj,
      const Handle(Prs3d_Presentation) P)
{
      StdPrs_HLRPolyShape::Add(P,myShape,myDrwr);
      //Cas-cade hidden line mode calculation algorithm
}
```

3. 3 Selections

3. 3. 1 Conventions

An interactive object can have an indefinite number of modes of selection, each representing a “decomposition” into sensitive primitives; each primitive has an Owner (*SelectMgr_EntityOwner*) which allows us to identify the exact entity which has been detected (see ANNEX II).

Each Selection mode is identified by an index. The set of sensitive primitives, which correspond to a given mode, is stocked in a SELECTION (*SelectMgr_Selection*).

By Convention, the default selection mode which allows us to grasp the Interactive object in its entirety will be mode *0*.

3. 3. 2 Virtual functions

The calculation of Selection primitives (or sensitive primitives) is done by the intermediary of a virtual function, *ComputeSelection*. This should be implemented for each type of interactive object on which you want to make different type selections using the following function:

- `AIS_ConnectedInteractive::ComputeSelection`

A detailed explanation of the mechanism and the manner of implementing this function has been given in ANNEX II.

Moreover, just as the most frequently manipulated entity is TopoDS_Shape, the most used Interactive Object is AIS_Shape. You will see below that activation functions for standard selection modes are proposed in the Interactive context (selection by vertex, by edges etc.). To create new classes of interactive object with the same behavior as AIS_Shape - such as vertices and edges - you must redefine the virtual function:

- `AIS_ConnectedInteractive::AcceptShapeDecomposition.`

3. 3. 3 Other Services

You can change the default selection mode index of an Interactive Object. For instance, you can:

- check to see if there is a selection mode
- check the current selection mode
- set a selection mode
- unset a selection mode.

The following functions are concerned:

- `AIS_InteractiveObject::HasSelectionMode`
- `AIS_InteractiveObject::SelectionMode`
- `AIS_InteractiveContext::SetSelectionMode`
- `AIS_InteractiveContext::UnsetSelectionMode`

These functions are only of interest if you decide that the *0* mode adopted by convention will not do. In the same way, you can temporarily change the priority of certain interactive objects for selection of 0 mode. You could do this to make it easier to detect them graphically. You can:

- check to see if there is a selection priority setting for the owner
- check the current priority
- set a priority
- unset the priority.

To do this, you use the following functions:

- `AIS_InteractiveObject::HasSelectionPriority`
- `AIS_InteractiveObject::SelectionPriority`
- `AIS_InteractiveObject::SetSelectionPriority`
- `AIS_InteractiveObject::UnsetSelectionPriority`

3. 3. 4 Graphic attributes of an interactive object.

Keep in mind the following points concerning graphic attributes:

- Each interactive object can have its own visualization attributes.
- The set of graphic attributes of an interactive object is stocked in an *AIS_Drawer*, which is only a *Prs3d_Drawer* with the possibility of a link to another drawer
- By default, the interactive object takes the graphic attributes of the context in which it is visualized (visualization mode, deflection values for the calculation of presentations, number of isoparameters, color, type of line, material, etc.)
- In the *AIS_InteractiveObject* abstract class, several standard attributes have been privileged. These include: color, thickness of line, material, and transparency. Consequently, a certain number of virtual functions, which allow us to act on these attributes, have been proposed. Each new class of interactive object must redefine these functions in order to bring about the changes it should produce in the behavior of the class.

Change:
AISPoint::SetColor
- define:
AIS_Drawer::PointAspect

Figure 13. Redefinition of virtual functions for changes in AIS_Point

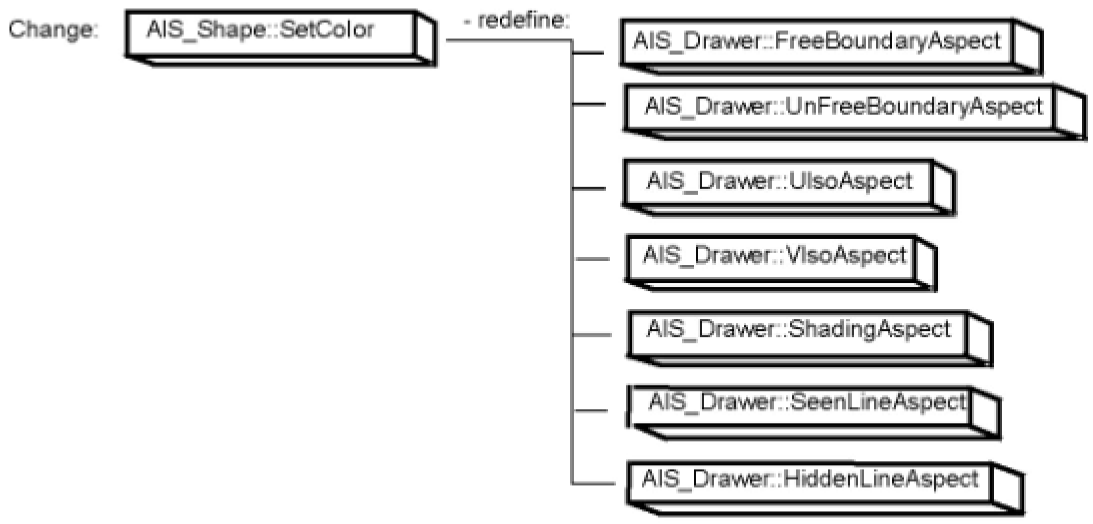

Change:
AIS_Shape::SetColor
- redefine:
AIS_Drawer::FreeBoundaryAspect
AIS_Drawer::UnFreeBoundaryAspect
AIS_Drawer::UIsoAspect
AIS_Drawer::VIsoAspect
AIS_Drawer::ShadingAspect
AIS_Drawer::SeenLineAspect
AIS_Drawer::HiddenLineAspect

Figure 14. Redefinition of virtual functions for changes in AIS_Shape.

The virtual functions concerned here allow you to provide settings for:

- color
- width
- material
- transparency

The functions concerned are the following:

- `AIS_InteractiveObject::UnsetColor`
- `AIS_InteractiveObject::SetWidth`
- `AIS_InteractiveObject::UnsetWidth`
- `AIS_InteractiveObject::SetMaterial (const Graphic3d_NameOfPhysicalMaterial & aName)`
- `AIS_InteractiveObject::SetMaterial (const Graphic3d_MaterialAspect & aMat)`
- `AIS_InteractiveObject::UnsetMaterial`
- `AIS_InteractiveObject::SetTransparency`
- `AIS_InteractiveObject::UnsetTransparency`

For other types of attribute, it is appropriate to change the Drawer of the object directly using:

- `AIS_InteractiveObject::SetAttributes`
- `AIS_InteractiveObject::UnsetAttributes`

3. 3. 5 Manipulation of Attributes

Some of these functions may imply the recalculation of presentations of the object. It is important to know which ones. If an interactive object's presentation mode is to be updated, a flag from *PrsMgr_PresentableObject* indicates this. The mode should be updated using the functions *Display* and *Redisplay* in *AIS_InteractiveContext*.

3. 3. 6 Complementary Services - Precautions

When using complementary services for interactive objects, pay special attention to the following cases:

Functions allowing us to temporarily "move" the representation and selection of Interactive Objects in a view without recalculation.

- `AIS_InteractiveContext::SetLocation`
- `AIS_InteractiveContext::ResetLocation`
- `AIS_InteractiveContext::HasLocation`
- `AIS_InteractiveContext::Location`

How you link applicative entities to interactive objects.

Each Interactive Object has functions, which allow us to attribute it an Owner in the form of a Transient.

- AIS_InteractiveObject::SetOwner
- AIS_InteractiveObject::HasOwner
- AIS_InteractiveObject::Owner

An interactive object can therefore be associated with an applicative entity or not, without this affecting its behavior.

3. 4 THE INTERACTIVE CONTEXT

3. 4. 1 PRELIMINARY RULES

The Interactive Context allows us to manage in a transparent way, the graphic and "selectable" behavior of interactive objects in one or more viewers. Most functions which allow us to modify the attributes of interactive objects, and which were presented in the preceding chapter, will be looked at again here.

There is one essential rule to follow: the modification of an interactive object, which is already known by the Context, must be done using Context functions. You can only directly call the functions available for an interactive object if it has not been loaded into an Interactive Context.

Example

```
Handle (AIS_Shape) TheAISShape = new AIS_Shape (ashape);
myIntContext->Display(TheAISShape);
myIntContext->SetDisplayMode(TheAISShape ,1);
myIntContext->SetColor(TheAISShape,Quantity_NOC_RED);
```

//but you can write

```
Handle (AIS_Shape) TheAISShape = new AIS_Shape (ashape);
TheAISShape->SetColor(Quantity_NOC_RED);
TheAISShape->SetDisplayMode(1);
myIntContext->Display(TheAISShape);
```

3. 4. 2 Groups of functions

You must distinguish two states in the Interactive Context:

- No Open Local Context; which will be referred to as Neutral Point.
- One or several open local contexts, each representing a temporary state of selection and presentation.

Some functions can only be used in open Local Context; others in closed local context; others do not have the same behavior in one state as in the other.

The Interactive Context is composed of a great many functions, which can be conveniently grouped according to theme:

- management proper to the context
- management in the local context
- presentations and selection in open/closed context
- selection strictly speaking

3. 4. 3 Management proper to the Interactive Context

The Interactive Context is made up of a Principal Viewer and, optionally, a trash bin or “Collector” Viewer. It also has a group of adjustable settings allowing you to personalize the behavior of presentations and selections:

- Default Drawer, containing all the color and line attributes which can be used by interactive objects, which do not have their own attributes.
- Default Visualization Mode for interactive objects

  Default: mode 0
- Highlight color of entities detected by mouse movement

  Default: Quantity_NOC_CYAN1
- Preselection color

  Default: Quantity_NOC_GREEN
- Selection color (when you click on a detected object)

  Default: Quantity_NOC_GRAY80
- Sub-Intensity color

  Default: Quantity_NOC_GRAY40

All of these settings can be modified by functions proper to the Context.

When you change a graphic attribute pertaining to the Context (visualization mode, for example), all interactive objects, which do not have the corresponding appropriate attribute, are updated. The Interactive Context is made up of a Principal Viewer and, optionally, a trash bin or “Collector” Viewer. It also has a group of adjustable settings allowing you to personalize the behavior of presentations and selections:

Example

```
//obj1, obj2: 2 interactive objects.

TheCtx->Display(obj1,Standard_False); // False = no update
of viewer.
TheCtx->Display(obj2,Standard_True); // True = Update of
viewer
TheCtx->SetDisplayMode(obj1,3,Standard_False);
TheCtx->SetDisplayMode(2);
// obj2 is visualised in mode 2 (if it accepts this mode)
// obj1 stays visualised in its mode 3.
```

To the main Viewer, are associated a *PresentationManager3D* and a *Selector3D* which manage the presentation and selection of present interactive objects. The same is true of the optional Collector. As we shall see, this management is completely transparent for the user.

3. 5 Management of Local Context

3. 5. 1 Rules and Conventions

- Opening a local context allows you to prepare an environment for temporary presentations and selections, which will disappear once the local context is closed.
- It is possible to open several local contexts, but only the last one will be active.
- When you close a local context, the one before, which is still on the stack, reactivates. If none is left, you return to Neutral Point.
- Each local context has an index created when the context opens. You should close the local context, which you have opened.

3. 5. 2 Important functionality

The interactive object, which is used the most by applications, is *AIS_Shape*. Consequently, standard functions are available which allow you to easily prepare selection operations on the constituent elements of shapes (selection of vertices, edges, faces etc) in an open local context. The selection modes specific to “Shape” type objects are called **Standard Activation Mode**. These modes are only taken into account in open local context and only act on interactive objects which have redefined the virtual function *AcceptShapeDecomposition()* so that it returns *TRUE*.

- Objects, which are temporarily in a local context, are not recognized by other local contexts a priori. Only objects visualized in Neutral Point are recognized by all local contexts.
- The state of a temporary interactive object in a local context can only be modified while another local context is open (except for one special case - see III.4.2)

WARNING

The specific modes of selection only concern the interactive objects, which are present in the Principal Viewer. In the Collector, you can only locate interactive objects, which answer positively to the positioned filters when a local context is open. Under no circumstances are they decomposed in standard mode etc.

3. 5. 3 Use

Opening and closing a local context are easy to put into operation:

- `AIS_InteractiveContext::OpenLocalContext`

The options available allow you to control what you want to do:

- *UseDisplayedObjects*: allows you to load or not load the interactive objects visualized at Neutral Point in the local context, which you open. If *FALSE*, the local context is empty after being opened. If *TRUE*, the objects at Neutral Point are modified by their default selection mode.

- *AllowShapeDecomposition*: AIS_Shape allows or prevents decomposition in standard shape location mode of objects at Neutral Point, which are type-"privileged" (see selection chapter). This Flag is only taken into account when *UseDisplayedObjects* is *TRUE*.
- *AcceptEraseOfObjects*: authorises other local contexts to erase the interactive objects present in this context. This option is rarely used. The last option has no current use.

This function returns the index of the created local context. It should be kept and used when the context is closed. Closing Local Contexts is done by:

- `AIS_InteractiveContext::CloseLocalContext`
- `AIS_InteractiveContext::CloseAllContexts`

WARNING

When the index isn't specified in the first function, the current Context is closed. This option can be dangerous, as other Interactive Functions can open local contexts without necessarily warning the user.

For greater security, you have to close the context with the index given on opening. The second function allows you to close all open local contexts at one go. In this case, you find yourself directly at Neutral Point. To load objects visualized at Neutral Point into a local context or remove them from one:

- `AIS_InteractiveContext::UseDisplayedObjects`
- `AIS_InteractiveContext::NotUseDisplayedObjects`

To get the index of the current context, use the following function:

- `AIS_InteractiveContext::IndexOfCurrentLocal`

When you close a local context, all temporary interactive objects are erased (deleted), all selection modes concerning the context are cancelled, and all content filters are emptied.

3. 5. 4 Management of Presentations and Selections

You must distinguish between the Neutral Point and the Open Local Context states. Although the majority of visualization functions can be used in both situations, their behavior is different:

3. 5. 5 Presentation in Neutral Point

Neutral Point should be used to visualize the interactive objects, which represent and select an applicative entity. Visualization and Erasing orders are straightforward:

- `AIS_InteractiveContext::Display`

```
      (const Handle(AIS_InteractiveObject)& anIobj,
       const Standard_Boolean
updateviewer=Standard_True);
```

-
  ```
  AIS_InteractiveContext::Display
        (const Handle(AIS_InteractiveObject)& anIobj,
         const Standard_Integer amode,
         const Standard_Integer aSelectionMode,
         const Standard_Boolean
  updateviewer = Standard_True,
         const Standard_Boolean
  allowdecomposition = Standard_True);
  ```

- `AIS_InteractiveContext::Erase`
- `AIS_InteractiveContext::EraseMode`
- `AIS_InteractiveContext::ClearPrs`
- `AIS_InteractiveContext::Redisplay`
- `AIS_InteractiveContext::Remove`
- `AIS_InteractiveContext::EraseAll`
- `AIS_InteractiveContext::Hilight`
- `AIS_InteractiveContext::HilightWithColor`

3. 5. 6 Important Remarks:

Bear in mind the following points:

- The first Display function visualizes the object in its default mode, if it has one, or in the default context mode, if it has one. If it has neither, the function displays it in 0 presentation mode. The object's default selection mode is automatically activated (0 mode by convention).
- The second Display function should only be used in Neutral Point to visualize a supplementary mode for the object, which you can erase by EraseMode (...). You activate the selection mode. This is passed as an argument. By convention, if you do not want to activate a selection mode, you must set the *SelectionMode* argument to the value of -1. This function is especially interesting in open local context, as we will see below.
- In Neutral Point, it is unadvisable to activate other identification modes than the default selection mode. It is preferable to open a local context in order to activate particular selection modes.
- When you apply the **Erase** (Interactive object) function, the *PutIncollector* option, which is TRUE by default, allows you to visualize the object directly in the Collector and makes it selectable (by activation of 0 mode). You can nonetheless block its passage through the Collector by changing the value of this option. In this case, the object is present in the Interactive Context, but is not seen anywhere.
- Modifications of visualization attributes and graphic behavior is effected through a set of functions similar to those which are available for the interactive object (color, thickness of line, material, transparency, locations etc.) The context then manages immediate and deferred updates.

- It is important to take an Interactive Object out of the Context when it is no longer needed. This allows you to avoid keeping a not insignificant stock of data in memory. In Neutral Point, an Interactive Object remains known to the Context as long as the *Remove* function has not been called, even if it is not visualized.

3. 5. 7 Presentation in Local Context

In open local context, the Display functions presented above apply as well.

WARNING

The function, AIS_InteractiveObject::Display, automatically activates the object's default selection mode. When you only want to visualize an Interactive Object in open Context, you must call the second function:

```
AIS_InteractiveContext::Display.
```

You can activate or deactivate specific selection modes in local open context in several different ways:

Use the Display functions with the appropriate modes

Activate standard mode:

- `AIS_InteractiveContext::ActivateStandardMode`
  only if a local Context is opened
- `AIS_InteractiveContext::DeactivateStandardMode`
- `AIS_InteractiveContext::ActivatedStandardModes`
- `AIS_InteractiveContext::SetShapeDecomposition`

This has the effect of activating the corresponding selection mode for all objects in Local Context, which accept decomposition into sub-shapes. Every new Object which has been loaded into the interactive context and which answers these decomposition criteria is automatically activated according to these modes.

WARNING

If you have opened a local context by loading an object with the default options (AllowShapeDecomposition = Standard_True), all objects of the "Shape" type are also activated with the same modes. You can act on the state of these "Standard" objects by using SetShapeDecomposition(Status).

Load an interactive object by the following function:

- `AIS_InteractiveContext::Load.`

This function allows you to load an Interactive Object whether it is visualized or not with a given selection mode, and/or with the desired decomposition option. If *AllowDecomp=TRUE* and obviously, if the interactive object is of the “Shape” type, these “standard” selection modes will be automatically activated as a function of the modes present in the Local Context.

Directly activate/deactivate selection modes on an object:

- `AIS_InteractiveContext::Activate`
- `AIS_InteractiveContext::Deactivate.`

3. 5. 8 USE OF FILTERS

When Interactive objects have been “prepared” in local context, you can add rejection filters. The root class of objects is *SelectMgr_Filter*. The principle behind it is straightforward: a filter tests to see whether the owners (*SelectMgr_EntityOwner*) detected in mouse position by the Local context selector answer *OK*. If so, it is kept; if not, it is rejected.

You can therefore create your own class of filter objects by implementing the deferred function *IsOk()*:

Example

```
class MyFilter : public SelectMgr_Filter {
};
virtual Standard_Boolean MyFilter::IsOk
      (const Handle(SelectMgr_EntityOwner)& anObj) const =
0;
```

In *SelectMgr*, there are also Composition filters (AND Filters, OR Filters), which allow you to combine several filters. In InteractiveContext , all filters that you add are stocked in an OR filter (which answers *OK* if at least one filter answers *OK*).

There are Standard filters, which have already been implemented in several packages:

- `StdSelect_EdgeFilter`
  Filters acting on edges such as lines and circles
- `StdSelect_FaceFilter`
  Filters acting on faces such as planes, cylinders and spheres

- `StdSelect_ShapeTypeFilter`
  Filters shape types such as compounds, solids, shells and wires
- `AIS_TypeFilter`
  Acts on types of interactive objects
- `AIS_SignatureFilter`
  Acts on types and signatures of interactive objects
- `AIS_AttributeFilter`
  Acts on attributes of Interactive Objects such as color and width

Because there are specific behaviors on shapes, each new Filter class must, if necessary, redefine a function, which allows a Local Context to know if it acts on specific types of sub-shapes:

- `AIS_LocalContext::ActsOn.`

By default, this function answers *FALSE*.

WARNING

Only type filters are activated in Neutral Point. This is to make it possible to identify a specific type of visualized object. For filters to come into play, one or more object selection modes must be activated. There are several functions to manipulate filters:

- `AIS_InteractiveContext::AddFilter`
- `AIS_InteractiveContext::RemoveFilter`

to remove a filter passed as an argument.

- `AIS_InteractiveContext::RemoveFilters`

to remove all filters present.

- `AIS_InteractiveContext::Filters`

to get the list of filters active in a local context.

Example

```
myContext->OpenLocalContext(Standard_False);
// no object in neutral point is loaded

myContext->ActivateStandardMode(TopAbs_Face);
//activates decomposition of shapes into faces.
Handle (AIS_Shape) myAIShape = new AIS_Shape ( ATopoShape);

myContext->Display(myAIShape,1,-
1,Standard_True,Standard_True); //shading visualization mode, no
specific mode, authorization for //decomposition into sub-shapes. At this Stage,
myAIShape is decomposed into faces...

Handle(StdSelect_FaceFilter) Fil1= new
      StdSelect_FaceFilter(StdSelect_Revol);
```

```
Handle(StdSelect_FaceFilter) Fil2= new
      StdSelect_FaceFilter(StdSelect_Plane);

myContext->AddFilter(Fil1);
myContext->AddFilter(Fil2);
//only faces of revolution or planar faces will be selected

***

myContext->MoveTo( xpix,ypix,Vue);
// detects of mouse position
```

3. 5. 9 Selection strictly speaking.

Dynamic detection and selection are put into effect in a straightforward way. There are only a few conventions and functions to be familiar with. The functions are the same in neutral point and in open local context:

- AIS_InteractiveContext::MoveTo

  passes mouse position to Interactive Context selectors
- AIS_InteractiveContext::Select

  stocks what has been detected on the last MoveTo. Replaces the previously selected object. Empties the stack if nothing has been detected at the last move
- AIS_InteractiveContext::ShiftSelect

  if the object detected at the last move was not already selected , it is added to the list of those selected. If not, it is withdrawn. Nothing happens if you click on an empty area.
- AIS_InteractiveContext::Select

  selects everything found in the surrounding area
- AIS_InteractiveContext::ShiftSelect

  selects what was not previously in the list of selected, deselects those already present.

Highlighting is detected and selected entities are automatically managed by the Interactive Context, whether you are in neutral point or Local Context. The Highlight colors are those dealt with above. You can nonetheless disconnect this automatic mode if you want to manage this part yourself:

- `AIS_InteractiveContext::SetAutomaticHilight`
- `AIS_InteractiveContext::AutomaticHilight`

If there is no open local context, the objects selected are called CURRENT OBJECTS; SELECTED OBJECTS if there is one. Iterators allow entities to be recovered in either case. A set of functions allows you to manipulate the objects, which have been placed in these different lists.

WARNING

When a Local Context is open, you can select entities other than interactive objects (vertices, edges etc.) from decompositions in standard modes, or from activation in specific modes on specific interactive objects. Only interactive objects are stocked in the list of selected objects. You can question the Interactive context by moving the mouse. The following functions will allow you to:

- tell whether something has been detected
- tell whether it is a shape
- get the shape if the detected entity is one
- get the interactive object if the detected entity is one.

The following functions are concerned:

- `AIS_InteractiveContext::HasDetected`
- `AIS_InteractiveContext::HasDetectedShape`
- `AIS_InteractiveContext::DetectedShape`
- `AIS_InteractiveContext::DetectedInteractive`

After using the Select and ShiftSelect functions in Neutral Point, you can explore the list of selections, referred to as current objects in this context. You can:

- initiate a scan of this list
- extend the scan
- resume the scan
- get the name of the current object detected in the scan.

The following functions are concerned:

- `AIS_InteractiveContext::InitCurrent`
- `AIS_InteractiveContext::MoreCurrent`
- `AIS_InteractiveContext::NextCurrent`
- `AIS_InteractiveContext::Current`

You can:

- get the first current interactive object
- highlight current objects
- remove highlight from current objects
- empty the list of current objects in order to update it
- find the current object.

The following functions are concerned:

- `AIS_InteractiveContext::FirstCurrentObject`
- `AIS_InteractiveContext::HilightCurrents`
- `AIS_InteractiveContext::UnhilightCurrents`
- `AIS_InteractiveContext::ClearCurrents`

- `AIS_InteractiveContext::IsCurrent.`

In Local Context, you can explore the list of selected objects available. You can:

- initiate,
- extend,
- resume a scan, and then
- get the name of the selected object.

The following functions are concerned:

- `AIS_InteractiveContext::InitSelected`
- `AIS_InteractiveContext::MoreSelected`
- `AIS_InteractiveContext::NextSelected`
- `AIS_InteractiveContext::SelectedShape.`

You can:

- check to see if you have a selected shape, and if not,
- get the picked interactive object,
- check to see if the applicative object has an owner from Interactive attributed to it
- get the owner of the detected applicative entity
- get the name of the selected object.

The following functions are concerned:

- `AIS_InteractiveContext::HasSelectedShape`
- `AIS_InteractiveContext::Interactive`
- `AIS_InteractiveContext::HasApplicative`
- `AIS_InteractiveContext::Applicative`
- `AIS_InteractiveContext::IsSelected.`

Example

```
myAISCtx->InitSelected();
while (myAISCtx->MoreSelected())
      {
      if (myAISCtx->HasSelectedShape)
            {
            TopoDS_Shape ashape = myAISCtx-
      >SelectedShape();
            // to be able to use the picked shape
            }
      else
            {
```

```
            Handle_AIS_InteractiveObject aniobj = myAISCtx-
      >Interactive();
            // to be able to use the picked interactive object
            }
myAISCtx->NextSelected();
}
```

3. 5. 10 Remarks:

In Local Context and in the iteration loop, which allows you to recover selected entities, you have to ask whether you have selected a shape or an interactive object before you can recover the entity. If you have selected a Shape from TopoDS on decomposition in standard mode, the *Interactive ()* function returns the interactive object, which provided the selected shape. Other functions allow you to manipulate the content of Selected or Current Objects:

- erase selected objects
- display them,
- put them in the list of selections

The following functions are concerned:

- `AIS_InteractiveContext::EraseSelected`
- `AIS_InteractiveContext::DisplaySelected`
- `AIS_InteractiveContext::SetSelected`

You can also:

- take the list of selected objects from a local context and put it into the list of current objects in Neutral Point,
- add or remove an object from the list of selected entities,
- highlight and
- remove highlighting from a selected object
- empty the list of selected objects.

The following functions are concerned:

- `AIS_InteractiveContext::SetSelectedCurrent`
- `AIS_InteractiveContext::AddOrRemoveSelected`
- `AIS_InteractiveContext::HilightSelected`
- `AIS_InteractiveContext::UnhilightSelected`
- `AIS_InteractiveContext::ClearSelected`

You can highlight and remove highlighting from a current object, and empty the list of current objects.

- `AIS_InteractiveContext::HilightCurrents`
- `AIS_InteractiveContext::UnhilightCurrents`
- `AIS_InteractiveContext::ClearCurrents`

When you are in open Local Context, you may be lead to keep “temporary” interactive objects. This is possible using the following functions:

- `AIS_InteractiveContext::KeepTemporary`
- `AIS_InteractiveContext::SetSelectedCurrent`

The first function transfers the characteristics of the interactive object seen in its local context (visualization mode etc.) to the neutral point. When the local context is closed, the object does not disappear. The second allows the selected object to become the current object when you close the local context.

You can also want to modify in a general way the state of the local context before continuing a selection (emptying objects, removing filters, standard activation modes). To do that, you must use the following function:

- `AIS_InteractiveContext::ClearLocalContext`

3. 5. 11 Advice on Using Local Contexts

The possiblities of use for local contexts are numerous depending on the type of operation that you want to perform:

- working on all visualized interactive objects,
- working on only a few objects,
- working on a single object.

1. When you want to work on one type of entity, you should open a local context with the option UseDisplayedObjects set to FALSE. Some functions which allow you to recover the visualized interactive objects, which have a given Type, and Signature from the “Neutral Point” are:

```
AIS_InteractiveContext::DisplayedObjects
(AIS_ListOfInteractive& aListOfIO) const;

AIS_InteractiveContext::DisplayedObjects
(const AIS_KindOfInteractive WhichKind,
 const Standard_Integer WhichSignature,
   AIS_ListOfInteractive& aListOfIO) const;
```

At this stage, you only have to load the functions Load, Activate, and so on.

2. When you open a Local Context with default options, you must keep the following points in mind:

The Interactive Objects visualized at Neutral Point are activated with their default selection mode. You must deactivate those, which you do not want to use.

The Share Type Interactive Objects are automatically decomposed into sub-shapes when standard activation modes are launched.

The “temporary” Interactive Objects present in the Local Contexts are not automatically taken into account. You have to load them manually if you want to use them.

The stages could be the following:

1. Open a Local Context with the right options;
2. Load/Visualize the required complementary objects with the desired activation modes.
3. Activate Standard modes if necessary
4. Create its filters and add them to the Local Context
5. Detect/Select/recover the desired entities
6. Close the Local Context with the adequate index.

It is useful to create an INTERACTIVE EDITOR, to which you pass the Interactive Context. This will take care of setting up the different contexts of selection/presentation according to the operation, which you want to perform.

Example

You have visualized several types of interactive objects: *AIS_Points*, *AIS_Axes*, *AIS_Trihedrons*, and *AIS_Shapes*.

For your applicative function, you need an axis to create a revolved object. You could obtain this axis by identifying:

-an axis which is already visualized -2 points -a rectilinear edge on the shapes which are present

-a cylindrical face on the shapes (You will take the axis of this face)

```
myIHMEditor::myIHMEditor
      (const Handle(AIS_InteractiveContext)& Ctx,
       ....) :
       myCtx(Ctx),
      ...

{
}

myIHMEditor::PrepareContext()
{
```

```
myIndex =myCtx->OpenLocalContext();

//the filters

Handle(AIS_SignatureFilter) F1 = new
       AIS_SignatureFilter(AIS_KOI_Datum,AIS_SD_Point);
//filter on the points

Handle(AIS_SignatureFilter) F2 = new
      AIS_SignatureFilter(AIS_KOI_Datum,AIS_SD_Axis);
//filters on the axes.

Handle(StdSelect_FaceFilter) F3 = new
       StdSelect_FaceFilter(AIS_Cylinder);
//cylindrical face filters


//...

// activation of standard modes on the shapes..
myCtx->ActivateStandardMode(TopAbs_FACE);
myCtx->ActivateStandardMode(TopAbs_VERTEX);
myCTX->Add(F1);
myCTX->Add(F2);
myCTX->Add(F3);

// at this point, you can call the selection/detection function
}

void myIHMEditor::MoveTo(xpix,ypix,Vue)

{
myCTX->MoveTo(xpix,ypix,vue);
// the hilight of what is detected is automatic.
}

Standard_Boolean myIHMEditor::Select()
{
// returns true if you should continue the selection

      myCTX->Select();
      myCTX->InitSelected();
      if(myCTX->MoreSelected())
             {
             if(myCTX->HasSelectedShape())

            { const TopoDS_Shape& sh = myCTX-
      >SelectedShape();
            if( vertex){
                  if(myFirstV...)
                  {
```

```
            //if it's the first vertex, you stock it, then you deactivate
      the faces and only keep the filter on the points:
            mypoint1 = ....;
            myCtx->RemoveFilters();
            myCTX-
      >DeactivateStandardMode(TopAbs_FACE);
            myCtx->Add(F1);
            // the filter on the AIS_Points
            myFirstV = Standard_False;
            return Standard_True;
             }
            else
             {
            mypoint2 =...;
            // construction of the axis return Standard_False;
            }
       }
       else
        {
      //it is a cylindrical face : you recover the axis; visualize it; and
   stock it.
      return Standard_False;
      }
        }
      // it is not a shape but is no doubt a point.
   else
   {
   Handle(AIS_InteractiveObject)
      SelObj = myCTX->SelectedInteractive();
      if(SelObj->Type()==AIS_KOI_Datum)
      {
            if(SelObj->Signature()==1)
            {
                  if (firstPoint)
                  {
                  mypoint1 =...
                        return Standard_True;
                  }
                  else
                  {
                  mypoint2 = ...;
                  //construction of the axis, visualization, stocking
                  return Standard_False;
                  }
            }

            else
            {
                    // you have selected an axis; stock the axis
                    return Standard_False;
                    }
                    }
                    }
                    }
                    }
void myIHMEditor::Terminate()
{
myCtx->CloseLocalContext(myIndex);
...
}
```

ANNEX I: Standard Interactive Object Classes in AIS DATUMS:

AIS_Point
AIS_Axis
AIS_Line
AIS_Circle
AIS_Plane
AIS_Trihedron : 4 selection modes

- mode 0 : selection of a trihedron
- mode 1 : selection of the origin of the trihedron
- mode 2 : selection of the axes
- mode 3 : selection of the planes XOY, YOZ, XOZ

when you activate one of modes 1 2 3 4 , you pick AIS objects of type:

- AIS_Point
- AIS_Axis (and information on the type of axis)
- AIS_Plane (and information on the type of plane).

AIS_PlaneTrihedron offers 3 selection modes:

- mode 0 : selection of the whole trihedron
- mode 1 : selection of the origin of the trihedron
- mode 2 : selection of the axes - same remarks as for the Trihedron.

Warning

For the presentation of planes and trihedra, the default unit of length is millimeter, and the default value for the representation of axes is 100. If you modify these dimensions, you must temporarily recover the object DRAWER. From inside it, take the Aspects in which the values for length are stocked (PlaneAspect for the plane, FirstAxisAspect for trihedra), and change these values inside these Aspects. Finally, recalculate the presentation.

OBJECTS

AIS_Shape : 3 visualization modes :

- mode 0 : Line (default mode)
- mode 1 : Shading (depending on the type of shape)
- mode 2 : Bounding Box

7 maximum selection modes, depending on the complexity of the shape :

- • mode 0 : selection of the AIS_Shape
- • mode 1 : selection of the vertices
- • mode 2 : selection of the edges
- • mode 3 : selection of the wires

- • mode 4 : selection of the faces
- • mode 5 : selection of the shells
- • mode 6 : selection of the constituent solids.

AIS_ConnectedInteractive: Interactive Object connecting to another interactive object reference, and located elsewhere in the viewer makes it possible not to calculate presentation and selection, but to deduce them from your object reference.

AIS_ConnectedShape: Object connected to interactive objects having a shape; this class has the same decompositions as AIS_Shape. What's more, it allows a presentation of hidden parts, which are calculated automatically from the shape of its reference.

AIS_MultipleConnectedInteractive: Object connected to a list of interactive objects (which can also be Connected objects. It does not require memory hungry calculations of presentation

AIS_MultipleConnectedShape: Interactive Object connected to a list of interactive objects having a Shape (AIS_Shape, AIS_ConnectedShape, AIS_MultipleConnectedShape). The presentation of hidden parts is calculated automatically.

RELATIONS

The list is not exhaustive.

AIS_ConcentricRelation

AIS_FixRelation

AIS_IdenticRelation

AIS_ParallelRelation

AIS_PerpendicularRelation

AIS_Relation

AIS_SymmetricRelation

AIS_TangentRelation

DIMENSIONS

AIS_AngleDimension

AIS_Chamf2dDimension

AIS_Chamf3dDimension

AIS_DiameterDimension

AIS_DimensionOwner

AIS_LengthDimension

AIS_OffsetDimension

AIS_RadiusDimension

ANNEX II : Principles of Dynamic Selection

The idea of dynamic selection is to represent the entities, which you want to select by a bounding box in the actual 2D space of the selection view. The set of these zones is ordered by a powerful sorting algorithm. To then find the applicative entities actually detected at this position, all you have to do is read which rectangles are touched at mouse position (X,Y) of the view, and judiciously reject some of the entities which have provided these rectangles.

How to go from the objects to 2D boxes

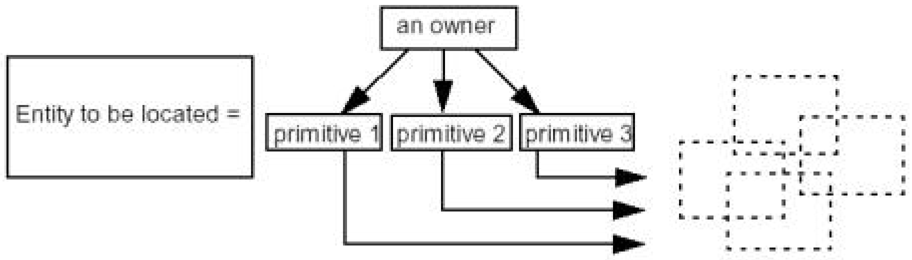
An intermediary stage consists in representing what you can make selectable by means of sensitive primitives and owners, entities of a high enough level to be known by the selector mechanisms.

The sensitive primitive is capable of:

- giving a 2D bounding box to the selector.
- answering the rejection criteria positively or negatively by a "Matches" function.
- being projected from 3D in the 2D space of the view if need be.
- returning the owner which it will represent in terms of selection.

A set of standard sensitive primitives exists in Select3D packages for 3D primitives, and Select2D for 2D primitives.

The owner is the entity, which makes it possible to link the sensitive primitives and the objects that you really wanted to detect. It stocks the diverse information, which makes it possible to find objects. An owner has a priority (*5* by default), which you can modulate, so as to make one entity more selectable than another.



Implementation in an interactive/selectable object

1. Define the number of selection modes possible, i.e. what you want to identify by activating each of the selection modes. Example: for an interactive object representing a topological shape,

   mode 0: selection of the interactive object itself

   mode 1: selection of the vertices

   mode 2: selection of the edges

   mode 3: selection of the wires

   mode 4: selection of the faces detectable

2. For each selection mode of an interactive object, “model” the set of entities, which you want to locate by these primitives and these owners.

3. There exists an “owner” root class, *SelectMgr_EntityOwne*r, containing a reference to a selectable object, which has created it. If you want to stock its information, you have to create classes derived from this root class. Example: for shapes, there is the *StdSelect_BRepOwner* class, which can save a TopoDS shape as a field as well as the Interactive Object.

4. The set of sensitive primitives which has been calculated for a given mode is stocked in *SelectMgr_Selection*.

5. For an Interactive object, the modeling is done in the *ComputeSelection* virtual function.

Example

Let an interactive object represent a box.

We are interested in having 2 location modes:

- mode 0: location of the whole box.
- mode 1: location of the edges on the box.

For the first mode, all sensitive primitives will have the same owner, which will represent the interactive object. In the second case, we have to create an owner for each edge, and this owner will have to contain the index for the edge, which it represents. You will create a class of owner, which derives from *SelectMgr_EntityOwner*.

The *ComputeSelection* function for the interactive box can have the following form:

```
void InteractiveBox::ComputeSelection
      (const Handle(SelectMgr_Selection)& Sel,
       const Standard_Integer Mode)
{
      switch(Mode)
```

```
            {
            case 0:
            //locating the whole box by making its faces sensitive...
            {
      Handle(SelectMgr_EntityOwner) Ownr = new
             SelectMgr_EntityOwner(this,5);
      for(Standard_Integer I=1;I<=Nbfaces;I++)
      {
      //Array is a TColgp_Array1OfPnt: which represents the array of vertices.
    Sensitivity is
Select3D_TypeOfSensitivity value
      Sel->Add(new
Select3D_SensitiveFace(Ownr,Array,Sensitivity));
      }
      break;
   }
  case 1:
      // locates the edges
      {
        for(Standard_Integer i=1;i<=12;i++)
            {
            // 1 owner per edge...
            Handle(mypk_EdgeOwner) Ownr =
                  new mypk_EdgeOwner(this,i,6);
                  //6->priority
                  Sel->Add(new Select3D_SensitiveSegment
                        (Ownr,firstpt(i),lastpt(i)));
                  }
                  break;
            }
      }
}
```

How It Works Concretely

Selectable objects are loaded in the selection manager, which has one or more selectors; in general, we suggest assigning one selector per viewer. All you have to do afterwards is to activate or deactivate the different selection modes for selectable objects. The *SelectionManager* looks after the call to the *ComputeSelection* functions for different objects. NOTE: This procedure is completely hidden if you use the interactive contexts of AIS (see section 3.3, Contexts)

Example

//We have several “ interactive boxes “ box1, box2, box3;

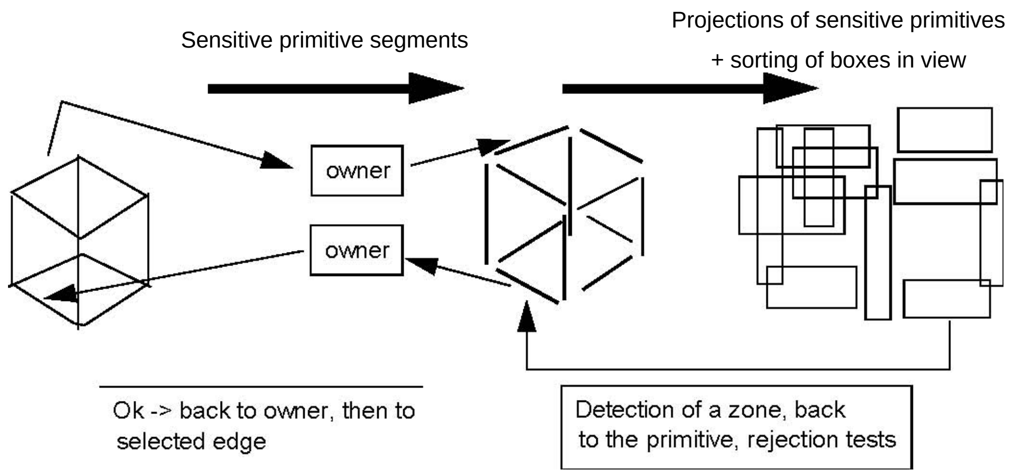
```
  Handle(SelectMgr_SelectionManager) SM = new
  SelectMgr_SelectionManager();
  Handle(StdSelect_ViewerSelector3d) VS = new
  StdSelect_ViewerSelector3d();

        SM->Add(VS);
        SM->Load(box1);SM->Load(box2);SM->Load(box3);
        // box load.
        SM->Activate(box1,0,VS);
        // activates mode 0 of box 1 in the selector VS
        SM->Activate(box1,1,VS);
        M->Activate(box3,1,VS);

VS->Pick(xpix,ypix,vue3d)
// detection of primitives by mouse position.

Handle(EntityOwner) POwnr = VS->OnePicked();
// picking of the "best" owner detected

for(VS->Init();VS->More();VS->Next())
  {
  VS->Picked();
  // picking of all owners detected
    }
  SM->Deactivate(box1);
  // deactivate all active modes of box1
```


Sensitive primitive segments
Projections of sensitive primitives
+ sorting of boxes in view
owner
owner
Ok -> back to owner, then to selected edge
Detection of a zone, back to the primitive, rejection tests

1st activation of the box’s mode 1: calculation of sensitive primitives + 3D/2D projection + sorting

deactivation of mode: only updated by sorting

rotation of the view: only projection + sorting of active primitives

modification of the box -> Recalculation of the active selection, recalculation flag on the inactive ones + 3D/2D projection + sorting

4. 3D Display

4. 1 Glossary of 3D terms

4. 1. 1 From Graphic3d

Primitive	A primitive is a drawable element. It has a definition in 3D space. Primitives can either be lines, faces, text, or markers. Once displayed markers and text remain the same size. Lines and faces can be modified e.g. zoomed. Primitives must be stored in a group.
Group	A set of primitives and attributes on those primitives. Primitives and attributes may be added to a group but cannot be removed from a group, except by erasing them globally. A group can have a pick identity.
Structure	Manages a set of groups. The groups are mutually exclusive. A structure can be edited, adding or removing groups. A structure can reference other structures to form a hierarchy. It has a default (identity) transformation and other transformations may be applied to it (rotation, translation, scale, etc). It has no default attributes for the primitive lines, faces, markers, and text. Attributes may be set in a structure but they are overridden by the attributes in each group. Each structure has a display priority associated with it, which rules the order in which it is redrawn in a 3D viewer. If the visualization mode is incompatible with the view it is not displayed in that view, e.g. a shading-only object is not visualized in a wireframe view.

4. 1. 2 From V3d

View	A view is defined by a view orientation, a view mapping, and a context view
Viewer	Manages a set of views.
View orientation	Defines the manner in which the observer looks at the scene in terms of View Reference Coordinates.
View mapping	Defines the transformation from View Reference Coordinates to the Normalized Projection Coordinates. This follows the Phigs scheme.
Light	There are five kinds of light source -ambient, headlight, directional, positional and spot. The light is only activated in a shading context in a view.

Depth-cueing	Reduces the color intensity for the portion of an object further away from the eye to give the impression of depth. This is used for wireframe objects. Shaded objects do not require this.
Z-Buffering	This is a form of hidden surface removal in shading mode only. This is always active for a view in the shading mode. It cannot be suppressed.
Anti-aliasing	This mode attempts to improve the screen resolution by drawing lines and curves in a mixture of colors so that to the human eye the line or curve is smooth. The quality of the result is linked to the quality of the algorithm used by the workstation hardware.

4. 2 Creating a 3D display

To create 3D graphic objects and display them on the screen, follow the procedure below:

1. Create attributes.

2. Create a 3D viewer..

3. Create a view.

4. Create an interactive context.

5. Create interactive objects.

6. Create primitives in the interactive object

7. Display the interactive object.

4. 2. 1 Create attributes

Create colors.

Example

```
Quantity_Color Black (Quantity_NOC_BLACK);
Quantity_Color Blue (Quantity_NOC_MATRABLUE);
Quantity_Color Brown (Quantity_NOC_BROWN4);
Quantity_Color Firebrick (Quantity_NOC_FIREBRICK);
Quantity_Color Forest (Quantity_NOC_FORESTGREEN);
Quantity_Color Gray (Quantity_NOC_GRAY70);
Quantity_Color
      MyColor (0.99, 0.65, 0.31, Quantity_TOC_RGB);
Quantity_Color Beet (Quantity_NOC_BEET);
Quantity_Color White (Quantity_NOC_WHITE);
```

Create line attributes.

Example

```
Handle(Graphic3d_AspectLine3d) CTXLBROWN =
      new Graphic3d_AspectLine3d ();
Handle(Graphic3d_AspectLine3d) CTXLBLUE =
      new Graphic3d_AspectLine3d ();
Handle(Graphic3d_AspectLine3d) CTXLWHITE =
      new Graphic3d_AspectLine3d();
CTXLBROWN->SetColor (Brown);
CTXLBLUE->SetColor (Blue);
CTXLWHITE->SetColor (White);
```

Create marker attributes.

Example

```
Handle(Graphic3d_AspectMarker3d) CTXMFIREBRICK =
      new Graphic3d_AspectMarker3d();
CTXMFIREBRICK->SetColor (Firebrick);
CTXMFIREBRICK->SetScale (1.0);
CTXMFIREBRICK->SetType (Aspect_TOM_BALL);
```

Create facet attributes.

Example

```
Handle(Graphic3d_AspectFillArea3d) CTXF =
      new Graphic3d_AspectFillArea3d ();
Graphic3d_MaterialAspect BrassMaterial
(Graphic3d_NOM_BRASS);
Graphic3d_MaterialAspect GoldMaterial
(Graphic3d_NOM_GOLD);
CTXF->SetInteriorStyle (Aspect_IS_SOLID);
CTXF->SetInteriorColor (MyColor);
CTXF->SetDistinguishOn ();
CTXF->SetFrontMaterial (GoldMaterial);
```

```
CTXF->SetBackMaterial (BrassMaterial);
CTXF->SetEdgeOn ();
```

Create text attributes.

Example

```
Handle(Graphic3d_AspectText3d) CTXT =
      new Graphic3d_AspectText3d
      (Forest, Graphic3d_NOF_ASCII_MONO, 1., 0.);
```

4. 2. 2 Create a 3D Viewer (a Windows example)

Example

```
Handle(Graphic3d_WNTGraphicDevice) TheGraphicDevice = ...;
TCollection_ExtendedString aName(“3DV”);
myViewer =
      new V3d_Viewer (TheGraphicDevice,aName.ToExtString
      (), “”);
myViewer -> SetDefaultLights ();
myViewer -> SetLightOn ();
```

4. 2. 3 Create a 3D view (a Windows example)

It is assumed that a valid Windows window may already be accessed via the method GetSafeHwnd().

Example

```
Handle (WNT_Window) aWNTWindow;
aWNTWindow = new WNT_Window (TheGraphicDevice,
GetSafeHwnd());
myView = myViewer -> CreateView();
myView -> SetWindow (a WNTWindow);
```

4. 2. 4 Create an interactive context

Example

```
myAISContext = new AIS_InteractiveContext (myViewer);
```

You are now able to display interactive objects such as an AIS_Shape.

Example

```
TopoDS_Shape aShape = BRepAPI_MakeBox(10,20,30)_Solid();
Handle (AIS_Shape) aAISShape = new AIS_Shape(aShape);
myAISContext -> Display (aAISShape);
```

4. 2. 5 Create your own interactive object

Follow the procedure below to compute the presentable object:

1. Build a presentable object inheriting from AIS_InteractiveObject (refer to the Chapter on Presentable Objects).

2. Reuse the Prs3d_Presentation provided as an argument of the compute methods.

NOTE

There are two compute methods: one for a 'standard display', and the other for a 'degenerated display', i.e. in hidden line removal and wireframe modes.

Example of the compute methods

```
Void
myPresentableObject::Compute
      (const Handle(PrsMgr_PresentationManager3d)&
            aPresentationManager,
      const Handle(Prs3d_Presentation)& aPrs,
      const Standard_Integer aMode)
(
//...
)
```

```
void
myPresentableObject::Compute
            (const Handle(Prs3d_Projector)&,
      const Handle(Prs3d_Presentation)& aPrs)
(
//...
)
```

4. 2. 6 Create primitives in the interactive object

Get the group used in Prs3d_Presentation.

Example

```
Handle(Graphic3d_Group) TheGroup =
Prs3d_Root::CurrentGroup(aPrs);
```

Update the group attributes.

Example

```
TheGroup -> SetPrimitiveAspect(CTXLBLUE);
```

Create two triangles in group TheGroup.

Example

```
Graphic3d_Array1OfVertexN Tpts4TMeshN(0, 5);
Tpts4TMeshN (0).SetCoord (-34.0, 7.0, -20.0);
Tpts4TMeshN (1).SetCoord (-30.0, 19.0, -20.0);
Tpts4TMeshN (2).SetCoord (-26.0, 7.0, -20.0);
Tpts4TMeshN (3).SetCoord (-22.0, 19.0, -20.0);
Tpts4TMeshN (4).SetCoord (-18.0, 7.0, -20.0);
```

```
Tpts4TMeshN (5).SetCoord (-14.0, 19.0, -20.0);
Tpts4TMeshN (0).SetNormal ( 0.0, 0.0, 1.0);
Tpts4TMeshN (1).SetNormal ( 0.0, 0.0, 1.0);
Tpts4TMeshN (2).SetNormal ( 0.0, 0.0, 1.0);
Tpts4TMeshN (3).SetNormal ( 0.0, 0.0, 1.0);
Tpts4TMeshN (4).SetNormal ( 0.0, 0.0, 1.0);
Tpts4TMeshN (5).SetNormal ( 0.0, 0.0, 1.0);
Graphic3d_Array1OfVertex Tpts4TMesh(0, 5);
Tpts4TMesh (0).SetCoord (-24.0, 7.0, -10.0);
Tpts4TMesh (1).SetCoord (-20.0, 19.0, -10.0);
Tpts4TMesh (2).SetCoord (-26.0, 7.0, -10.0);
Tpts4TMesh (3).SetCoord (-22.0, 19.0, -10.0);
Tpts4TMesh (4).SetCoord (-18.0, 7.0, -10.0);
Tpts4TMesh (5).SetCoord (-14.0, 19.0, -10.0);
TheGroup->BeginPrimitives ();
      TheGroup->TriangleMesh (Tpts4TMesh N);
      TheGroup->TriangleMesh (Tpts4TMesh);
TheGroup->EndPrimitives ();
```

The BeginPrimitives () and EndPrimitives () methods are used when creating a set of various primitives in the same group.

Use the polyline function to create a boundary box for the Struct structure in group TheGroup.

Example

```
Standard_Real Xm, Ym, Zm, XM, YM, ZM;
Struct->MinMaxValues (Xm, Ym, Zm, XM, YM, ZM);

Graphic3d_Array1OfVertex Tpts6 (0, 15);
Tpts6(0).SetCoord (Xm, Ym, Zm);
Tpts6(1).SetCoord (Xm, Ym, ZM);
Tpts6(2).SetCoord (Xm, YM, ZM);
Tpts6(3).SetCoord (Xm, YM, Zm);
Tpts6(4).SetCoord (Xm, Ym, Zm);
Tpts6(5).SetCoord (XM, Ym, Zm);
Tpts6(6).SetCoord (XM, Ym, ZM);
Tpts6(7).SetCoord (XM, YM, ZM);
Tpts6(8).SetCoord (XM, YM, Zm);
Tpts6(9).SetCoord (XM, Ym, Zm);
Tpts6(10).SetCoord (XM, YM, Zm);
Tpts6(11).SetCoord (Xm, YM, Zm);
Tpts6(12).SetCoord (Xm, YM, ZM);
Tpts6(13).SetCoord (XM, YM, ZM);
```

```
Tpts6(14).SetCoord (XM, Ym, ZM);
Tpts6(15).SetCoord (Xm, Ym, ZM);
TheGroup->Polyline (Tpts6);
```

Create text and markers in group TheGroup.

Example

```
  static char *texte[3] = {  ”Application title”,
            ”My company”,
            ”My company address.” };
  Graphic3d_Array1OfVertex Tpts8 (0, 1);
  Tpts8(0).SetCoord (-40.0, -40.0, -40.0);
  Tpts8(1).SetCoord (40.0, 40.0, 40.0);
  TheGroup->MarkerSet (Tpts8);
  Graphic3d_Vertex Marker (0.0, 0.0, 0.0);

  for (i=0; i<=2; i++) {
        Marker.SetCoord (-(Standard_Real)i*4 + 30,
              (Standard_Real)i*4,-(Standard_Real)i*4);
        TheGroup->Text (texte[i], Marker,
              (Standard_Real)(i+2)/100.); }
```

5. 3D Resources

The 3D resources include the Graphic3d and V3d packages.

5. 1 Graphic3D

5. 1. 1 Overview

The **Graphic3d** package is used to create 3D graphic objects in a 3D viewer. These objects called **structures** are made up of groups of primitives and attributes. A group is the smallest editable element of a structure. A transformation can be applied to a structure. Structures can be connected to form a tree of structures, composed by transformations. Structures are globally manipulated by the viewer.

5. 1. 2 Provided services

Graphic structures can be:

- Displayed,
- Highlighted,
- Erased,
- Transformed,
- Connected to form a tree.

There are classes for:

- Visual attributes for lines, faces, markers, text, materials,
- Vectors and vertices,
- Defining an Advanced Graphic Device,
- Graphic objects, groups, and structures.

5. 1. 3 About the primitives

Markers

- Have one or more vertices,
- Have a type, a scale factor, and a color,
- Have a size, shape, and orientation independent of transformations.

Polygons

- Have one closed boundary,
- Have at least three vertices,
- Are planar and have a normal,

- Have interior attributes - style, color, front and back material, texture and reflection ratio,
- Have a boundary with the following attributes - type, width scale factor, color. The boundary is only drawn when the interior style is hollow.

Polygons with holes

- Have multiple closed boundaries, each one with at least three vertices,
- Are planar and have a normal,
- Have interior attributes - style, color, front and back material,
- Have a boundary with the following attributes - type, width scale factor, color. The boundary is only drawn when the interior style is hollow.

Polylines

- Have two or more vertices,
- Have the following attributes - type, width scale factor, color.

Text

- Has geometric and non-geometric attributes,
- Geometric attributes - character height, character up vector, text path, horizontal and vertical alignment,
- Non-geometric attributes - text font, character spacing, character expansion factor, color.

5. 1. 4 About materials

A **material** is defined by coefficients of:

- Transparency,
- Diffuse reflection,
- Ambient reflection,
- Specular reflection.

Two properties define a given material:

- Transparency
- Reflection properties - its absorption and reflection of light.

Diffuse reflection is seen as a component of the color of the object.

Specular reflection is seen as a component of the color of the light source.

The following items are required to determine the three colors of reflection:

- Color,
- Coefficient of diffuse reflection,
- Coefficient of ambient reflection,

- Coefficient of specular reflection.

5. 1. 5 About textures

A **texture** is defined by a name.

Three types of texture are available:

- 1D,
- 2D,
- Environment mapping.

5. 1. 6 Display priorities

Structure display priorities control the order in which structures are drawn. When you display a structure you specify its priority. The lower the value, the lower the display priority. When the display is regenerated the structures with the lowest priority are drawn first. For structures with the same display priority the order in which they were displayed determines the drawing order. CAS.CADE supports eleven structure display priorities.

5. 1. 7 About structure hierarchies

The root is the top of a structure hierarchy or structure network. The attributes of a parent structure are passed to its descendants. The attributes of the descendant structures do not affect the parent. Recursive structure networks are not supported.

5. 2 V3d

5. 2. 1 Overview

The **V3d** package provides the resources to define a 3D viewer and the views attached to this viewer (orthographic, perspective). This package provides the commands to manipulate the graphic scene of any 3D object visualized in a view on screen.

A set of high-level commands allows the separate manipulation of parameters and the result of a projection (Rotations, Zoom, Panning, etc.) as well as the visualization attributes (Mode, Lighting, Clipping, Depth-cueing, etc) in any particular view.

5. 2. 2 Provided services

The V3d package is basically a set of tools directed by commands from the viewer front-end. This tool set contains methods for creating and editing classes of the viewer such as:

- Default parameters of the viewer,
- Views (orthographic, perspective),
- Lighting (positional, directional, ambient, spot, headlight),

- Clipping planes (note that only Z-clipping planes can work with the Phigs interface),
- Instantiated sequences of views, planes, light sources, graphic structures, and picks,
- Various package methods.

5. 2. 3 A programming example

Example

This sample TEST program for the V3d Package uses primary packages Xw and Graphic3d and secondary packages Visual3d, Aspect, Quantity, Phigs, math.

//Create a Graphic Device from the default DISPLAY

```
Handle(Graphic3d_GraphicDevice) GD =
            new Graphic3d_GraphicDevice(””) ;
```

// Create a Viewer to this Device

```
Handle(V3d_Viewer) VM = new V3d_Viewer(GD, 400.,
    // Space size
    V3d_Xpos,// Default projection Quantity_NOC_DARKVIOLET,
    // Default background
    V3d_ZBUFFER,
    // Type of visualization
    V3d_GOURAUD,
    // Shading model
    V3d_WAIT);
    // Update mode
```

// Create a structure in this Viewer

```
Handle(Graphic3d_Structure) S =
      new Graphic3d_Structure(VM->Viewer()) ;
```

// Type of structure

```
S->SetVisual (Graphic3d_TOS_SHADING);
```

// Create a group of primitives in this structure

```
Handle(Graphic3d_Group) G = new Graphic3d_Group(S) ;
```

// Fill this group with one polygon of size 100

```
Graphic3d_Array1OfVertex Points(0,3) ;
Points(0).SetCoord(-100./2.,-100./2.,-100./2.) ;
Points(1).SetCoord(-100./2., 100./2.,-100./2.) ;
Points(2).SetCoord( 100./2., 100./2.,-100./2.) ;
Points(3).SetCoord( 100./2.,-100./2.,-100./2.) ;
Normal.SetCoord(0.,0.,1.) ;
G->Polygon(Points,Normal) ;
```

// Create Ambient and Infinite Lights in this Viewer
```
Handle(V3d_AmbientLight) L1 = new V3d_AmbientLight
      (VM,Quantity_NOC_GRAY50) ;
Handle(V3d_DirectionalLight) L2 = new V3d_DirectionalLight
      (VM,V3d_XnegYnegZneg,Quantity_NOC_WHITE) ;
```

// Create a 3D quality Window from the same GraphicDevice
```
Handle(Xw_Window) W =
      new Xw_Window(GD,”Test V3d”,0.5,0.5,0.5,0.5) ;
```

// Map this Window to this screen
```
 W->Map() ;
```

// Create a Perspective View in this Viewer
```
Handle(V3d_PerspectiveView) V =
      new V3d_PerspectiveView(VM);
```

// Set the Eye position
```
V->SetEye(100.,100.,100.) ;
```

// Associate this View with the Window
```
V->SetWindow(W) ;
```

// Activate ALL defined Lights in this View
```
V->SetLightOn() ;
```

// Display ALL structures in this View
```
(VM->Viewer())->Display() ;
```

// Finally update the Visualization in this View
```
V->Update() ;
```

5. 2. 4 Glossary of view transformations

The following terms are used to define view orientation, i.e. transformation from World Coordinates (WC) to the View Reference Coordinates system (VRC)

View Reference Point (VRP)	Defines the origin of View Reference Coordinates.
View Reference Plane Normal (VPN)	Defines the normal of projection plane of the view.
View Reference Up Vector (VUP)	Defines the vertical of observer of the view.

The following terms are used to define view mapping, i.e. transformation from View Reference Coordinates (VRC) to the Normalized Projection Coordinates (NPC)

Projection type	Orthographic or perspective.
Projection Reference Point (PRP)	Defines the observer position.
Front Plane Distance (FPD)	Defines the position of the front clipping plane in View Reference Coordinates system.
Back Plane Distance (BPD)	Defines the position of the back clipping plane in View Reference Coordinates system.
View Plane Distance (VPD)	Defines the position of the view projection plane in View Reference Coordinates system. View plane must be located between front and back clipping planes.
Window Limits	Defines the visible part of the view projection plane (left, right, top and bottom boundaries: Umin, Umax, Vmax and Vmin respectively) in View Reference Coordinates.

The V3d_View API uses the following terms to define view orientation and mapping

At	Position of View Reference Point (VRP) in World Coordinates
Eye	Position of the observer (projection reference point) in World Coordinates. Influences to the view projection vector and depth value.
Proj	View projection vector (VPN)
Up	Position of the high point / view up vector (VUP)
Depth	Distance between Eye and At point
ZSize	Distance between front and back clipping planes
Size	Window size in View Reference Coordinates
Focal Reference point	Position of Projection Reference Point (PRP) in World Coordinates
Focale	Distance between Projection Reference Point (PRP) and View projection plane

5. 2. 5 Management of perspective projection

The perspective projection allows definition of viewing volume as a truncated pyramid (frustum) with apex at the Projection Reference Point. In the View Reference Coordinate system it can be presented by the following picture:

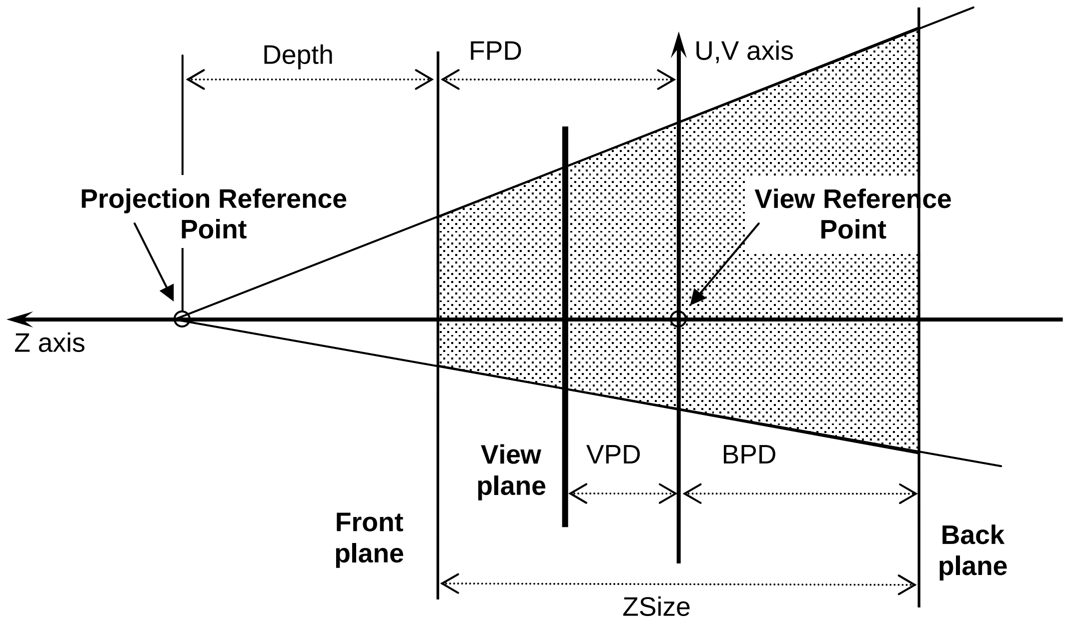


Figure 1 View Reference Coordinate System, perspective viewing volume and view mapping parameters

During panning, window limits are changed, as if a sort of “frame” through which the user sees a portion of the view plane was moved over the view. The perspective frustum itself remains unchanged.

The perspective projection is defined by two parameters:

- **Depth** value defines distance between Projection Reference Point and the nearest (front) clipping plane.
- **ZSize** defines distance between Front and Back clipping planes. The influence of this parameter is caused by the OCCT specific to center viewing volume around View Reference Point so the front and back plane distances were the same: FPD = BPD = ZSize / 2.

Note that the closer the displayed object to the Projection Reference Point the more visible its perspective distortion. Thus, in order to get a good perspective it is recommended to set ZSize value comparable with the expected model size and small Depth value.

However, very small Depth values might lead to inaccuracy of “fit all” operation and to non-realistic perspective distortion.

Example

```
// Create a Perspective View in Viewer VM
Handle(V3d_PerspectiveView) V =
      new V3d_PerspectiveView(VM);

// Set the ZSize
V->SetZSize(2000.) ;

// Set the Depth value
V->SetDepth(20.) ;

// Set the current mapping as default
// to be used by Reset() operation
V->SetViewMappingDefault() ;
```

As an alternative to manual setting of perspective parameters the *V3d_View::DepthFitAll* function can be used.

Example

```
// Display  shape in Viewer VM
Handle(AIS_InteractiveContext) aContext =
      new AIS_InteractiveContext(VM);
aContext->Display(shape);

// Create a Perspective View in Viewer VM
Handle(V3d_PerspectiveView) V =
      new V3d_PerspectiveView(VM);

// Set automatically the perspective parameters
V->DepthFitAll() ;

// Fit view to object size
V->FitAll();

// Set the current mapping as default
// to be used by Reset() operation
V->SetViewMappingDefault() ;
```

It is necessary to take into account that during rotation Z size of the view might be modified automatically to fit the model into the viewing volume.

Make sure the Eye point never gets between the Front and Back clipping planes.

In perspective view, changing Z size results in changed perspective effect. To avoid this, an application should specify the maximum expected Z size using V3d_View::SetZSize() method in advance.

V3d_View::FitAll() with FitZ = Standard_True and V3d_View::ZFitAll() also change the perspective effect and should therefore be used with precautions similar to those for rotation.

6. 2D Display

6. 1 Glossary of 2D terms

Attributes	The qualities attributed to an entity. For instance, primitives have attributes. A line primitive has a type e.g. continuous, dotted, etc. and a width e.g. thick, thin, etc. A text primitive has a font e.g. Helvetica. All primitives have a color attribute.
Attribute maps	So as to be available to an application, the attributes used by the program are stored in maps. A map is an array of (index, attribute) pairs.
Driver	A driver is the destination for the drawing of a view. The driver is associated with either a window or a plotter file as the ultimate destination of the view.
Graphic device	By default, the connection to the current workstation monitor screen and color map reservation.
Graphic object	Manages a set of primitives. A graphic object can be edited, adding or removing primitives. By default a graphic object is empty, drawable, plottable, pickable, and is neither displayed nor highlighted.
Primitive	A primitive is a drawable element. It has a definition in the space model. Primitives can either be lines, text, or images. Once displayed images and text remain the same size. Lines, with the exception of markers, can be modified e.g. zoomed. Primitives must be stored in a graphic object.
Space model	The overall 2D space under consideration. It has a point of origin and a size.
Update	This is the process of exploring the display list of the view to find each visible graphic object, which lies within the view mapping. Then for every such graphic object the Update process calls the Draw method of each graphic object. Then each graphic object calls the Draw method of each primitive it contains.
View	A view is a set of graphic objects arranged in a display list. The view offers the methods for pick and draw.
View mapping	A square region of the space model defined from its center and size (in meters). The purpose of the view mapping is to select the primitives to be displayed.
Window	A window provided by the window manager e.g. an X window.
Workspace	The workspace of a window driver is the size of its

window. For a plotter driver it is the size of a sheet of plot paper.

6. 2 Creating a 2D display

To create 2D graphic objects and display them on the screen, follow the procedure below:

1. Create the marker map.

2. Create the attribute maps.

3. Define the connection to a graphic device.

4. Create a window.

5. Create a window driver.

6. Install the maps.

7. Create a view.

8. Create a view mapping.

9. Create one or more graphic objects associated with a view.

10. Create primitives and associate them with a graphic object.

11. Get the workspace of the driver.

12. Update the view in the driver.

6. 2. 1 Creating the marker map

The marker map defines a set of markers available to the application. Markers may be predefined, Aspect_Tom_X for example, or user-defined.

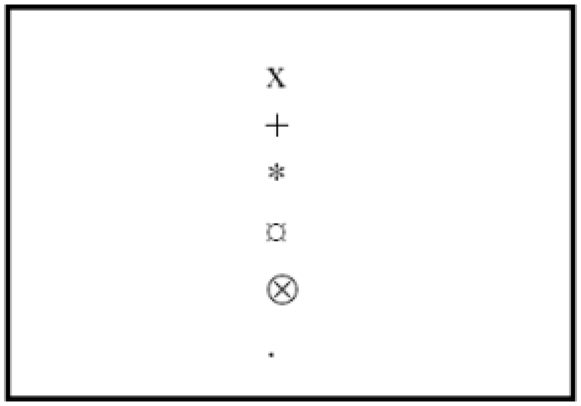


Figure 15. Markers.

The markers are manipulated by an index.

A marker map is defined as follows:

Example

```
Handle(Aspect_MarkMap) mkrmap = new Aspect_MarkMap;
Aspect_MarkMapEntry mkrmapentry1 (1, Aspect_TOM_X)
Aspect_MarkMapEntry mkrmapentry2 (2, Aspect_TOM_PLUS)
Aspect_MarkMapEntry mkrmapentry3 (3, Aspect_O_PLUS)

mkrmap->AddEntry (mkrmapentry1);
mkrmap->AddEntry (mkrmapentry2);
mkrmap->AddEntry (mkrmapentry3);
```

6. 2. 2 Creating the attribute maps

Maps are created for color, line type, line width, and text font. A map is used to reference a given attribute by an integer value.

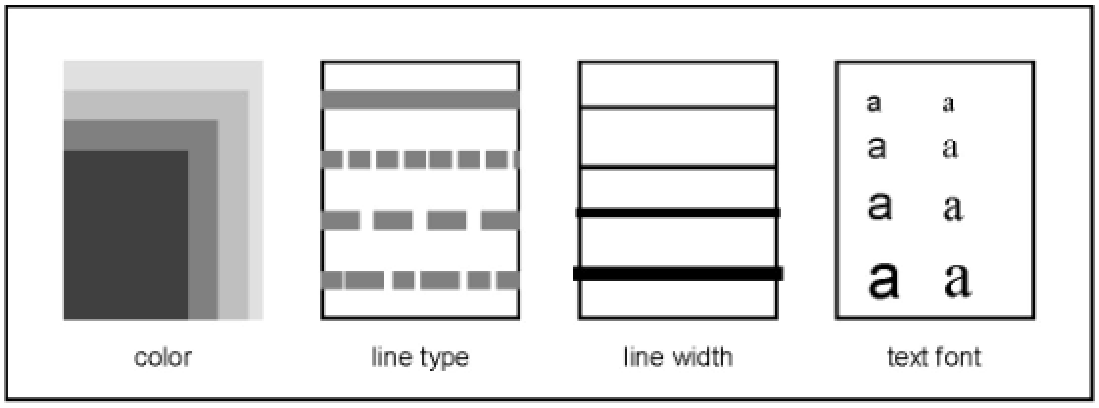


Figure 16. Attributes

The color map

The hardware system will certainly have default colors available but to make the application portable and durable, it must be insulated from external factors by defining the set of colors to be used.

A color map is defined as follows:

Example

```
Handle(Aspect_GenericColorMap) colmap =
                              new Aspect_GenericColorMap;
      Aspect_ColorMapEntry colmapentry;
      Quantity_Color YELLOW (Quantity_NOC_YELLOW);
      colmapentry.SetValue (1, YELLOW);
      colmap->AddEntry (colmapentry);
      Quantity_Color RED (Quantity_NOC_RED);
      colmapentry.SetValue (2, RED);
      colmap->AddEntry (colmapentry);
      Quantity_Color GREEN (Quantity_NOC_GREEN);
      colmapentry.SetValue (3, GREEN);
      colmap->AddEntry (colmapentry);
```

You can include as many colors in your color map as you like, though there are some restrictions related to the hardware.

The type map

Lines can be solid, dotted, dashed, dot-dashed, or user defined. For a user-defined type the pattern of solid and blank sections is listed.

A type map is defined as follows:

Example

```
Handle(Aspect_TypeMap) typmap = new Aspect_TypeMap;
      {TColQuantity_Array1OfLength myLineStyle(1,2);
      myLineStyle.SetValue(1, 2); // the solid part is 2 mm
      myLineStyle.SetValue(2, 3); // the blank part is 3 mm
      Aspect_LineStyle linestyle1 (Aspect_TOL_SOLID);
      Aspect_LineStyle linestyle2 (Aspect_TOL_DASH);
      Aspect_LineStyle linestyle3 (myLineStyle);
      Aspect_LineStyle linestyle4 (Aspect_TOL_DOTDASH);
      Aspect_TypeMapEntry typmapentry1 (1, linestyle1);
      Aspect_TypeMapEntry typmapentry2 (2, linestyle2);
      Aspect_TypeMapEntry typmapentry3 (3, linestyle3);
      Aspect_TypeMapEntry typmapentry4 (4, linestyle4);
      typmap->AddEntry (typmapentry1);
      typmap->AddEntry (typmapentry2);
      typmap->AddEntry (typmapentry3);
      typmap->AddEntry (typmapentry4);
```

NOTE

The line type enumeration and all the other enumerations are available from the Aspect package.

The width map

The width map defines a set of levels of line thickness available to your application. Widths and all other distances are specified in mms or as members of an enumeration.

A width map is defined as follows:

Example

```
Handle(Aspect_WidthMap) widmap = new Aspect_WidthMap;
Aspect_WidthMapEntry widmapentry1 (1,Aspect_WOL_THIN);
Aspect_WidthMapEntry widmapentry2 (2,Aspect_WOL_MEDIUM);
Aspect_WidthMapEntry widmapentry3 (3, 3);
Aspect_WidthMapEntry widmapentry4 (4, 40);
widmap->AddEntry (widmapentry1);
widmap->AddEntry (widmapentry2);
widmap->AddEntry (widmapentry3);
widmap->AddEntry (widmapentry4);
```

The font map

The font map defines a set of text fonts available to your application. Default fonts enumerated in Aspect may be used with addition of any other font known to the X driver, specifying the size and slant angle desired.

A font map is defined as follows:

Example

```
Handle(Aspect_FontMap) fntmap = new Aspect_FontMap;
Aspect_FontStyle fontstyle1 (”Courier-Bold”, 3, 0.0);
Aspect_FontStyle fontstyle2 (”Helvetica-Bold”, 3, 0.0);
Aspect_FontStyle fontstyle3 (Aspect_TOF_DEFAULT);
Aspect_FontMapEntry fntmapentry1 (1, fontstyle1);
Aspect_FontMapEntry fntmapentry2 (2, fontstyle2);
Aspect_FontMapEntry fntmapentry3 (3, fontstyle3);
fntmap->AddEntry (fntmapentry1);
fntmap->AddEntry (fntmapentry2);
fntmap->AddEntry (fntmapentry3);
```

6. 2. 3 Creating a 2D driver (a Windows example)

Example

```
Handle(WNT_GraphicDevice) TheGraphicDevice = ...;
TCollection_ExtendedString aName(“2DV”);
my2DViewer = new V2d_Viewer(TheGraphicDevice,
            aName.ToExtString());
```

6. 2. 4 Installing the maps

When the 2D viewer has been created, you may install the maps created earlier.

Example

```
my2DViewer->SetColorMap(colormap);
my2DViewer->SetTypeMap(typmap);
my2DViewer->SetWidthMap(widthmap);
my2DViewer->SetFontMap(fntmap);
```

6. 2. 5 Creating a view (a Windows example)

It is assumed that a valid Windows window may be accessed via the method GetSafeHwnd().

Example

```
Handle(WNT_Window) aWNTWindow;
aWNTWindow = new
      WNT_Window(TheGraphicDevice, GetSafeHwnd());
aWNTWindow->SetBackground(Quantity_NOC_MATRAGRAY);
Handle(WNT_WDriver) aDriver = new
WNT_WDriver(aWNT_Window);
myV2dView = new V2d_View(aDriver, my2dViewer, 0,0,50);
// 0,0: view center and 50: view size
```

6. 2. 6 Creating the presentable object

Follow the procedure below to compute the presentable object.

1. Build a presentable object inheriting from AIS_InteractiveObject (refer to Chapter 1 Fundamental Concepts, Section Presentable objects)

2. Re-use the graphic object provided as an argument of the Compute method for your presentable object.

Example

```
void
myPresentableObject::Compute (
      const Handle(Prs_Mgr_PresentationManager2D)&
            aPresentationManager,
      const Handle(Graphic2d_GraphicObject)& aGrObj,
      const Standard_Integer aMode)
{
...
}
```

6. 2. 7 Creating a primitive

Primitives may be created using the resources of the Graphic2d package. Here for example an array is instantiated and filled with a set of three circles with different radii, line widths, and colors, centered on given origin coordinates (4.0, 1.0) and passed to the specified graphic object (go).

Example

```
Handle(Graphic2d_Circle) tcircle[4];
Quantity_Length radius;
for (i=1; i<=4; i++) {
radius = Quantity_Length (i);
tcircle[i-1] = new Graphic2d_Circle (aGrObj, 4.0, 1.0,
radius);
tcircle[i-1]->SetColorIndex (i);
tcircle[i-1]->SetWidthIndex (1); }
```

Add a filled rectangle to your graphic object. It will be put outside of your view mapping.

Example

```
TColStd_Array1OfReal aListX (1, 5);
TColStd_Array1OfReal aListY (1, 5);
aListX (1) = -7.0; aListY (1) = -1.0;
aListX (2) = -7.0; aListY (2) = 1.0;
aListX (3) = -5.0; aListY (3) = 1.0;
aListX (4) = -5.0; aListY (4) = -1.0;
aListX (5) = -7.0; aListY (5) = -1.0;
Handle(Graphic2d_Polyline) rectangle =
      new Graphic2d_Polyline (go, 0., 0., aListX, aListY);
rectangle->SetColorIndex (6);
rectangle->SetWidthIndex (1);
rectangle->SetTypeOfPolygonFilling(Graphic2d_TOPF_FILLED);
rectangle->SetDrawEdge(Standard_True);
```

NOTE

A given primitive can only be assigned to a single graphic object.

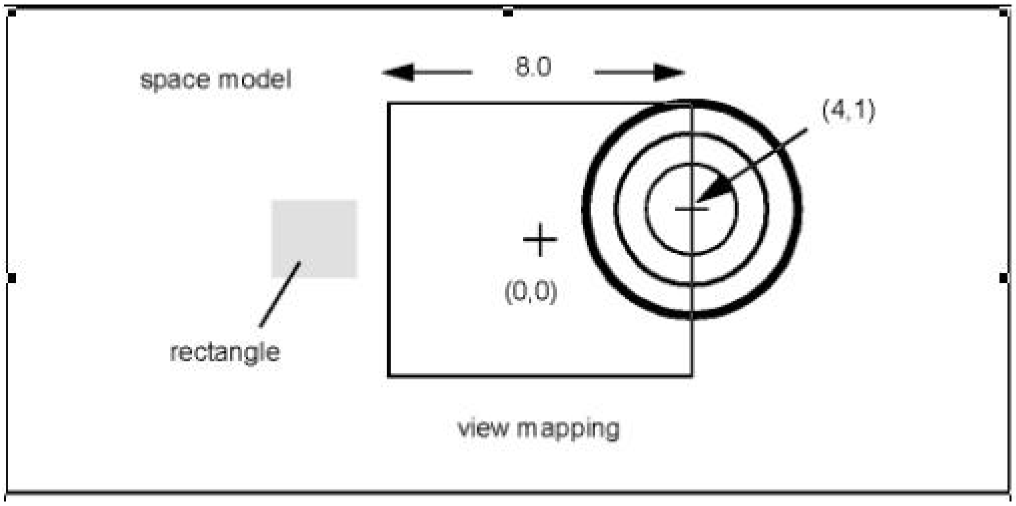


Figure 17. Graphic object and view mapping in the space model.

6. 3 Dealing with images

6. 3. 1 General case

Images are primitives too. The graphic resources can currently accept all image types described in the *AlienImage* package. In the following example only **.xwd** formats are accepted.

Define the primitive Image in the GraphicObject.

Example

```
Handle(Image_Image) anImage;
if (XwdImage || RgbImage) {
anImage = AlienUser->ToImage ();
Handle(Graphic2d_Image) gImage = new Graphic2d_Image
      (aGrObj, anImage, 0., 0., 0., 0., Aspect_CP_CENTER);
}
```

NOTE

The above constructor for image takes as arguments the graphic object which will contain the image, the image itself, XY coordinates for the center, XY offsets in the device space, and a cardinal point value to give a direction of display.

Now update the view in the driver. In other words, draw the image.

Example

```
Standard_Boolean clear = Standard_True
view->Update (driver, viewmapping, W/2., H/2., scale,
clear);
```

6. 3. 2 Specific case: xwd format

When the manipulated image is stored with the xwd format, a special class Graphic2d_ImageFile may be used to increase performance.

Example

```
OSD_Path aPath (“C:\test.xwd”);
OSD_File aFile (aPath);
Handle(Graphic2d_ImageFile)gImageFile =
            new Graphic2d_ImageFile (aGrObj,
                              aFile,
                              0.,0.,
                        0.,0.,
                              Aspect_CP_Center, 1);
gImageFile->SetZoomable(Standard_True);
```

The graphic contains now an image, which is manipulated as a primitive.

6. 4 Dealing with text

The constructor for the Graphic2d_Text takes a reference point in the space model and an angle (in radians) as its arguments, as well as the graphic object to which it is assigned. Note that the angle is ignored unless the Xdps driver, which allows angled text, is in use.

Example

```
TCollection_ExtendedString str1 (”yellow Courier-bold”);
TCollection_ExtendedString str2 (”red Helevetica-bold”);
TCollection_ExtendedString str3 (”green
Aspect_TOF_DEFAULT”);
Handle(Graphic2d_Text) t1 = new Graphic2d_Text
      (aGrObj, str1, 0.3, 0.3, 0.0);
Handle(Graphic2d_Text) t2 = new Graphic2d_Text
      (aGrObj, str2, 0.0, 0.0, 0.0);
Handle(Graphic2d_Text) t3 = new Graphic2d_Text
      (aGrObj, str3, -0.3, -0.3, 0.0);
t1->SetFontIndex (1); t1->SetColorIndex (1);
t2->SetFontIndex (2); t2->SetColorIndex (2);
t3->SetFontIndex (3); t3->SetColorIndex (3);
```

6. 5 Dealing with markers

A marker is a primitive that retains its original size when the view is zoomed. Markers can be used, for example, as references to dimensions.

6. 5. 1 Vectorial markers

Every marker takes an XY point as its reference point. The constructor also takes another pair of XY values as an offset from this reference point. For CircleMarker and EllipsMarker this offset point is its center. For PolylineMarker this offset point is its origin i.e. the first point in its list.

In the example below, a rectangle is created using Graphic2d_Polyline.

Example

```
TColStd_Array1OfReal rListX (1, 5);
TColStd_Array1OfReal rListY (1, 5);
rListX (1) = -0.3; rListY (1) = -0.3;
rListX (2) = -0.3; rListY (2) = 0.3;
rListX (3) = 0.3; rListY (3) = 0.3;
rListX (4) = 0.3; rListY (4) = -0.3;
rListX (5) = -0.3; rListY (5) = -0.3;
```

```
Handle(Graphic2d_Polyline) rp =
      new Graphic2d_Polyline (aGrObj, rListX, rListY);
```

Two Graphic2d_CircleMarkers are created. The first one has no offset from its center. The second is constrained to be a given offset from a reference point.

Example

```
Handle(Graphic2d_CircleMarker) rc1 = new
Graphic2d_CircleMarker
      (aGrObj, 0.04, 0.03, 0.0, 0.0, 0.01);
Handle(Graphic2d_CircleMarker) rc2 = new
Graphic2d_CircleMarker
      (aGrObj, 0.03, -0.03, 0.01, 0.0, 0.01);
window->Clear ();
```

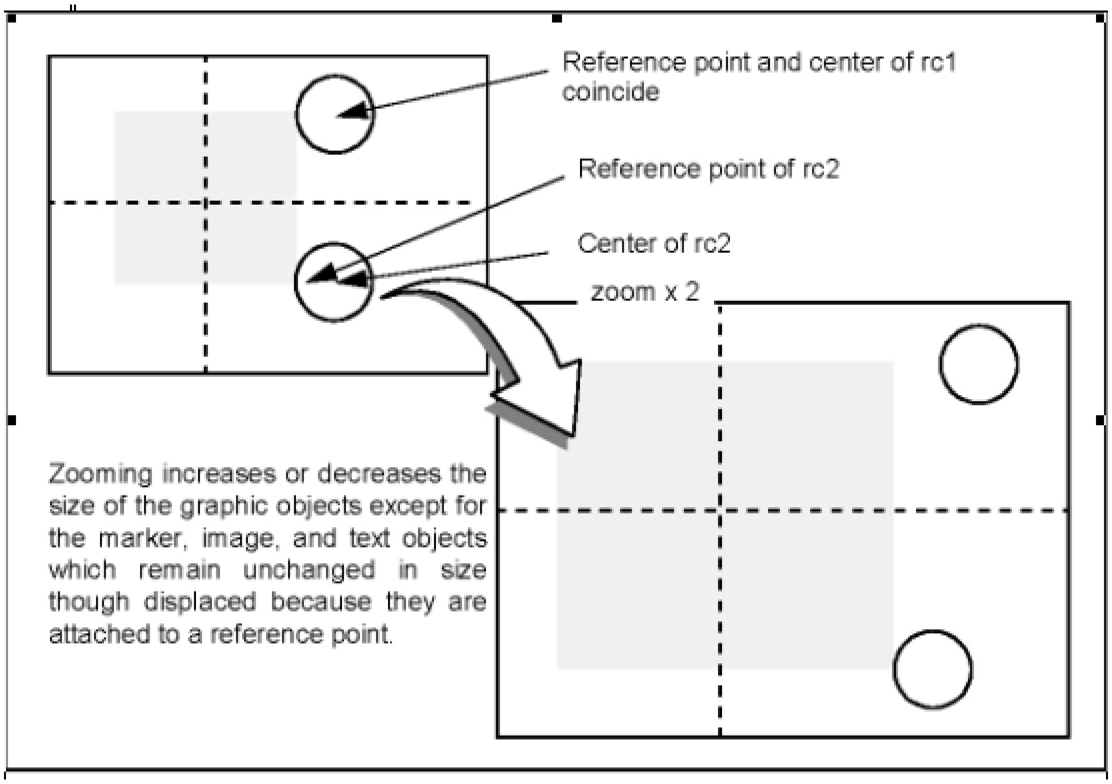

Reference point and center of rc1 coincide
Reference point of rc2
Center of rc2
zoom x 2
Zooming increases or decreases the size of the graphic objects except for the marker, image, and text objects which remain unchanged in size though displaced because they are attached to a reference point.

Figure 18. Figure of zoom and attachment point of a marker.

6. 5. 2 Indexed markers

Once the marker map has been created, indexed markers may be added to a graphic object.

Example

```
Handle (Graphic2d_Marker) xmkr = new Graphic2d_Marker
      (aGrObj, 1, 0.04, 0.03, 0.0, 0.0, 0.0);
Handle (Graphic2d_Marker) plusmkr = new Graphic2d_Marker
      (aGrObj, 2, 0.04, 0.0, 0.0, 0.0, 0.0);
Handle (Graphic2d_Marker) oplusmkr = new Graphic2d_Marker
      (aGrObj, 3, 0.04, -0.03, 0.0, 0.0, 0.0);
```

6. 6 Dragging with Buffers

A **buffer** is used to draw very quickly a partial area of the scene without deleting the background context.

A buffer contains a set of graphic objects or primitives which are to be moved, rotated or scaled above the scene in the front planes of the view (in this case, double-buffering is not active). For example:

1. Draw a very complex scene in the view.
2. Create a buffer of primitives with the primitive color index 10 and the font index 4:

```
buffer = new Graphic2d_Buffer (view, 0., 0., 10, 4);
```

3. Add graphic objects or primitives:

```
buffer->Add (go);
buffer->Add (tcircle[1]);
buffer->Add (t1);
```

4. Post the buffer in the view:

```
buffer->Post ();
```

5. Move, rotate or scale the buffer above the view:

```
buffer->Move (x,y); buffer->Rotate (alpha);
buffer->Scale (zoom_factor);
```

6. Unpost the buffer from the view:

```
buffer->Unpost ();
```

7. 2D Resources

The 2D resources include the Graphic2d, Image, AlienImage, and V2d packages.

7. 1 Graphic2d

7. 1. 1 Overview

The **Graphic2d** package is used to create a 2D graphic object. Each object, called a GraphicObject, is composed of primitives. Each primitive is a class and contains attributes. Each primitive has its own Draw method.

A Graphic2d_Image is created from an Image from the Image package.

7. 1. 2 The services provided

The **Graphic2d** packages provides classes for creating the following primitives:

- Circle
- Curve
- Ellips
- InfiniteLine
- Polyline
- Segment
- SetOfSegments
- Text
- Marker
- SetOfMarkers
- VectorialMarker
- CircleMarker

2D Resources

- PolylineMarker
- EllipsMarker
- Image
- ImageFile
- SetOfCurves

7. 2 Image

7. 2. 1 Overview

The **Image** package provides the resources to produce and manage bitmap images. It has two purposes:

- To define what is an image on the CAS.CADE platform.
- To define operations which can be carried out on an image.

The package allows the user to manipulate images without knowing their type. For various functionalities such as zoom, pan, and rotation, an application does not need to know the type nor the format of the image. Consequently, the image could be stored as an integer, real, or object of the Color type.

Another important asset of the package is to make the handling of images independent of the type of pixel. Thus a new image based on a different pixel type can be created without rewriting any of the algorithms.

7. 2. 2 The services provided

The classes **ColorImage** and **PseudoColorImage** define the two types of image, which can be handled by the Image toolkit.

ColorImage is used to create 24-bit TrueColor images:

- Create a ColorImage object with a given background color.
- Request the type of the image.
- Request or set the color of a given pixel.
- Zoom, rotating, translating, simple and refining transformations.
- Set position and size.
- Transpose, shift, clip, shift, clear.
- Draw line and rectangle.

PseudoColorImage is used to create 32-bit images:

- Create a PseudoColorImage object with a given background color associated with a given ColorMap (Generic, ColorCube, ColorRamp)
- Ask or set the color of a given pixel, row, or column.
- Find the maximum & minimum pixel values of an image.
- Change the pixel values by scaling.
- Change the pixel values below a threshold value.
- Zoom, rotating, translating, simple and refining transformations.
- Set position and size.

- Transpose, shift, clip, shift, clear.
- • Draw line and rectangle.

Convertor is used to:

- Change an image from a ColorImage to a PseudoColorImage. Select between two dithering algorithms for the change.
- Change an image from a PseudoColorImage to a ColorImage.
- Change a PseudoColorImage into one with a different ColorMap.

LookupTable is used to:

- Transform the pixels of a PseudoColorImage.

Various **PixelInterpolation** classes are available for dealing with pixel values at non-integer coordinates.

The package also includes a number of **package methods** for zooming, rotation, translation, as well as simple and refining transformations.

7. 3 AlienImage

7. 3. 1 Overview

The **AlienImage** package is used to import 2D images from some other format into the CAS.CADE format.

7. 3. 2 Available Services

- Reads the content of an AlienImage object from a file.
- Writes the content of an AlienImage object to a file.
- Converts an AlienImage object to an Image object.
- Converts an Image object to an AlienImage object.

7. 4 V2d

7. 4. 1 Overview

This package is used to build a 2D mono-view viewer in a windowing system. It contains the commands available within the viewer (zoom, pan, pick, etc).

7. 4. 2 The services provided

The **V2d** package contains the **View** class. **View** is used to:

- Create a view in an window.

- Handle the view:
  - zoom
  - fit all
  - pan
  - translate
  - erase
  - pick
  - highlight
  - set drawing precision
  - Postscript output

8. Graphic Attributes

8. 1 Aspect

8. 1. 1 Overview

The **Aspect** package provides classes for the graphic elements, which are common to all 2D and 3D viewers - screen background, windows, edges, groups of graphic attributes that can be used in describing 2D and 3D objects.

8. 1. 2 The services provided

The **Aspect** package provides classes to implement:

- Color maps,
- Pixels,
- Groups of graphic attributes,
- Edges, lines, background,
- Font classes,
- Width map classes,
- Marker map classes,
- Type of Line map classes,
- Window,
- Driver, PlotterDriver (inherited by PS_Driver), WindowDriver,
- Graphic device (inherited by Xw_GraphicDevice, Graphic3d_GraphicDevice),
- Enumerations for many of the above,
- Array instantiations for edges,
- Array instantiations for map entries for color, type, font, width, and marker.