Learning upon OpenCascade

# Introduction

### History

* Early 1990s: Developed as CAS.CADE by Matra Datavision
* 1999: Open-sourced and renamed to OpenCASCADE
* 2000: Open Cascade SAS company established
* 2011: Community initiated Open Cascade Community Edition fork
* 2013: License changed to LGPL

### Current Operational Model

* Primarily maintained and developed by Open Cascade SAS
* Adopts an open collaborative development model
* Accepts community contributions via public Git repository and bug tracking system
* Releases 1-2 major versions annually

### Open Source License

* Current License: LGPL 2.1-only with additional exception (since December 18, 2013)
* LGPL Commercial Characteristics:
  + Allows linking the library to proprietary software
  + Modifications to the library must be released under the same license
  + Applications using the library don't need to be open-sourced

### Architecture

Open CASCADE Technology consists of C++ classes grouped into Packages. They are organized into Toolkits (libraries), and the latest are grouped into seven Modules.

A diagram of a structure

Description automatically generated

### Features

**Geometric modeling:**

* Calculate the intersection of two 2D curves, surfaces, or a 3D curve and a surface
* Project points onto 2D and 3D curves, points onto surfaces, and 3D curves onto surfaces
* Construct lines and circles from constraints
* Construct curves and surface from constraints
* Construct curves and surfaces by interpolation and approximation

**Surface and solid modeling:**

* Construction of primitives (box, sphere, cylinder, cone, torus, wedge)
* Computation and comparison of distances between shapes
* Construction of prisms and pipes
* Surface extrusion/revolution
* Defining offset surfaces/curves
* Defining fillets and chamfers
* Boolean operations (input, common, cut, fuse.)

A yellow and purple cubes

Description automatically generated

**Supported formats:**

**B-Rep**: STEP (ISO 10303), IGES, BREP (native)

**Mesh**: glTF, VRML (ISO 14772), OBJ Wavefront, STL

### Visualization(supported by OCCT visualization toolset)

* Cross-platform renderer based on OpenGL and OpenGL ES
* Compatibility with mobile, embedded (OpenGL ES) and web (WebGL) platforms
* Built-in photo-realistic Ray-Tracing engine supporting global illumination
* Clipping planes/box with capping
* PBR metallic-roughness material workflow
* Fast algorithms picking objects by mouse

# Install OCCT Test Environment

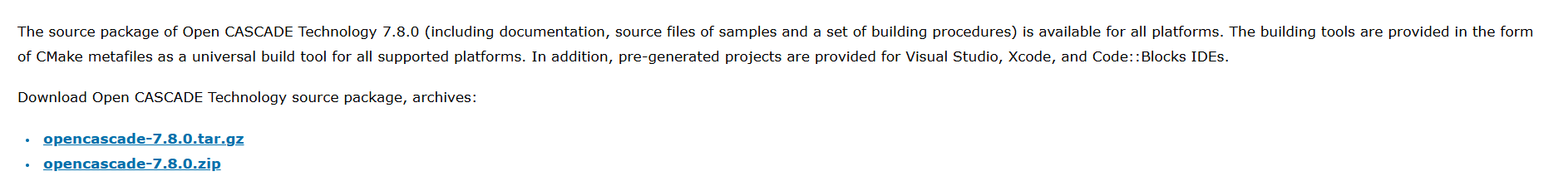
## Prerequests

|  |  |
| --- | --- |
| OCCT & OCCT third parts 7.8 | <https://dev.opencascade.org/release> |
| CMake 3.31.0+ | <https://cmake.org/download/> |
| QT 5.9 | <https://download.qt.io/archive/qt/> |

OCCT provides pre-compiled versions, but since these pre-compiled versions do not include OCCT Overview, it is necessary to choose to download the OCCT source code and third-party support libraries.

A close-up of a computer screen

Description automatically generated



## Build OCCT source

1. Create a build directory in the root of occt source, where will used to store builded files. A screenshot of a computer

   Description automatically generated
2. Open the CMake-GUI, in the first location, enter the path of the OCCT source code folder, and in the second location, enter the path of the build folder you just created. A screenshot of a computer

   Description automatically generated
3. Click 'Configure', select VS2019 configuration and x64, then click 'Finish‘A screenshot of a computer program

   Description automatically generated
4. In the '3RDPARTY\_DIR' line, enter the path to the third-party library files, which is the 'product' path. And the 3RDPARTY\_DIR is the path of 3rdparty-vc14-64. Check/tick the 'BUILD\_SAMPLES\_QT' option. And click the ‘Configure’ button. 
5. Modify the QT path. This Qt5.15.2 is the folder after installing QT. Please locate the corresponding files in it by yourself (if you cannot find them, it's probably because some options were not checked during the QT installation and download process). 
6. CMake configuration is complete. The final configuration results are shown below(No red region and no error). And click ‘Generate’ button.A white and grey striped background

   Description automatically generated with medium confidence A white rectangular object with a black and white stripe

   Description automatically generated with medium confidence

## Configure the project environment

1. The generated projects are shown below A screenshot of a computer

   Description automatically generated
2. Right-click on OCCTOverview and set it as the startup project. Then right-click on OCCTOverview again, navigate to Properties > Debugging > Environment, and paste the following content. This is for configuring the project's environment variables. Note: You need to modify the paths to match the paths on your own computer.

A screenshot of a computer program

Description automatically generated A screenshot of a computer

Description automatically generated

The Environment should configure like this:

A screenshot of a computer program

Description automatically generated

CASROOT=D:\ProgramFile\OCCTcode\OCCT-7\_8\_0

CSF\_FPE=0

CSF\_OCCTResourcePath=D:\ProgramFile\OCCTcode\OCCT-7\_8\_0\src

CSF\_OCCTDataPath=D:\ProgramFile\OCCTcode\OCCT-7\_8\_0\data

CSF\_OCCTSamplesPath=D:\ProgramFile\OCCTcode\OCCT-7\_8\_0\samples

CSF\_OCCTTestsPath=D:\ProgramFile\OCCTcode\OCCT-7\_8\_0\tests

QT\_DEBUG\_PLUGINS=1

PATH=%PATH%;D:\ProgramFile\Qt\5.15.2\msvc2015\_64\bin;D:\ProgramFile\Qt\5.15.2\msvc2015\_64\plugins\platforms;D:\ProgramFile\OCCTcode\products\freetype-2.5.5-vc14-64\bin;D:\ProgramFile\OCCTcode\products\freeimage-3.17.0-vc14-64\bin;

CASROOT needs to be configured to point to the OCCT source code folder.

CSF\_OCCT refers to the files located within the OCCT source code folder.

The last line configures paths that include QT's bin directory, QT's platforms directory, and third-party library files (freetype and freeimage)

1. Copy the 'platforms' folder from D:\ProgramFile\Qt\5.15.2\msvc2015\_64\plugins and paste it to D:\ProgramFile\OCCTcode\OCCT-7\_8\_0\build\win64\vc14\bind (in the generated build folder) path. If you want to run in release mode, paste it to D:\ProgramFile\OCCTcode\OCCT-7\_8\_0\build\win64\vc14\bin path.



1. Now, you can build and run the OCCTOverview A screenshot of a computer

   Description automatically generated

# Case Study: Create block

In this chapter, I will use examples to compare the creation of a solid box in both Parasolid and OCCT environments.

### Parasolid part

In the Parasolid environment, if you want to create a 10x10x10 solid block, you can use 'PK\_BODY\_create\_solid\_block' to achieve this. A computer screen with text

Description automatically generated

## OCC part

In OCC's 'BRepPrimAPI\_MakeBox.cxx' source code, four methods for creating a solid block are defined.

|  |  |
| --- | --- |
| Create Method | Source Code Definition |
| Create by three dimensions, and the start point is {0,0,0}; |  |
| Create by start point and three dimensions |  |
| Create by coordinate system and dimensions |  |
| Create by two points |  |

The describe of each class are as follows:

|  |  |
| --- | --- |
| Definition | Describe |
| Standard\_Real | Standard\_Real is equal to double type. |
| gp\_Pnt | The prefix 'gp' here means 'Geometric Primitives', and this class represents a point defined in a Cartesian coordinate system. |
| gp\_Ax2 | gp\_Ax2 is a 3D coordinate system defined by an origin point and two orthogonal unit direction vectors (X and Y axes), used to describe position and orientation in space. |
| TopoDS\_Shape | TopoDS\_Shape is the most fundamental topological geometric object class in OCC, serving as the abstract base class for all geometric shapes. |

Here, the solid block is created using a coordinate system and dimensions.

A screenshot of a computer

Description automatically generatedA computer code with text

Description automatically generated

In OCC, after creating a solid, you need to wrap the geometric object with AIS\_ColoredShape, set its color, and then add the object to the display list before it can be rendered. Additional steps are required to visualize it in the view, whereas Parasolid supports automatic display.

# Foundation Class

## Package

A package groups together a number of classes which have semantic links. For example, a geometry package would contain Point, Line, and Circle classes. A package can also contain enumerations, exceptions and package methods (functions). In practice, a class name is prefixed with the name of its package e.g. Geom\_Circle.

## Data Types

The data types in OCC is fall into two categories, data types manipulated by handle (or reference) and data types manipulated by value.

A diagram of value and value

Description automatically generated

A handle is OCC implementation of a smart pointer. Several handles can reference the same object. Also, a single handle may reference several objects, but only one at a time. To have access to the object it refers to, the handle must be de-referenced just as with a C++ pointer.

Objects of classes derived (directly or indirectly) from Transient, are normally allocated in dynamic memory using operator new, and manipulated by handle. Handle is defined as template class opencascade::handle<>. Open CASCADE Technology provides preprocessor macro Handle() that is historically used throughout OCCT code to name a handle:



In addition, for most OCCT classes additional *typedef* is defined for a handle, as the name of a class prefixed by *Handle\_*. For instance, the above example can be also coded as:



# Topology and Geometry

## The OCC Topol class hierarchy

OCC allows accessing and manipulating data of objects without dealing with their 2D or 3D representations. Whereas Geometry provides a description of objects in terms of coordinates or parametric values, Topology describes data structures of objects in parametric space. These descriptions use location in and restriction of parts of this space.

Abstract Topology is provided by six packages:

|  |  |
| --- | --- |
| Abstract Topology Packages | Describe |
| TopAbs | TopAbs Package Provides general resources for topology-driven applications. It contains enumerations that are used to describe basic topological notions: topological shape, orientation and state. |
| TopLoc | *TopLoc*package provides resources to handle 3D local coordinate systems: *Datum3D* and *Location*. *Datum3D* describes an elementary coordinate system, while *Location* comprises a series of elementary coordinate systems. |
| TopoDS | *TopoDS* package describes classes to model and build data structures that are purely topological. |
| TopTools | TopTools package provides basic tools to use on topological data structures. |
| TopExp | TopExp package provides classes to explore and manipulate the topological data structures described in the TopoDS package. |
| BRepTools | BRepTools package provides classes to explore, manipulate, read and write BRep data structures. These more complex data structures combine topological descriptions with additional geometric information, and include rules for evaluating equivalence of different possible representations of the same object, for example, a point. |

The design of the OpenCASCADE topology structure is based on the STEP standard ISO-10303-42. The structure is a directed graph, where the parent classes point to the child classes, without any reverse references. The abstract structure is implemented using the C++ classes in the TopoDS package. Below is an inheritance diagram of the TopoDS classes, taken from the [Doxygen-generated documentation](https://dev.opencascade.org/doc/refman/html/class_topo_d_s___shape.html).

A diagram of a diagram

Description automatically generated

TopoDS\_Shape is operated on by value, and contains three fields - location, orientation, and a myTShape handle (of type TopoDS\_TShape).

A diagram of a diagram

Description automatically generated

A diagram of a computer

Description automatically generated

The myTShape and Location are used to share data between various shapes, thereby saving a significant amount of memory. For example, if two connected faces have a common edge, that edge will have the same Location and myTShape. Compared to Parasolid, OpenCascade (OCC) does not have the topological concept of 'fin', but it can express the same meaning through the 'orientation' attribute.

For example, when an edge is the intersection of two faces, in Parasolid, these two faces will use this edge through their respective fins, with each fin recording how the edge is used in its corresponding face; in OCC, these two faces will directly use the same edge, but the edge will carry different orientation attributes in different faces, expressing how the edge is used in different faces through the forward/reverse of the orientation.

## Relationship between Topology and Geometry

In OCC, the connection between topology and geometry is implemented through the BRep package. The BRep package inherits the abstract topological classes from the TopoDS package, thereby acquiring the properties of the topological classes. At the same time, BRep also aggregates the attributes of the geometric classes, thus associating the topological objects with the geometric features.Moreover, only three types of topological objects have geometric features - they are Vertex, Edge, and Face.

A diagram of a diagram

Description automatically generated

## Primitive Geometric Object Type

In OCC, the representation of points is divided into 2D gp\_Pnt2d and 3D gp\_Pnt, and the representation of vectors is also divided into gp\_Vec2d and gp\_Vec. In OCC, most geometric elements, except for plane, cylinder, cone, sphere, and torus, are distinguished between 2D and 3D.

In Parasolid, a unified POINT\_t is used to represent points, and AXIS\_t is used to represent coordinate axes.

In the representation of curves, the two systems also have significant differences. OCC is characterized by a strict distinction between 2D and 3D, including basic curve types such as lines, circles, ellipses, parabolas, and hyperbolas, each with corresponding 2D and 3D versions.

In the representation of surfaces, the two systems have relatively smaller differences, both including plane, cylindrical, conical, spherical, and toroidal surfaces.

In terms of naming conventions, OCC uses the gp\_ prefix to indicate that the element belongs to the gp package, which represents the most basic geometric package. In Parasolid, the \_t suffix is used.

|  |  |  |
| --- | --- | --- |
| Geometric Object Type | OCC Type | Describe |
| Point | gp\_Pnt2d | 2D Point |
|  | gp\_Pnt | 3D Point |
| Vector | gp\_Vec2d | 2D Vector |
|  | gp\_Vec | 3D Vector |
| Curve | gp\_Lin2d | 2D Line |
|  | gp\_Lin | 3D Line |
| Circle | gp\_Circle2d | 2D Circle |
|  | gp\_Circle | 3D Circle |
| Plane | gp\_Pln | Plane |
| Cylinder | gp\_Cone | Cylinder |

## Geometric Transformation

These four code segments demonstrate the four basic transformation operations in OCC: translation, rotation, symmetry, and scaling.

**Translation**

In OCC, the gp\_Trsf class is used to represent translation transformation, which is achieved by setting the translation vector. The first code segment uses the gp\_Trsf class to directly set the translation vector to achieve the translation transformation. The second code segment uses the gp\_Vec to represent the translation vector, and applies the translation transformation through the Translated method of the gp\_Pnt.

A screenshot of a computer code

Description automatically generated

**Rotation**

The code defines a rotation axis gp\_Ax1 around the Z-axis, and specifies a 45-degree rotation angle.It then uses the SetRotation method of the gp\_Trsf class to set the rotation transformation.A screenshot of a computer code

Description automatically generated

**Symmetry**

The code defines a symmetry transformation gp\_Trsf centered at the origin.It applies the transformation to a point using the Transformed method of gp\_Pnt.A computer code with text

Description automatically generated

**Scaling**

The code defines a scaling transformation gp\_Trsf centered at the origin, and sets a scaling factor of 2.It applies the scaling transformation to a point using the Transformed method of gp\_Pnt.

A screenshot of a computer

Description automatically generated

Compared to OCC, Parasolid is relatively more generic and does not distinguish different types of transformations like OCC does. In the implementation of composite transformations, both OCC and Parasolid support achieving them through multiple calls to the transformation operations.

# Case Study: Enquiry & Calculations

In this case, we will query the topological structure contained in the block created in the previous case, using Parasolid and OCC respectively.

## Parasolid part

A screenshot of a computer code

Description automatically generated

## OCC part

A screenshot of a computer program

Description automatically generated

BRep\_Tool is a utility class, which provides a set of static functions to access and manipulate the underlying geometric data of BRep Shapes. The key functionslities provide by BRep\_Tool is pnt function, curve function, surface function, tolerance function and isclosed function.

|  |  |
| --- | --- |
| BRep\_Tool key function example | Describe |
|  | returns a gp\_Pnt object, which contains the 3D coordinates of the vertex. |
|  | Returns the reference about 3D curve of the edge E, its location L, and the parameter range [First, Last]. |
|  | Returns the reference about surface of the face F and its location L. |
|  | Returns the tolerance of the Vertex V. |
|  | Used to determine if an edge forms a closed loop. |

It can be seen that in terms of querying, Parasolid mainly implements through passing arrays, while OCC implements through creating iterators. Meanwhile, it's worth noting that when creating geometry through topology, geometric points are returned as class instances, while curves and surfaces are returned as corresponding pointers.

# Case Study: OCC Geometry related information

In this case, I created a B-spline by interpolation points array and queries basic parameters about parameter range, the number of control points and the number of knots.

A computer screen shot of a code

Description automatically generated

The curve in the provided code is a pointer of type **Geom\_BSplineCurve**. It represents a B-spline curve. If we need to search for the parameters about B-spline, we can run the function in the **Geom\_BSplineCurve**. Such as**, curve->NbKnots()** will return the number of knots, and **curve->Knot(i)** will return the knot of range index i.

A screenshot of a computer

Description automatically generated

From the above code, we can see that in OCC, if we want to query the parameters of a geometry, it is achieved by calling the query methods of that geometric class.

# Case Study: Example of tracking

 The example demonstrates how to create a solid block from primitives, add an edge blend, and then a cylindrical boss. The stages involved in the process are as follows:

* Create a set of curves that form the basis of a sheet profile.
* Create the sheet profile: make a wire body from the curves, make a face, and attach a surface.
* Sweep the profile to form a solid body.
* Blend an edge on the body.
* Add a cylindrical boss to the body.

In this Case, I will compare the Parasolid code and OpenCascade code step by step.

## Create curves to form a profile

In Parasolid, you need to create a series of curves that will be used to form a profile. Since you are creating a simple block, the profile consists of four lines. You can create these with four calls to PK\_LINE\_create, passing in a location (point) and an axis (direction) to each call via the line\_sf standard form to create four unbounded lines that pass through the specified points in the given directions.

A diagram of a diagram

Description automatically generated

In OpenCascade, you can Create four points and use them to create edges.

|  |  |
| --- | --- |
| Parasolid | OpenCascade |
|  |  |

BRepBuilderAPI\_MakeEdge creates an edge between two points.

## Create a Wire Profile

In Parasolid, you can parameterize the lines to create finite-length edges and then make a wire body.

A diagram of a red square with white text

Description automatically generated

A computer code on a white background

Description automatically generated

In OpenCascade you can combine edges into a wire by BRepBuilderAPI\_MakeWire function.

A screen shot of a computer code

Description automatically generated

## Create a Sheet Profile

In Parasolid, you have a wire body which, if used as a profile in a sweep operation, would create a sheet body. Since you are creating a solid block, you first need to turn this wire body into a sheet. You do this by first adding a face, and then attaching a surface to the face.

To add a face to the wire profile, use PK\_EDGE\_make\_faces\_from\_wire, which attaches faces to closed loops in a wire body. And you can pass this face into PK\_FACE\_attach\_surf\_fitting to create a suitable surface and attach it to the face, thereby completing the sheet profile you require.



In OpenCasCade, you need to create a face from wire by BRepBuilderAPI\_MakeFace.



## Sweep the Profile to Create a Solid

In Parasolid, Sweep the Sheet profile to form a block body.

A close up of a text

Description automatically generated

A diagram of a cube

Description automatically generated

In OpenCascade, BRepPrimAPI\_MakePrism can sweep the face to create solid.

A screen shot of a computer code

Description automatically generated

## Adding an Edge Blend

In this part, identified a vertical edge which is parallel to the Z-axis and add fillets to it. In OCC:

A screenshot of a computer program

Description automatically generated

TopExp\_Explorer is a topology explorer class that allows iteration through topological entities.

## Adding a Cylindrical

A screen shot of a computer

Description automatically generated

|  |  |
| --- | --- |
| A computer drawing of a rectangular object  Description automatically generated | A red object with a white object on it  Description automatically generated |

# Visualization

Visualization in OCC is based on the separation of:

1. on the one hand – the data which stores the geometry and topology of the entities you want to display and select, and
2. on the other hand – its **presentation** (what you see when an object is displayed in a scene) and **selection** (possibility to choose the whole object or its sub-parts interactively to apply application-defined operations to the selected entities).

In this part, I will focus on the presentation and selection.

## Presentation

In OCC, if you want to display an object on the screen involves three kinds of entities:

1. a presentable object, the AIS\_InteractiveObject
2. a Viewer
3. an interactive contet, the AIS\_InteractiveContext

A diagram of a diagram

Description automatically generated

### Presentable Object

The purpose of a presentable object is to provide the graphical representation of an object in the form of Graphic3d\_Structure. 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 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.

### Viewer

The viewer allows interactively manipulating 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 Graphic3d structures in your presentation algorithms allows you to use the 3D viewers provided in Open CASCADE Technology for 3D visualization.

### Interactive Context

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 displaying.

## Selection

Standard OCCT selection algorithm is represented by 2 parts: dynamic and static. Dynamic selection causes objects to be automatically highlighted as the mouse cursor moves over them. Static selection allows to pick particular object (or objects) for further processing.

### Sensitive entity

Sensitive entities in the same way as entity owners are links between objects and the selection mechanism.

The purpose of entities is to define what parts of the object will be selectable in particular. Thus, any object that is meant to be selectable must be split into sensitive entities (one or several). For instance, to apply face selection to an object it is necessary to explode it into faces and use them for creation of a sensitive entity set.

A diagram of a cylinder

Description automatically generated

### Selection Frustum

While the occ select an object, The first step of each run of selection is to build the selection frustum according to the currently activated selection type. For the point or the rectangular selection the base of the frustum is a rectangle built in conformity with the pixel tolerance or the dimensions of a user-defined area, respectively. For the polyline selection, the polygon defined by the constructed line is triangulated and each triangle is used as the base for its own frustum. Thus, this type of selection uses a set of triangular frustums for overlap detection. The frustum length is limited by near and far view volume planes and each plane is built parallel to the corresponding view volume plane.

A diagram of a cube

Description automatically generated

The image above shows the rectangular frustum: a) after mouse move or click, b) after applying the rectangular selection.

A white triangular object with black background

Description automatically generated

In the image above triangular frustum is set: a) by a user-defined polyline, b) by triangulation of the polygon based on the given polyline, c) by a triangular frustum based on one of the triangles.

### BVH Trees

To maintain selection mechanism at the viewer level, a speedup structure composed of 3 BVH trees is used. The first level tree is constructed of axis-aligned bounding boxes of each selectable object. Hence, the root of this tree contains the combination of all selectable boundaries even if they have no currently activated selections. Objects are added during the display of *AIS\_InteractiveObject* and will be removed from this tree only when the object is destroyed. The 1st level BVH tree is build on demand simultaneously with the first run of the selection algorithm.

The second level BVH tree consists of all sensitive entities of one selectable object. The 2nd level trees are built automatically when the default mode is activated and rebuilt whenever a new selection mode is calculated for the first time.

The third level BVH tree is used for complex sensitive entities that contain many elements: for example, triangulations, wires with many segments, point sets, etc. It is built on demand for sensitive entities with more than 800K sub-elements (defined by *StdSelect\_BRepSelectionTool::PreBuildBVH()*).

A screenshot of a diagram

Description automatically generated

### How to select

The selection algorithm in occ includes Pre-processing and three main stages.

In pre-processing stage, implies calculation of the selection frustum and its main characteristics.

After successful building of the selection frustum, the algorithm starts traversal of the object-level BVH tree. The nodes containing axis-aligned bounding boxes are tested for overlap with the selection frustum following the terms of separating axis theorem (SAT). When the traversal goes down to the leaf node, it means that a candidate object with possibly overlapping sensitive entities has been found. If no such objects have been detected, the algorithm stops and it is assumed that no object needs to be selected. Otherwise it passes to the next stage to process the entities of the found selectable object.

In the next stage, it is necessary to determine if there are candidates among all sensitive entities of one object.

First of all, at this stage the algorithm checks if there is any transformation applied for the current object. If it has its own location, then the correspondingly transformed frustum will be used for further calculations. At the next step the nodes of the second level BVH tree of the given object are visited to search for overlapping leaves. If no such leafs have been found, the algorithm returns to the second stage. Otherwise it starts processing the found entities by performing the following checks:

1. activation check - the entity may be inactive at the moment as it belongs to deactivated selection;
2. tolerance check - current selection frustum may be too large for further checks as it is always built with the maximum tolerance among all activated entities; thus, at this step the frustum may be scaled.

After these checks the algorithm passes to the last stage.

If the entity is atomic, a simple SAT test is performed. In case of a complex entity, the third level BVH tree is traversed. The quantitative characteristics (like depth, distance to the center of geometry) of matched sensitive entities is analyzed and clipping planes are applied (if they have been set). The result of detection is stored and the algorithm returns to the second stage.

# How to Render

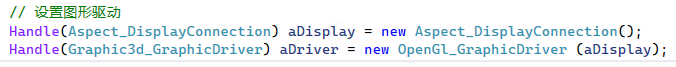
In this section, I will figure a example to build a minimal viewer to render a block. The general workflow of the program is as follow:

A diagram of a software process

Description automatically generated

## Set up the View

Firstly, I have to initialize Graphic Driver. All low-level details are hidden by Graphic3d\_GraphicDriver abstraction layer, so that application just needs choosing driver implementation:



OpenGl\_GraphicDriver from TKOpenGl library is the main implementation of this interface in OCCT. As can be deduced from it’s name, this driver relies on OpenGL – the mature graphic API available on almost every modern platform. A dedicated chapter at the end of the articles elaborate more details of various graphic APIs.

After you setup the Graphic Driver, you need to create a window implementing Aspect\_Window interface. A screen shot of a computer

Description automatically generated

OCC defines two basic classes for the viewer management – V3d\_Viewer and V3d\_View. A close-up of a computer code

Description automatically generated

Next, I set up a view background. And V3d\_Viewer is associated with a graphic driver, while V3d\_View is associated with a single window.

AIS\_InteractiveObject is a root interface defining an abstract interactive object in OCCT 3D Viewer. This class inherits two base classes at once – PrsMgr\_PresentableObject and SelectMgr\_SelectableObject, indicating that an Interactive Object can be displayed (defines presentation) and picked (defines selection sensitivities).

A screenshot of a computer code

Description automatically generated

If you’ll try running the code, at this point you may find objects drawn completely black! This is because our viewer setup misses light sources. The simplest way to solve the issue is calling V3d\_Viewer::SetDefaultLights(), which would add default ambient and directional light sources into the scene:



## Handling Window Message

In order to interact with the application (e.g., handling mouse events), we need to process window messages.

Since the Win32 API's window message handler needs to be static, we use a wrapper to call MyViewer's member functions.

A computer code with text

Description automatically generated

And the windowProc will deal process windows message:

A computer screen shot of a computer code

Description automatically generated

Then by calling the functions of AIS\_ViewController, it can handle mouse movement events and mouse wheel events to achieve view operations such as panning, rotating, zooming in and out.A computer screen shot of a computer program

Description automatically generated

A computer code with many text

Description automatically generated with medium confidence

The final result/effect is as follows:

A screenshot of a computer

Description automatically generated

# Appendix