Qt day

ELETTRONICA
MIND IS THE FIRST DEFENCE

Effective Fault Detection and Diagnosis with Qt 3D

QtDay 2018

Agenda

- What is BIT?
- Goals
- Why Qt 3D?
- The Asset Conditioning Pipeline
- Elt Bit 3D: an API on top of Qt 3D
  - Scenegraph
  - Framegraph
  - Picking
- Demo

What is BIT?

Built-In-Test (**BIT**) is a key component of safety- and mission-critical systems.

It provides a level of confidence in the integrity of each module at both power-up and during normal operation.

Three types of BIT:

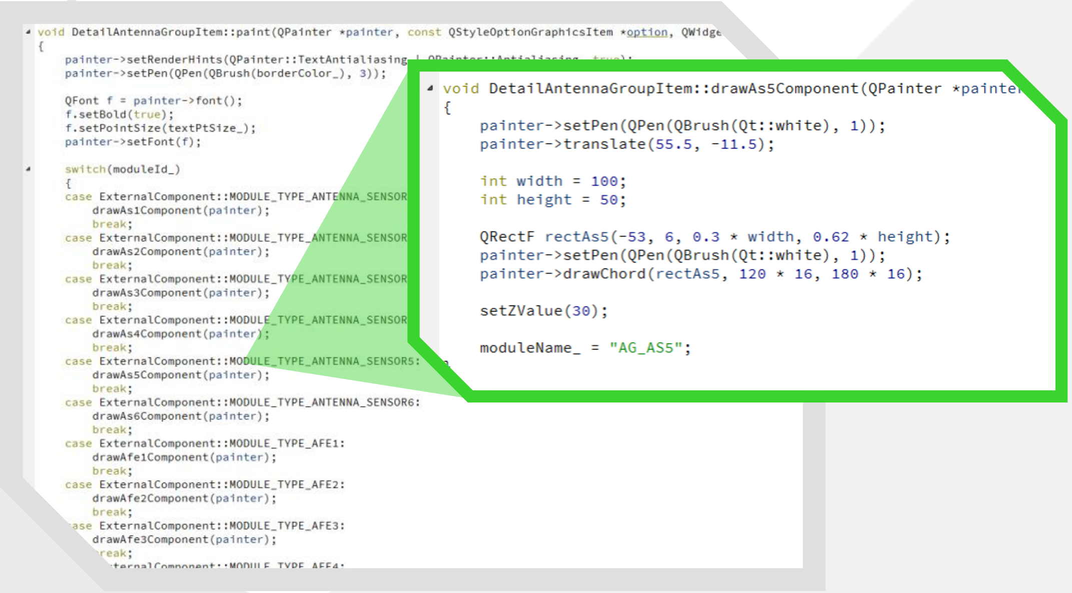
- Power-on BIT (PBIT)
- Continuous BIT (CBIT)
- Interruptive BIT (IBIT)

Traditional 2D UIs

2D Painting

Drawbacks:

- The activities of image elaboration and low level painting are not cost-effective
- 2D representation offers a static view (from a single point of view)



```
void DetailAntennaGroupItem::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidge
{
    painter->setRenderHints(QPainter::TextAntialiasing | QPainter::Antialiasing, true);
    painter->setPen(QPen(QBrush(borderColor_), 3));

    QFont f = painter->font();
    f.setBold(true);
    f.setPointSize(textPtSize_);
    painter->setFont(f);

    switch(moduleId_)
    {
    case ExternalComponent::MODULE_TYPE_ANTENNA_SENSOR
        drawAs1Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_ANTENNA_SENSOR
        drawAs2Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_ANTENNA_SENSOR
        drawAs3Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_ANTENNA_SENSOR
        drawAs4Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_ANTENNA_SENSOR5:
        drawAs5Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_ANTENNA_SENSOR6:
        drawAs6Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_AFE1:
        drawAfe1Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_AFE2:
        drawAfe2Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_AFE3:
        drawAfe3Component(painter);
        break;
    case ExternalComponent::MODULE_TYPE_AFE4:
```

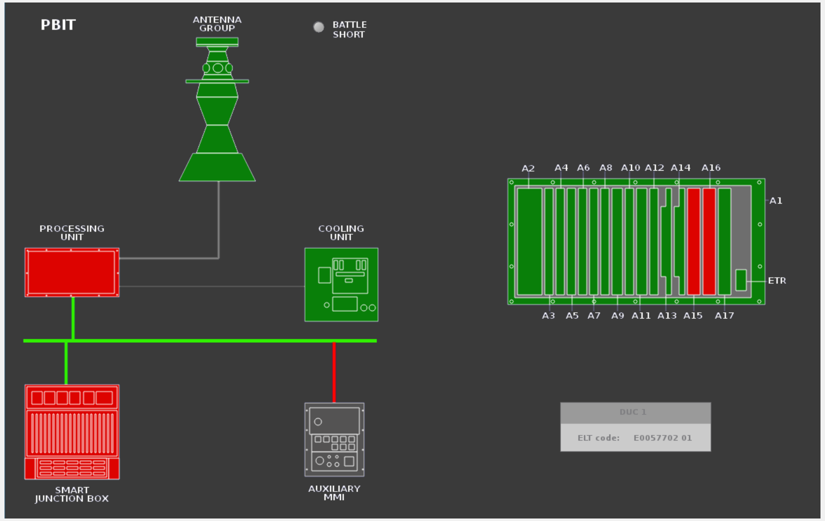
```
void DetailAntennaGroupItem::drawAs5Component(QPainter *painte
{
    painter->setPen(QPen(QBrush(Qt::white), 1));
    painter->translate(55.5, -11.5);

    int width = 100;
    int height = 50;

    QRectF rectAs5(-53, 6, 0.3 * width, 0.62 * height);
    painter->setPen(QPen(QBrush(Qt::white), 1));
    painter->drawChord(rectAs5, 120 * 16, 180 * 16);

    setZValue(30);

    moduleName_ = "AG_AS5";
```



UNCLASSIFIED

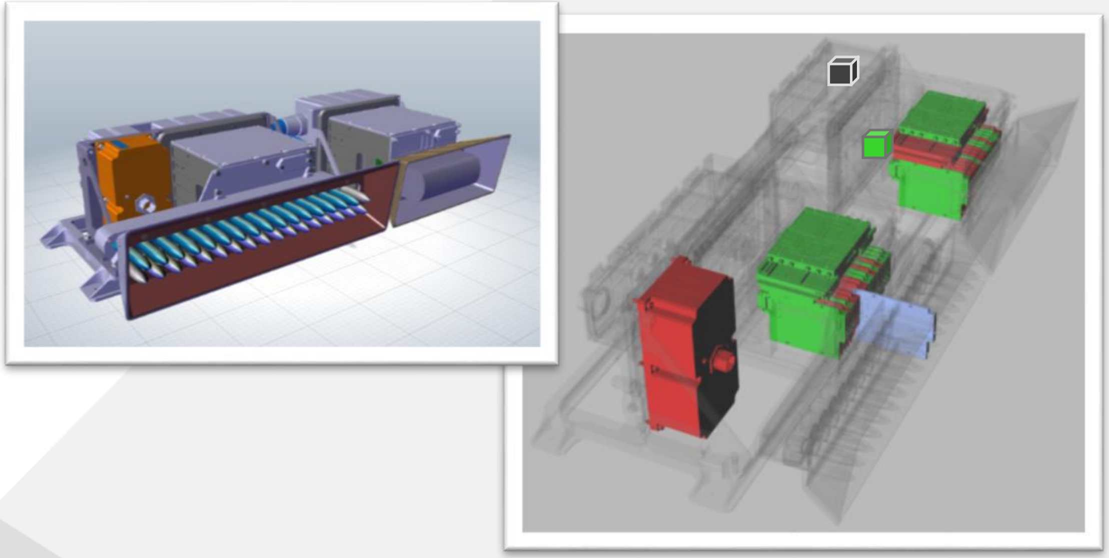
New Approach

3D Loading

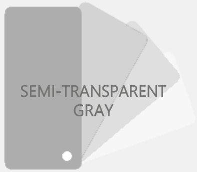
2D Painting

Goals:

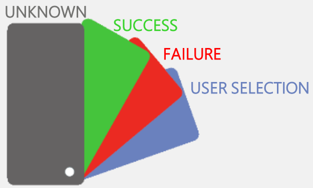
- To design a UI that allows to identify system faults effectively
    - fast component tree traversal
    - realistic and dynamic component representation
- To leverage 3D models provided by mechanical engineers
- To decouple the views from specific system configuration



Unreferenced Entities



Referenced Entities



Why Qt 3D?

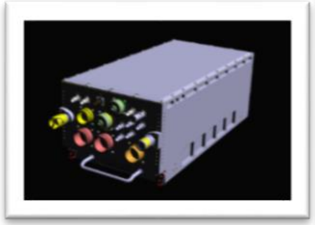
- our application is Qt-based (no further dependencies from 3rd party libraries)
- enables developers to quickly implement any rendering pipeline
- offers support for glTF (ideal for resource sensitive applications)

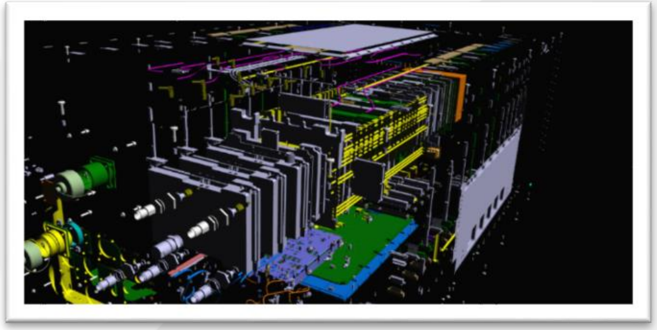
3D Models

3D Loading

2D Painting

Detailed **3D models** are heavy





=

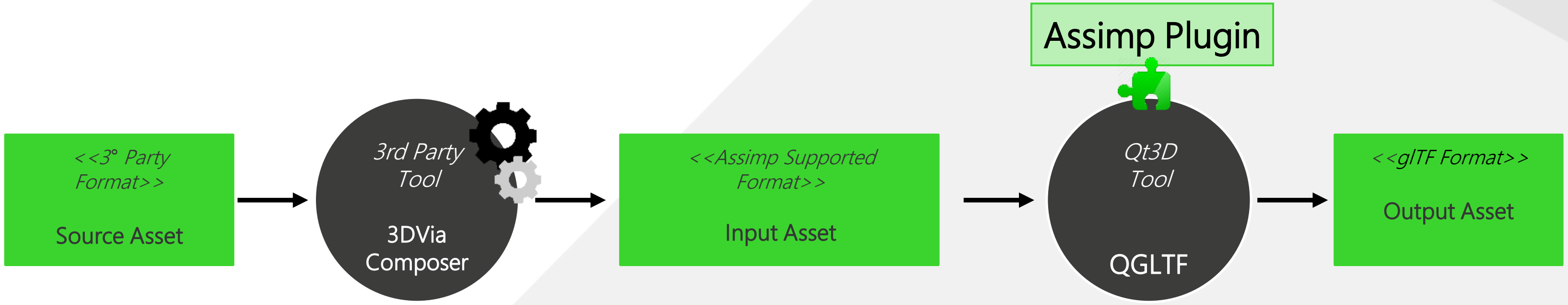
CATIA V5 File		
Size	Triangles	Entities
272 MB	1.624.040	3603

The Asset Conditioning Pipeline

3D Loading

QGLTF Optimization

2D Painting



Assimp Plugin

<<3° Party Format>>

Source Asset

3rd Party Tool

3DVia Composer

<<Assimp Supported Format>>

Input Asset

Qt3D Tool

QGLTF

<<glTF Format>>

Output Asset

CATIA V5 File
Size
272 MB

OBJ File	
Size	Loading Time
416 MB	100 s

3DS File	
Size	Loading Time
44 MB	34 s

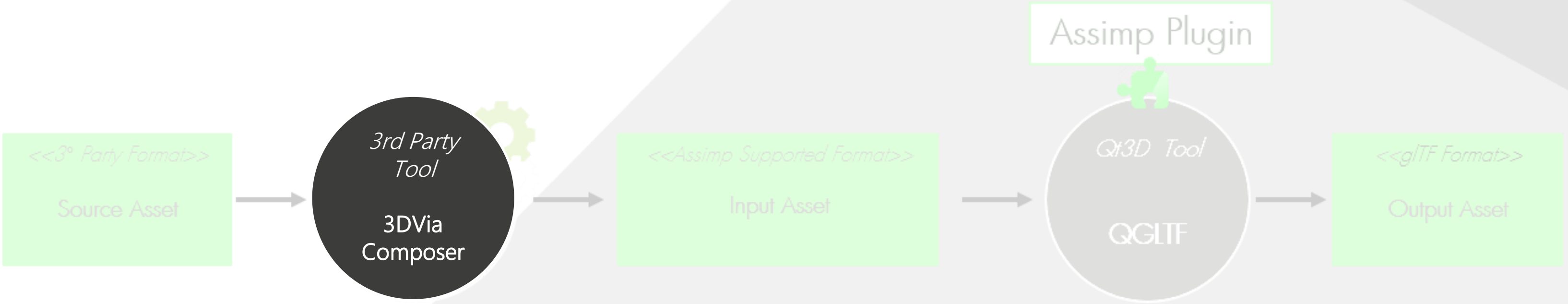
QGLTF File	
Size	Loading Time
52 MB	8 s

The Asset Conditioning Pipeline

3D Loading

Simplification

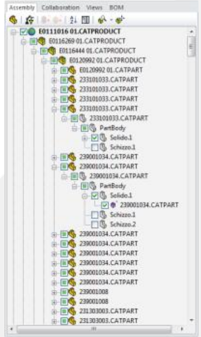
2D Painting



Triangles	Entities
1.624.040	3603

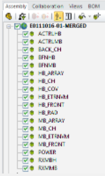
→

Triangles	Entities
464.204	30



```
Assembly  Collaboration  Views  BOM
E0111016 01.CATPRODUCT
  E0116269 01.CATPRODUCT
    E0116444 01.CATPRODUCT
      E0120992 01.CATPRODUCT
        E0120992 01.CATPART
        233101033.CATPART
        233101033.CATPART
        233101033.CATPART
        233101033.CATPART
          233101033.CATPART
            PartBody
              Solido.1
              Schizzo.1
        239001034.CATPART
        239001034.CATPART
          239001034.CATPART
            PartBody
              Solido.1
                239001034.CATPART
              Schizzo.1
              Schizzo.2
        239001034.CATPART
        239001034.CATPART
        239001034.CATPART
        239001034.CATPART
        239001034.CATPART
        239001034.CATPART
        239001008
        239001008
        231303003.CATPART
        231303003.CATPART
```

→



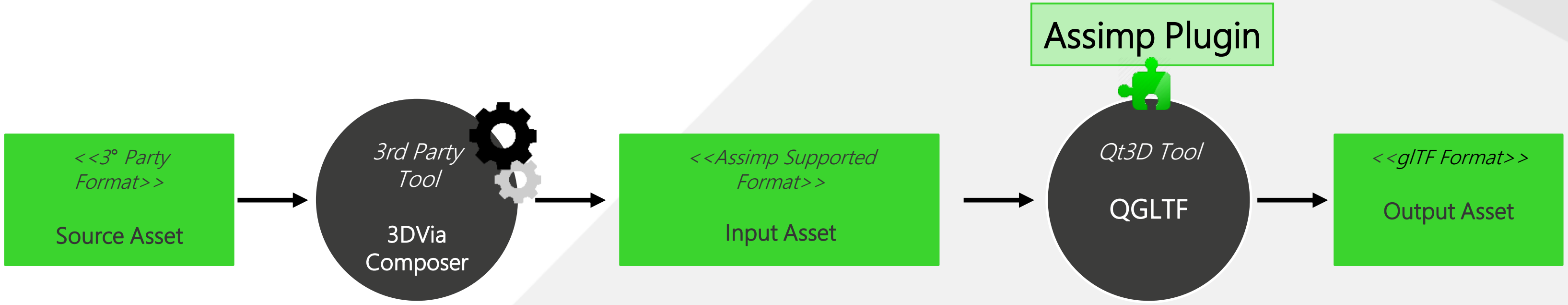
```
Assembly  Collaboration  Views  BOM
E0111016-01-MERGED
  ACTRLHB
  ACTRLMB
  BACK_CH
  BFNHB
  BFNMB
  HB_ARRAY
  HB_CH
  HB_COV
  HB_ETRNMM
  HB_FRONT
  HB_RAD
  MB_ARRAY
  MB_CH
  MB_ETRNMM
  MB_FRONT
  POWER
  RXMBH
  RXMMB
```

The Asset Conditioning Pipeline

3D Loading

2D Painting

Simplification + QGLTF Optimization



Assimp Plugin

<<3° Party Format>>
Source Asset

3rd Party Tool
3DVia Composer

<<Assimp Supported Format>>
Input Asset

Qt3D Tool
QGLTF

<<glTF Format>>
Output Asset

CATIA V5 File
Size
272 MB

OBJ File	
Size	Loading Time
223 MB	51 s

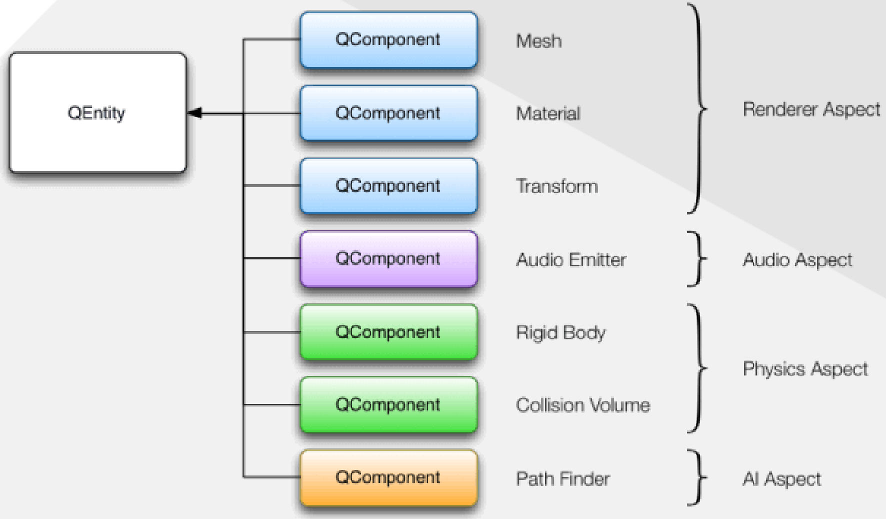
3DS File	
Size	Loading Time
11 MB	9 s

QGLTF File	
Size	Loading Time
18 MB	< 1 s

Qt3D Overview

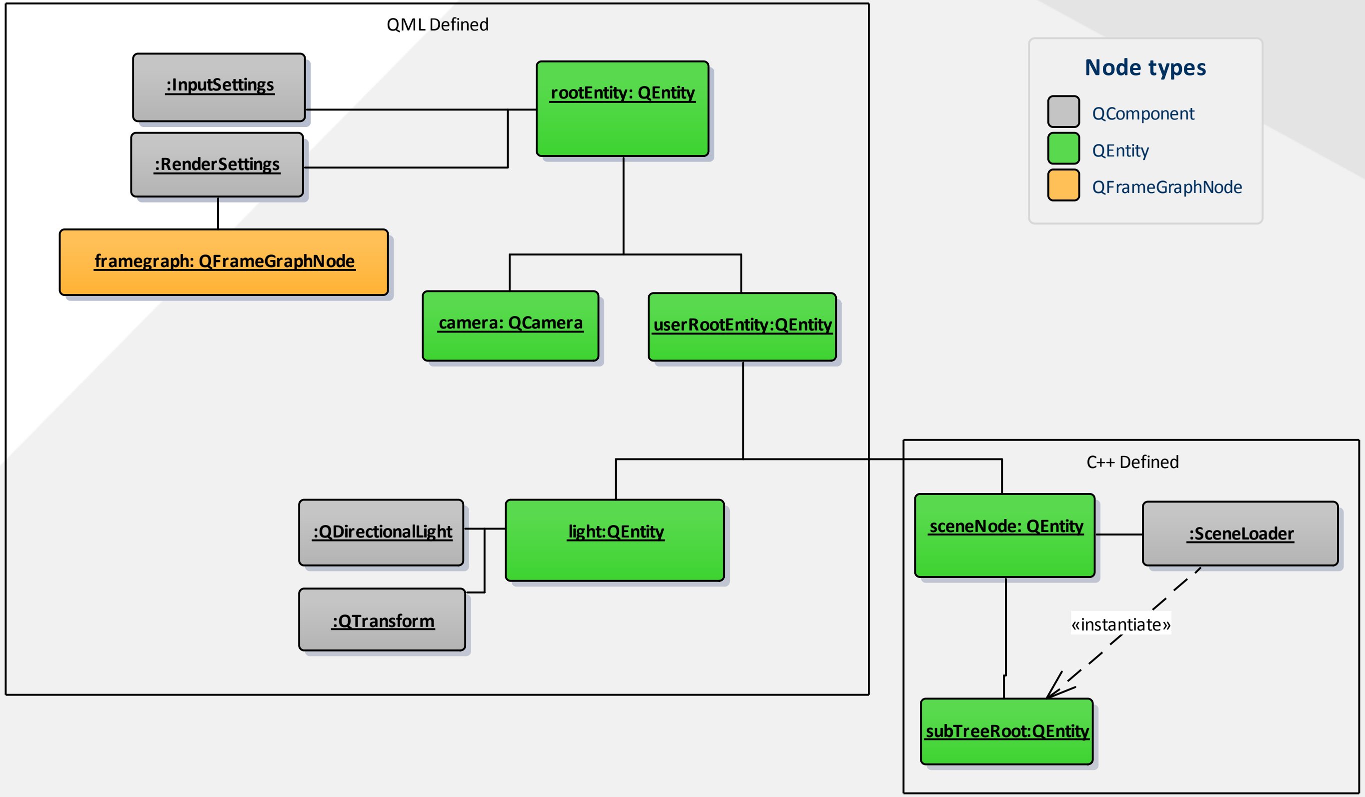
Qt 3D implements an Entity Component System (ECS)

- **Scenegraph** is a tree of entities describing the scene to render
- **Framegraph** is a data-driven description of how to render
- **Aspects** process and update entities with specific components



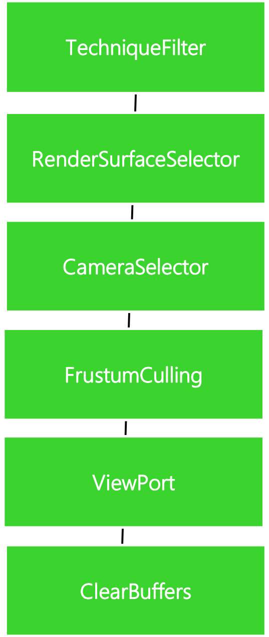
ELT Bit 3D Scenegraph – Single Scene

- Scenegraph is defined by a QML file
- C++ code manages runtime behavior:
  - Loading scene subtree
  - Controlling camera position
  - Assigning materials to entities according to the BIT status



ELT Bit 3D Framegraph – Single Scene

It contains the following nodes:



TechniqueFilter

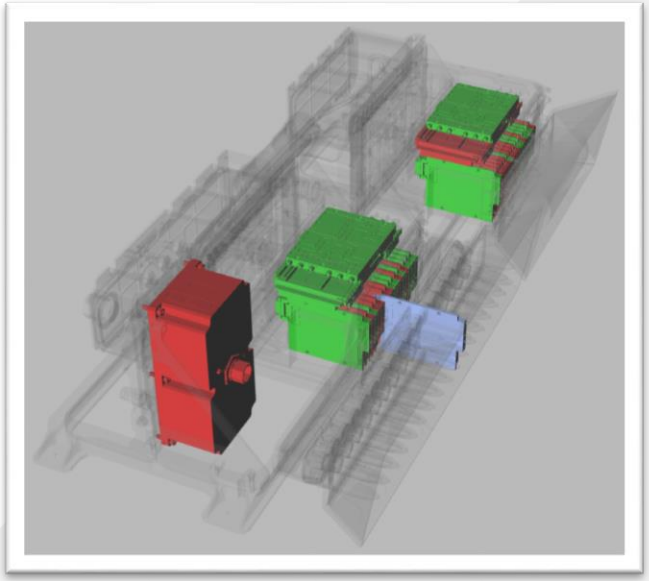
RenderSurfaceSelector

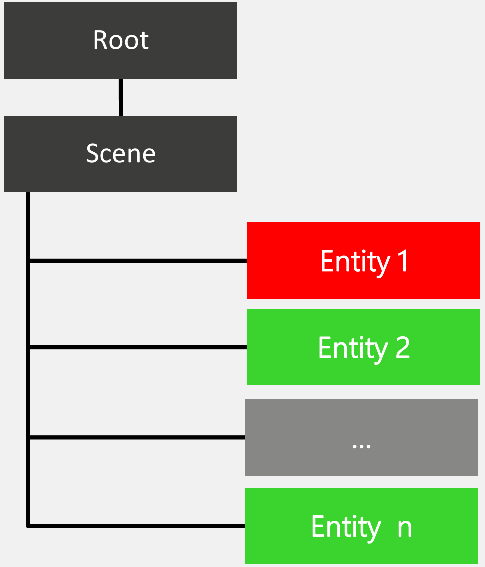
CameraSelector

FrustumCulling

ViewPort

ClearBuffers





Root

Scene

Entity 1

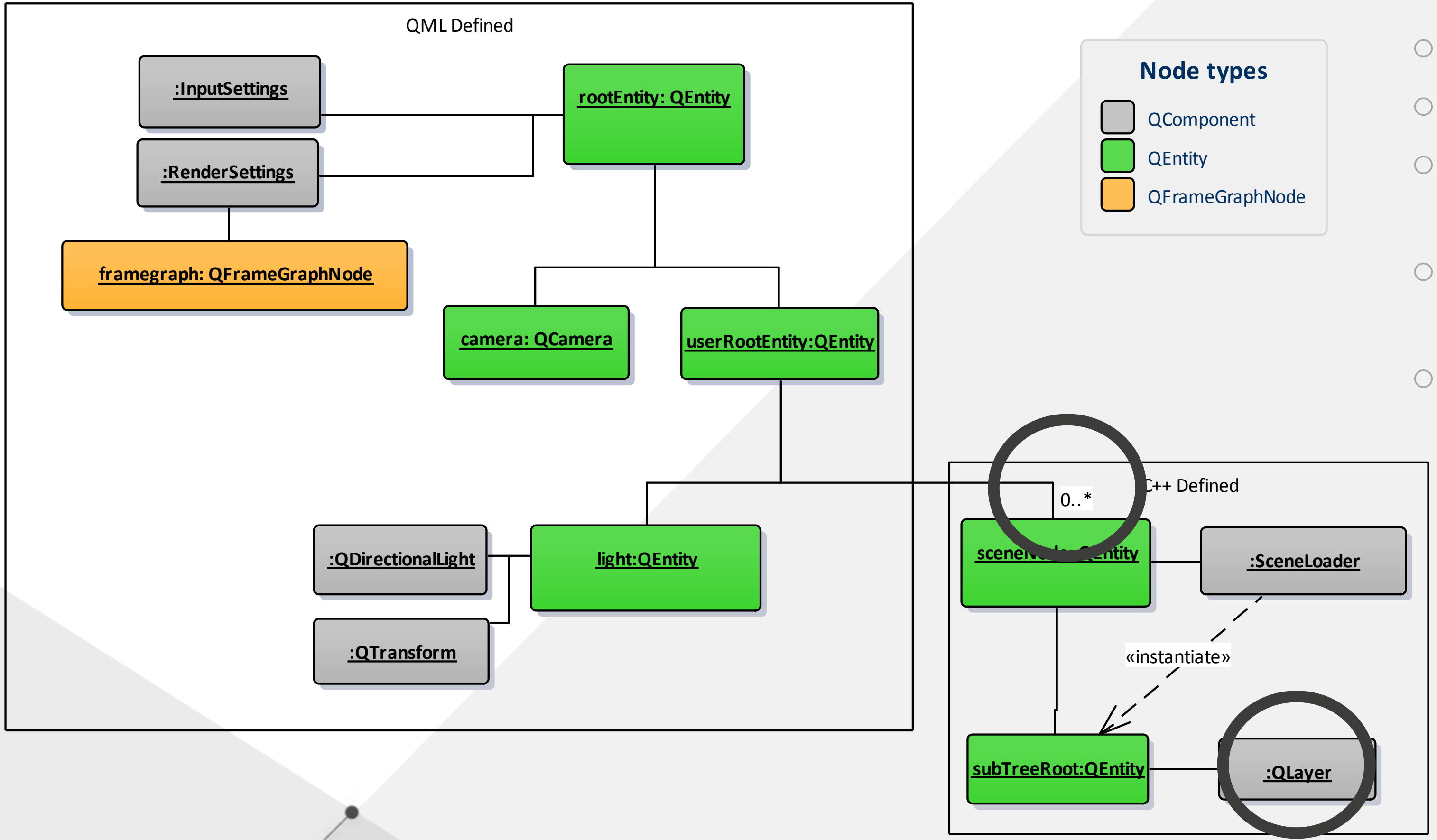
Entity 2

...

Entity n

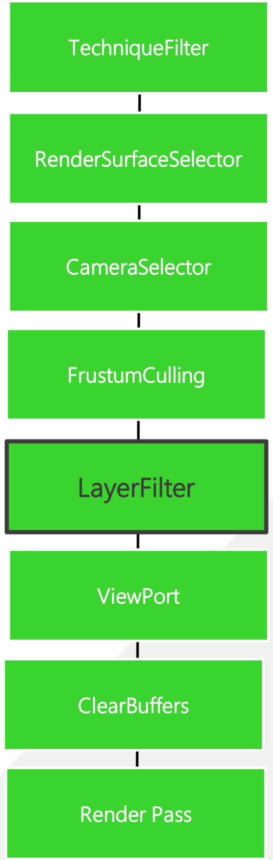
ELT Bit 3D Scenegraph – Multiple Scenes

- Scenegraph defined by a QML file
- C++ code manages runtime behavior:
  - Loading scene subtree
  - Controlling camera position
  - Assigning materials to entities according to the BIT status
  - Associating a layer for each subtree
  - Filtering layers



ELT Bit 3D Framegraph - Multiple Scenes

- unique tree
- one subtree filtered by layer



TechniqueFilter

RenderSurfaceSelector

CameraSelector

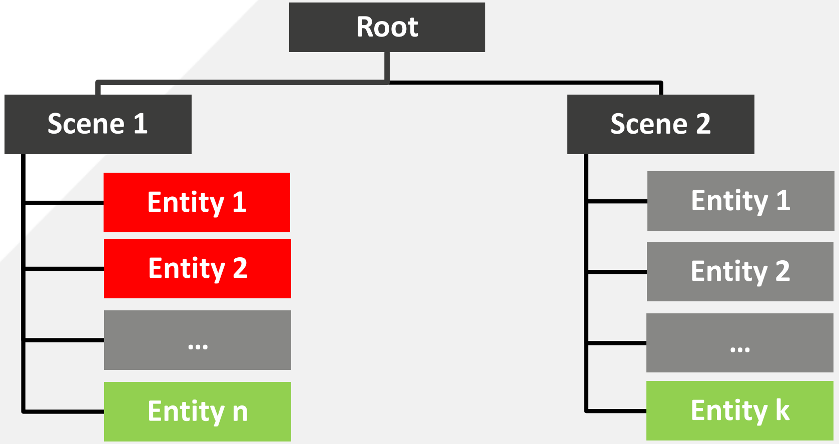
FrustumCulling

LayerFilter

ViewPort

ClearBuffers

Render Pass



Root

Scene 1

Entity 1

Entity 2

...

Entity n

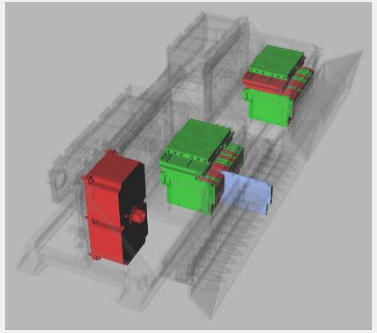
Scene 2

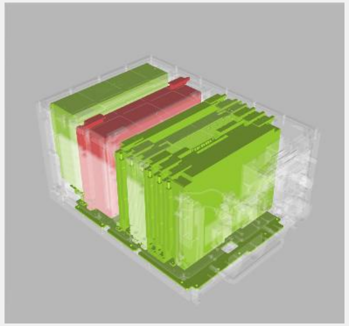
Entity 1

Entity 2

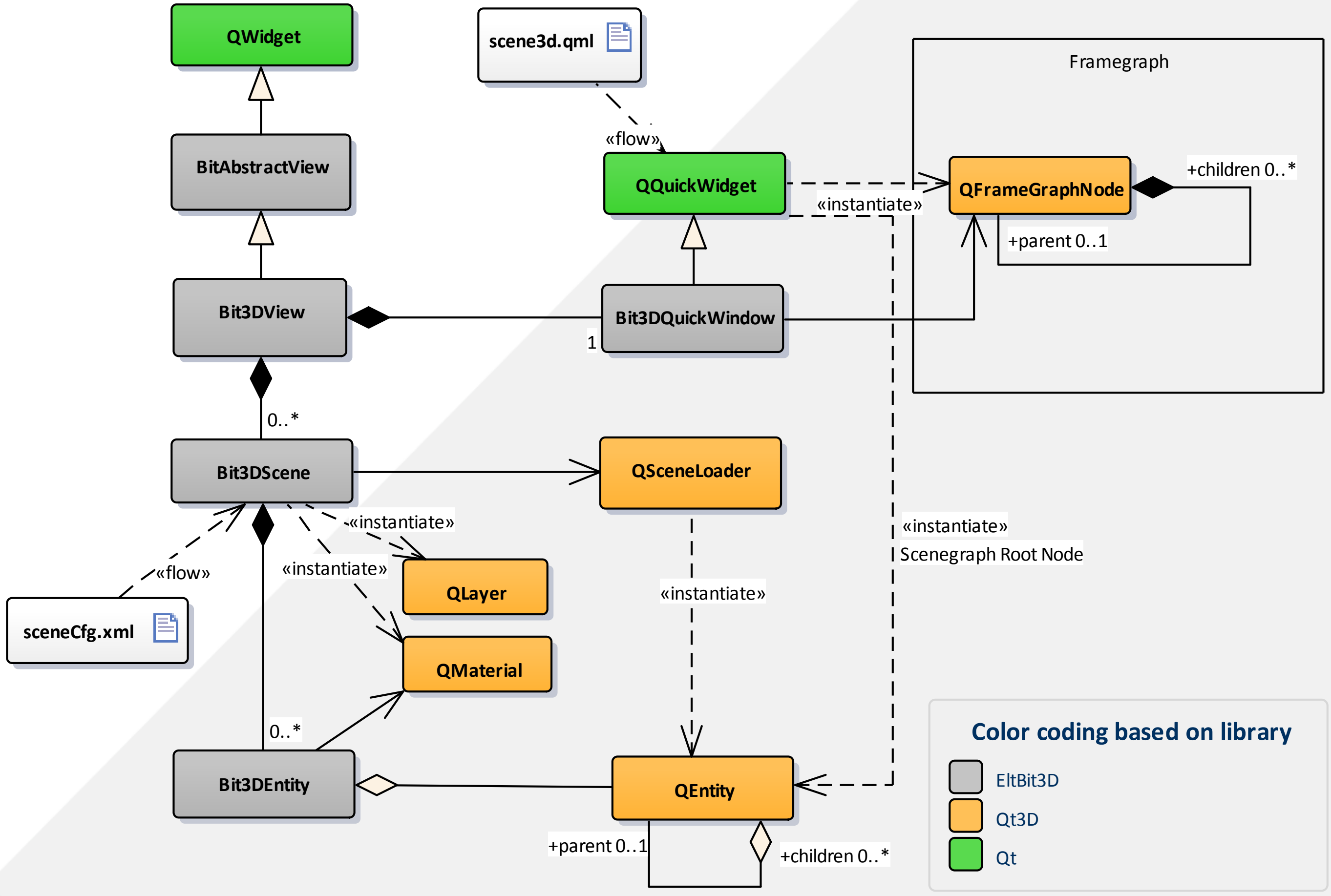
...

Entity k





ELT Bit 3D Engine



Color coding based on library

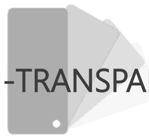
- ⬜ EltBit3D
- 🟧 Qt3D
- 🟩 Qt

Bit3DScene Overview

It’s responsible for:

- Loading a qgltf file and creating the entity’s subtree
- Associating a material to each entity according to:

Unreferenced Entities

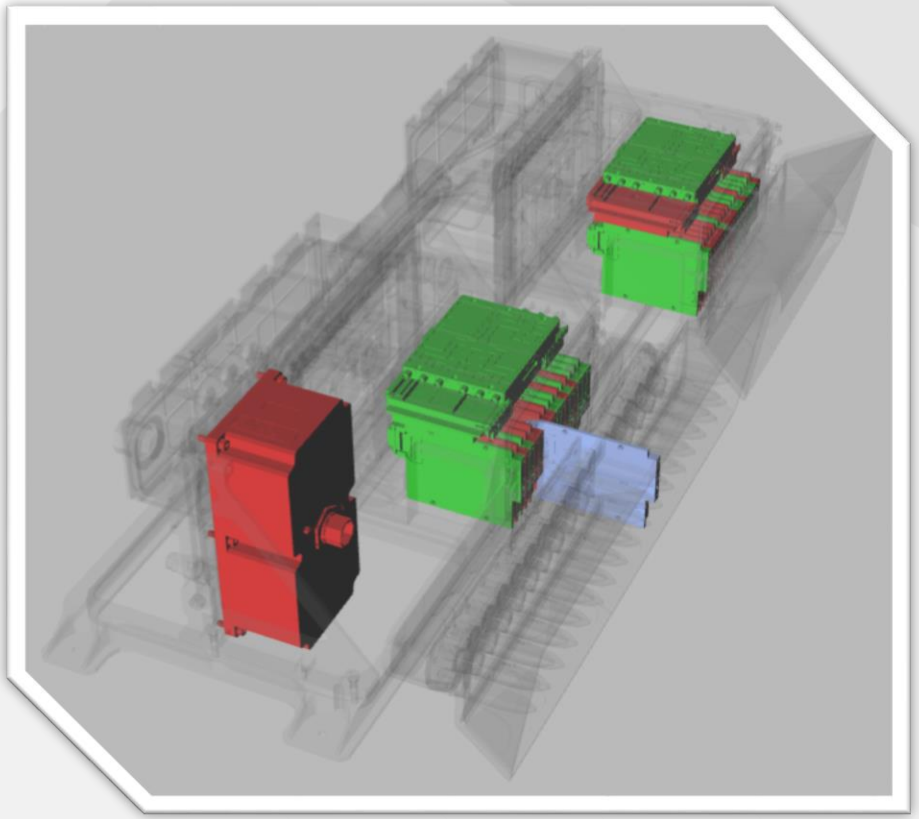


SEMI-TRANSPARENT

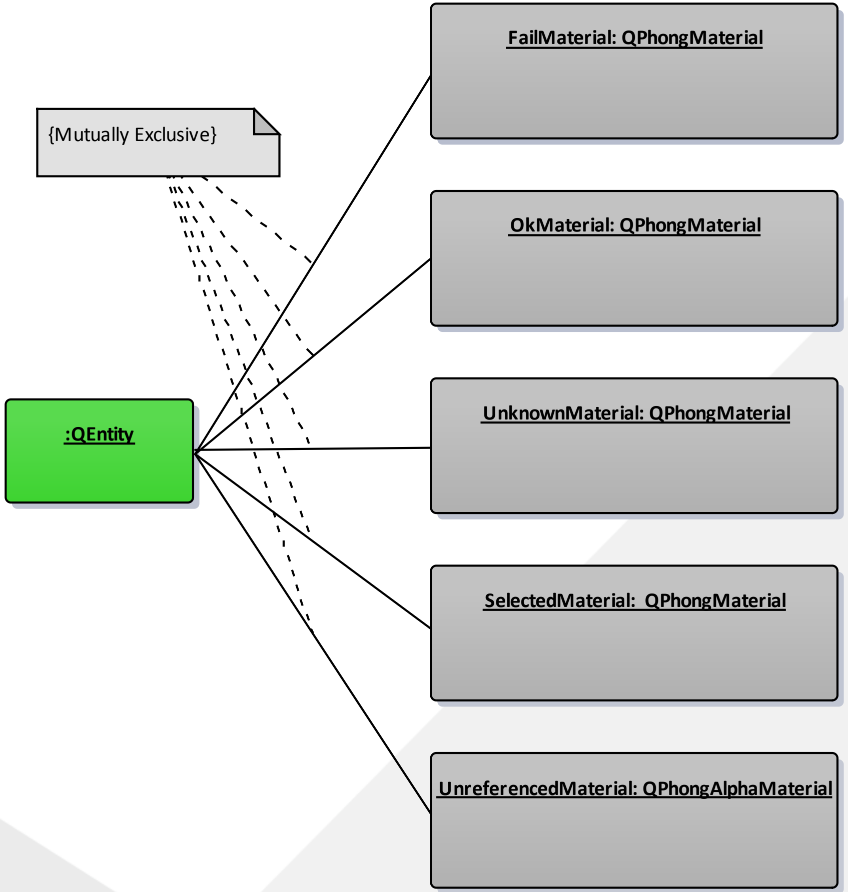
Referenced Entities

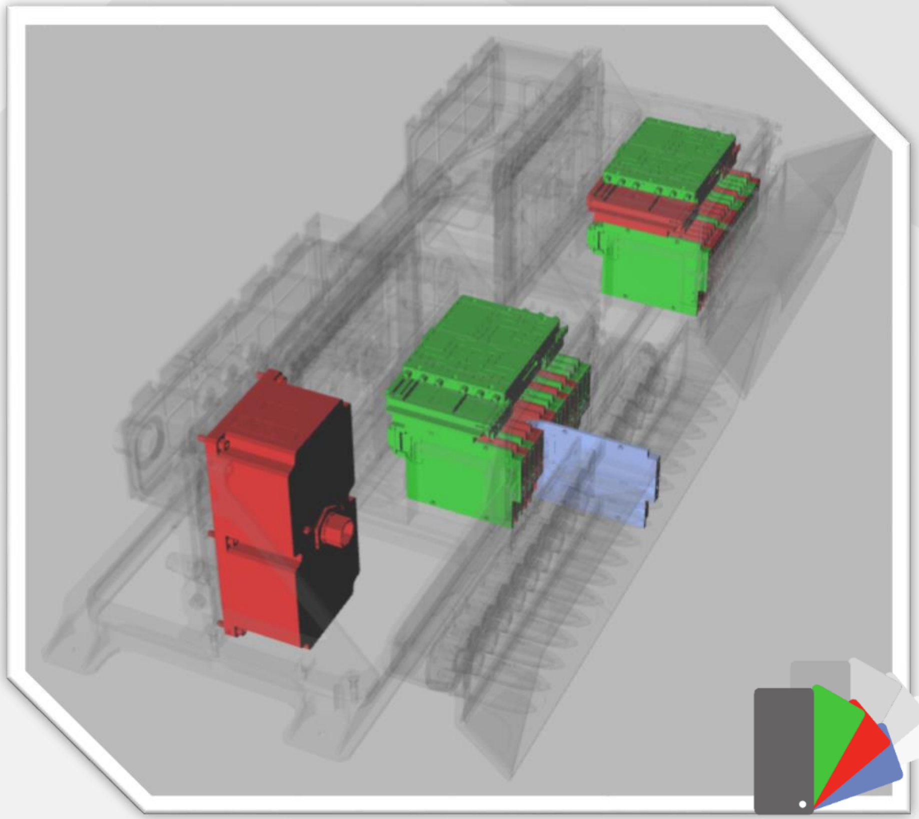
MATT

- Computing bounding-boxes for each entity
- Setting the behavior of any referenced entity according to the scene configuration file



Materials Management

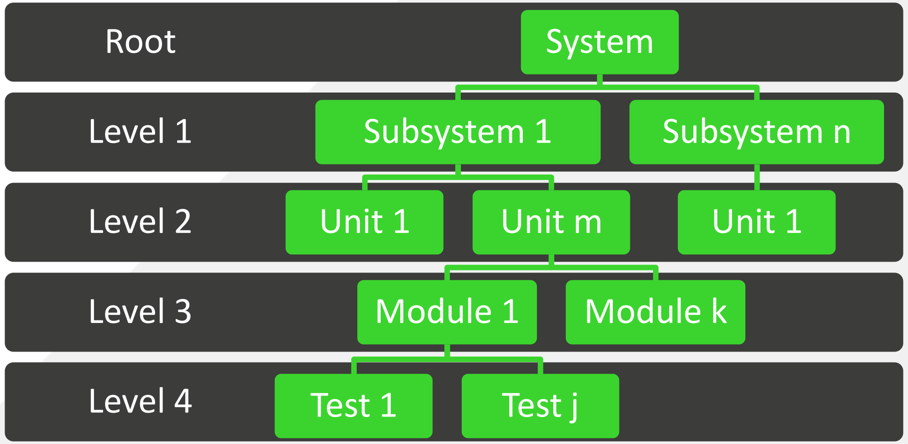

{Mutually Exclusive}
:QEntity
FailMaterial: QPhongMaterial
OkMaterial: QPhongMaterial
UnknownMaterial: QPhongMaterial
SelectedMaterial: QPhongMaterial
UnreferencedMaterial: QPhongAlphaMaterial



UNCLASSIFIED

BIT System Modeling

- Describes the system and all its components in a hierarchical model
- Consists of five-level hierarchical tree
- Has a logical sub-division that can reflect the physical structure of the system



Root — System

Level 1 — Subsystem 1, Subsystem n

Level 2 — Unit 1, Unit m, Unit 1

Level 3 — Module 1, Module k

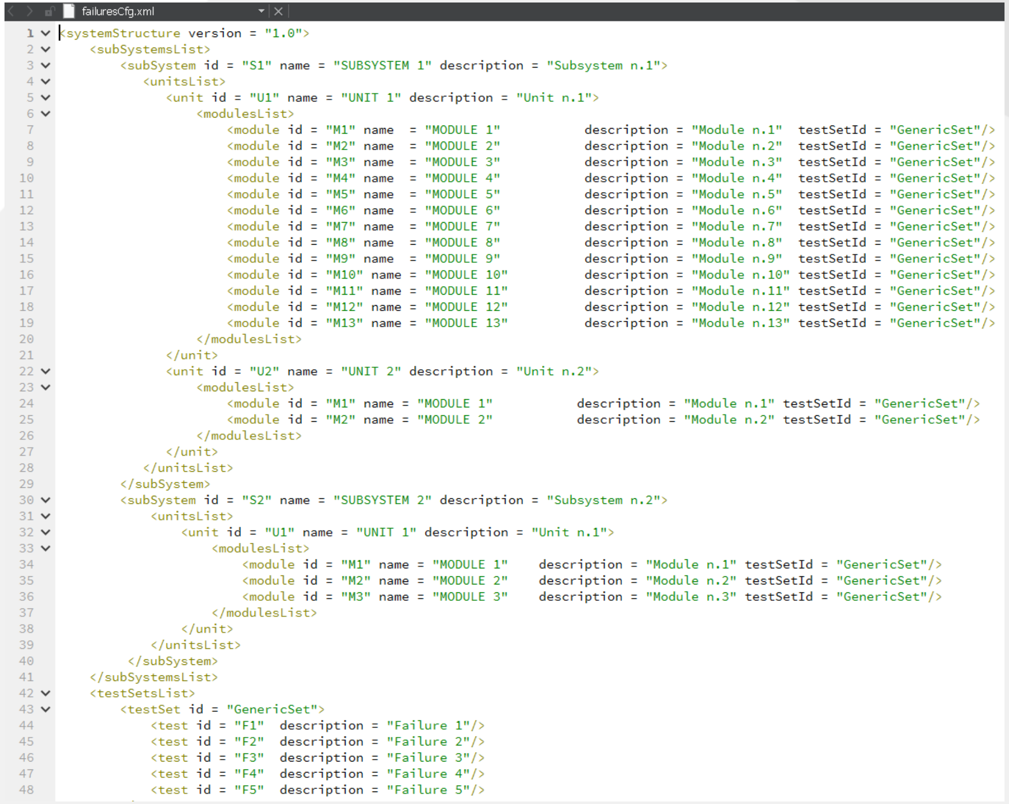
Level 4 — Test 1, Test j

A system configuration file (.xml) describes the system composition

BIT System Configuration File (Xml file)

It contains the System composition.

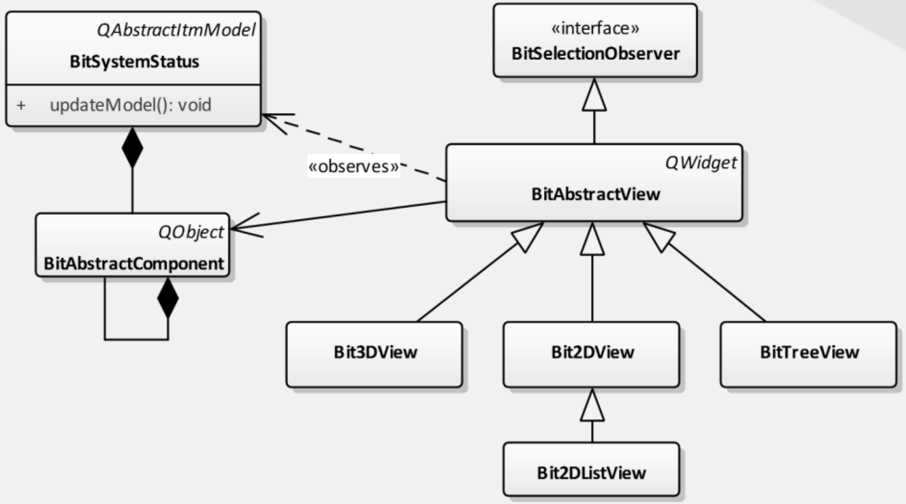
Each component is identified by a unique ID.



```xml
<systemStructure version = "1.0">
    <subSystemsList>
        <subSystem id = "S1" name = "SUBSYSTEM 1" description = "Subsystem n.1">
           <unitsList>
              <unit id = "U1" name = "UNIT 1" description = "Unit n.1">
                  <modulesList>
                      <module id = "M1" name  = "MODULE 1"          description = "Module n.1"  testSetId = "GenericSet"/>
                      <module id = "M2" name  = "MODULE 2"          description = "Module n.2"  testSetId = "GenericSet"/>
                      <module id = "M3" name  = "MODULE 3"          description = "Module n.3"  testSetId = "GenericSet"/>
                      <module id = "M4" name  = "MODULE 4"          description = "Module n.4"  testSetId = "GenericSet"/>
                      <module id = "M5" name  = "MODULE 5"          description = "Module n.5"  testSetId = "GenericSet"/>
                      <module id = "M6" name  = "MODULE 6"          description = "Module n.6"  testSetId = "GenericSet"/>
                      <module id = "M7" name  = "MODULE 7"          description = "Module n.7"  testSetId = "GenericSet"/>
                      <module id = "M8" name  = "MODULE 8"          description = "Module n.8"  testSetId = "GenericSet"/>
                      <module id = "M9" name  = "MODULE 9"          description = "Module n.9"  testSetId = "GenericSet"/>
                      <module id = "M10" name = "MODULE 10"         description = "Module n.10" testSetId = "GenericSet"/>
                      <module id = "M11" name = "MODULE 11"         description = "Module n.11" testSetId = "GenericSet"/>
                      <module id = "M12" name = "MODULE 12"         description = "Module n.12" testSetId = "GenericSet"/>
                      <module id = "M13" name = "MODULE 13"         description = "Module n.13" testSetId = "GenericSet"/>
                  </modulesList>
              </unit>
              <unit id = "U2" name = "UNIT 2" description = "Unit n.2">
                  <modulesList>
                      <module id = "M1" name = "MODULE 1"          description = "Module n.1" testSetId = "GenericSet"/>
                      <module id = "M2" name = "MODULE 2"          description = "Module n.2" testSetId = "GenericSet"/>
                  </modulesList>
              </unit>
           </unitsList>
        </subSystem>
        <subSystem id = "S2" name = "SUBSYSTEM 2" description = "Subsystem n.2">
            <unitsList>
                <unit id = "U1" name = "UNIT 1" description = "Unit n.1">
                    <modulesList>
                        <module id = "M1" name = "MODULE 1"    description = "Module n.1" testSetId = "GenericSet"/>
                        <module id = "M2" name = "MODULE 2"    description = "Module n.2" testSetId = "GenericSet"/>
                        <module id = "M3" name = "MODULE 3"    description = "Module n.3" testSetId = "GenericSet"/>
                    </modulesList>
                </unit>
            </unitsList>
         </subSystem>
    </subSystemsList>
    <testSetsList>
        <testSet id = "GenericSet">
            <test id = "F1"  description = "Failure 1"/>
            <test id = "F2"  description = "Failure 2"/>
            <test id = "F3"  description = "Failure 3"/>
            <test id = "F4"  description = "Failure 4"/>
            <test id = "F5"  description = "Failure 5"/>
```

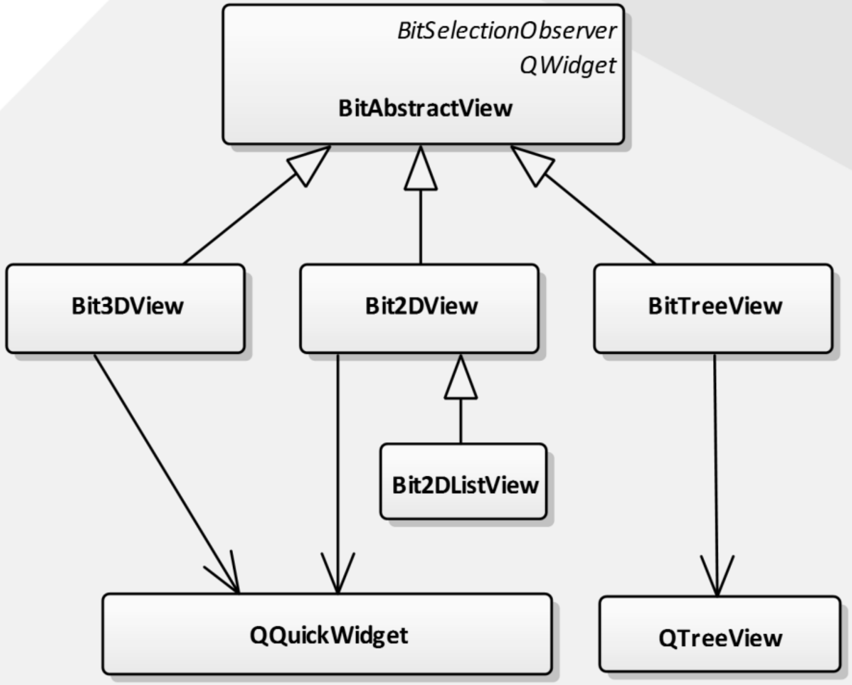
BIT Model/View Architecture

- BitSystemStatus
  - Holds and updates the status of all system components
  - Is the model for all the BIT views
- BitAbstractView
  - Observes the model
  - Provides common interface to different BIT view implementations
  - Notifies mouse activity on each component
- BitAbstractComponent
  - Holds the BIT status of the single system component



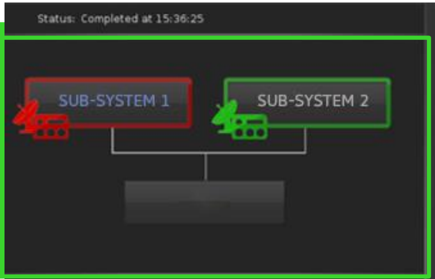
BIT System View Class Diagram

- **Bit2DView:** shows quick items connected to the status of any system component
- **Bit2DListView:** inherits Bit2DView – shows a list of images related to sibling components
- **BitTreeView:** shows the status of any model subtree
- **Bit3DView:** configures and displays 3D scenes

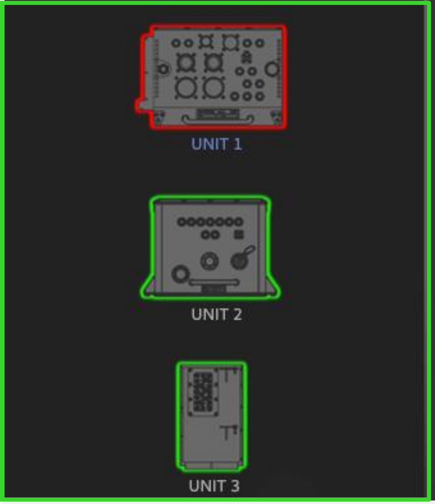


BIT Views

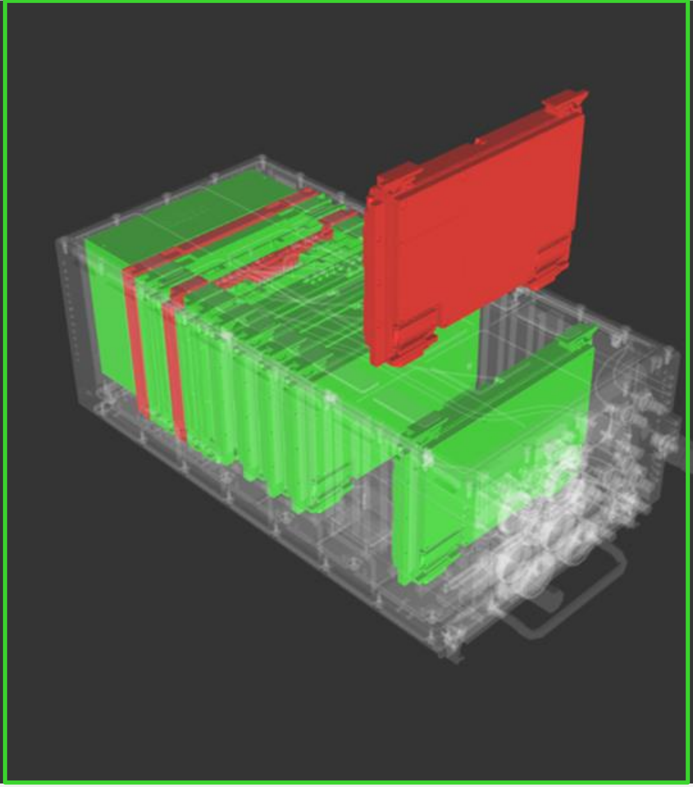
SUB-SYSTEMS VIEW



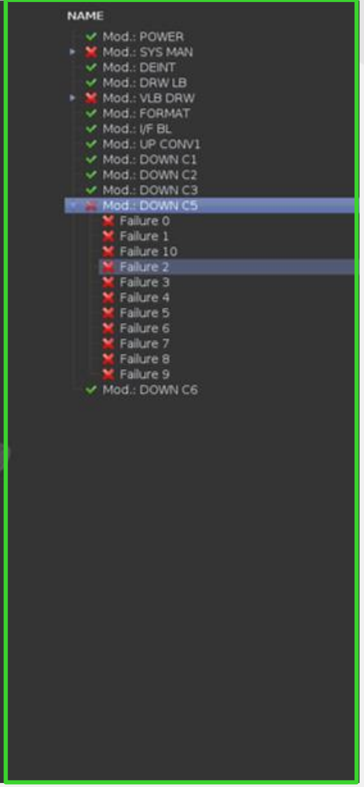
SELECETD SUB-SYSTEM 2D UNIT LIST VIEW



SELECETD UNIT 3D VIEW



SELECTED UNIT TREE VIEW

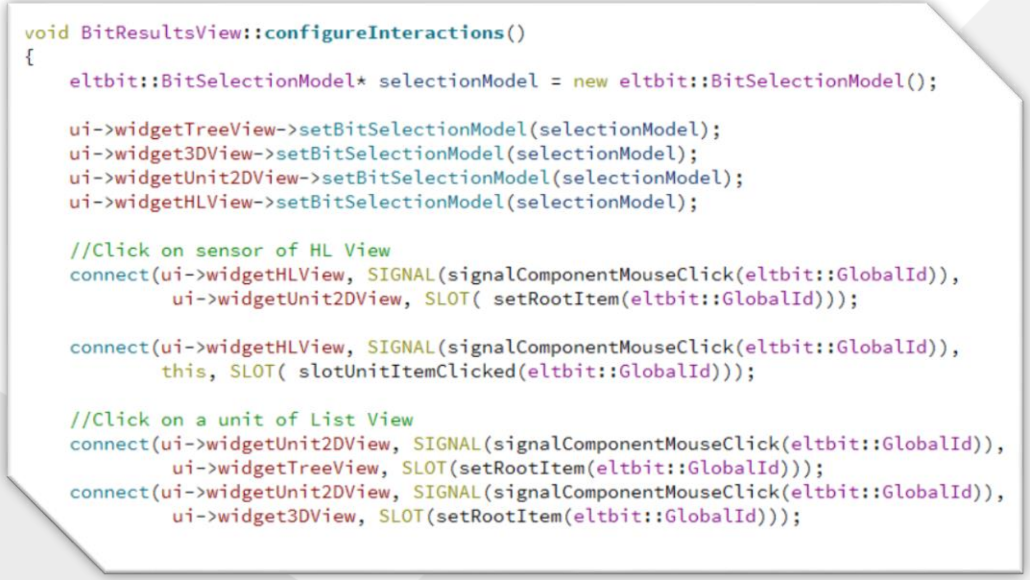


```
NAME
  ✓ Mod.: POWER
▸ ✗ Mod.: SYS MAN
  ✓ Mod.: DEINT
  ✓ Mod.: DRW LB
▸ ✗ Mod.: VLB DRW
  ✓ Mod.: FORMAT
  ✓ Mod.: I/F BL
  ✓ Mod.: UP CONV1
  ✓ Mod.: DOWN C1
  ✓ Mod.: DOWN C2
  ✓ Mod.: DOWN C3
▾ ✗ Mod.: DOWN C5
      ✗ Failure 0
      ✗ Failure 1
      ✗ Failure 10
      ✗ Failure 2
      ✗ Failure 3
      ✗ Failure 4
      ✗ Failure 5
      ✗ Failure 6
      ✗ Failure 7
      ✗ Failure 8
      ✗ Failure 9
  ✓ Mod.: DOWN C6
```

UNCLASSIFIED

Configuring interaction between views

- All **BitAbstractView** classes emit MouseEnter/MouseLeave/MouseClick in response to a mouse event on a single component
- **BitSelectionModel** provides two slots to set the selected and the current component
- It's possible to configure the interaction between views by combining connections of signals and slots of the views and the selection model



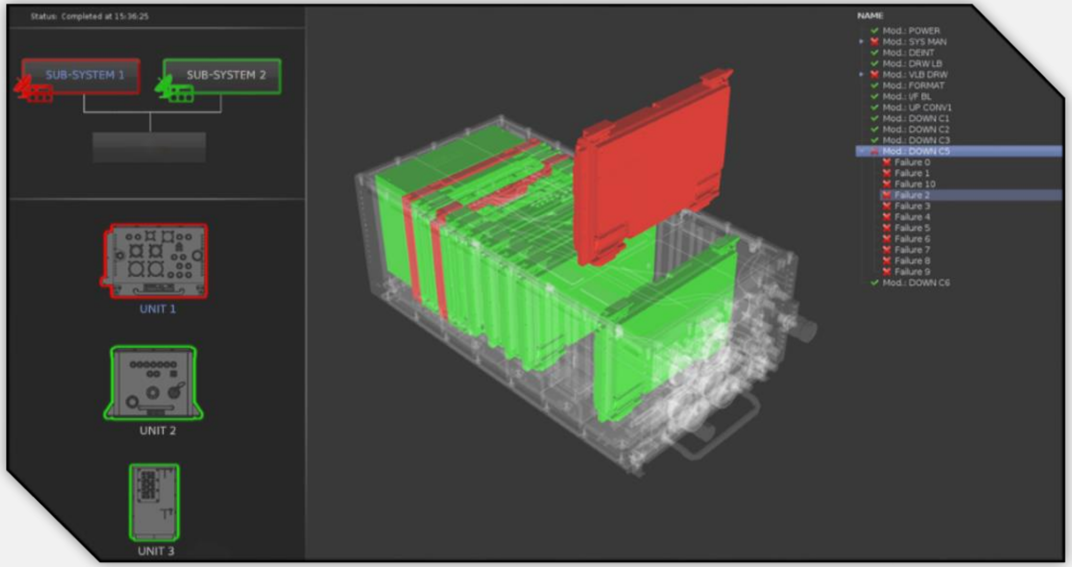
```
void BitResultsView::configureInteractions()
{
    eltbit::BitSelectionModel* selectionModel = new eltbit::BitSelectionModel();

    ui->widgetTreeView->setBitSelectionModel(selectionModel);
    ui->widget3DView->setBitSelectionModel(selectionModel);
    ui->widgetUnit2DView->setBitSelectionModel(selectionModel);
    ui->widgetHLView->setBitSelectionModel(selectionModel);

    //Click on sensor of HL View
    connect(ui->widgetHLView, SIGNAL(signalComponentMouseClick(eltbit::GlobalId)),
            ui->widgetUnit2DView, SLOT( setRootItem(eltbit::GlobalId)));

    connect(ui->widgetHLView, SIGNAL(signalComponentMouseClick(eltbit::GlobalId)),
           this, SLOT( slotUnitItemClicked(eltbit::GlobalId)));

    //Click on a unit of List View
    connect(ui->widgetUnit2DView, SIGNAL(signalComponentMouseClick(eltbit::GlobalId)),
            ui->widgetTreeView, SLOT(setRootItem(eltbit::GlobalId)));
    connect(ui->widgetUnit2DView, SIGNAL(signalComponentMouseClick(eltbit::GlobalId)),
            ui->widget3DView, SLOT(setRootItem(eltbit::GlobalId)));
```



Status: Completed at 15:36:25

Qt3D Picking

QPickerObject component provides high level picking:

- ray-cast based (boundingVolume or triangle based)
- Implicitly associated with mouse device
- emits signals pressed, released, moved, …

Issues:

1. **boundingVolume mode** is faster but imprecise
2. **triangle mode** is precise but very expensive on complex 3d models

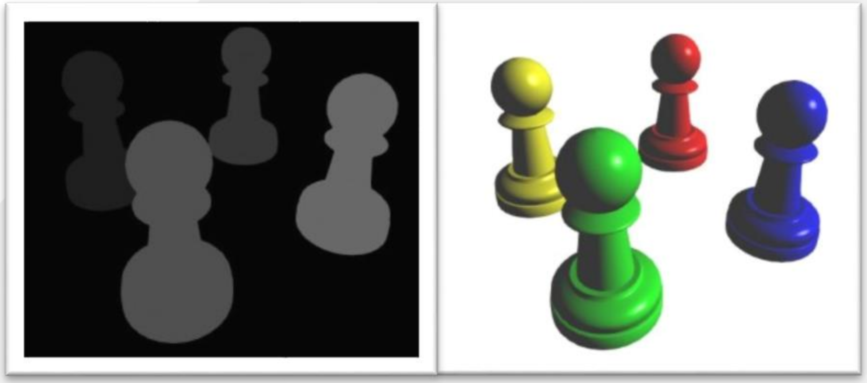
HOW CAN WE OBTAIN A PRECISE AND FAST PICKING FUNCTIONALITY?

Color Picking

The Color Picking solution is based on the following steps:

1. Use color coding to render the scene assigning a specific color to each pickable object
2. Read the pixel color where the mouse is located
3. Decode the color, and hence, identify the object

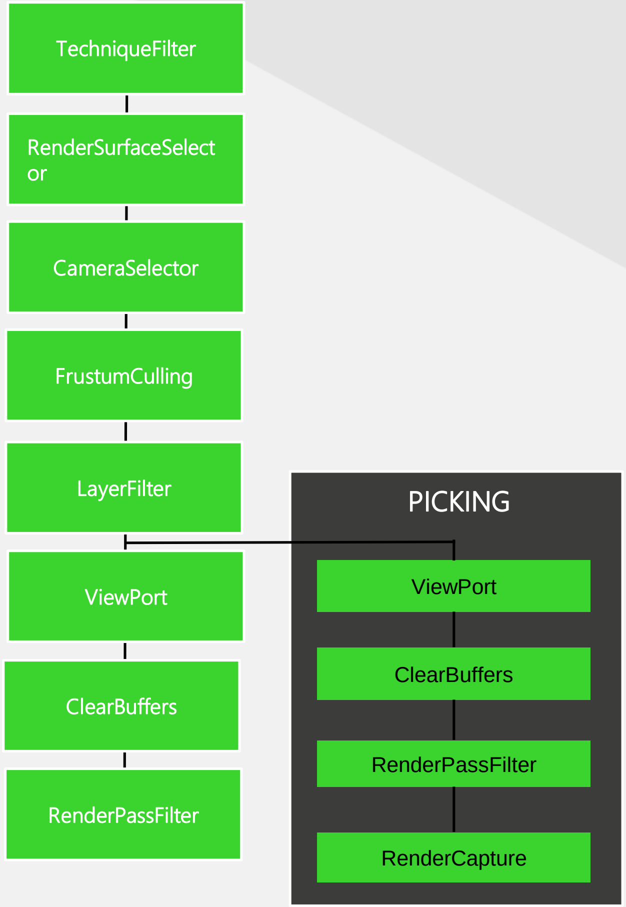
Note: the pseudo-color rendering will never be presented to the user



ELT BIT Color Picking implementation

- We have to modify the framegraph to add the rendering of a color-coded scene:
  1. Adding a new branch
  2. Adding RenderPassFilter nodes

- We have to add a specific RenderPass to the materials
  - Picking RenderPass uses shaders with coded colors
  - Main RenderPass uses standard shaders
  - Picking and Main RenderPass uses two FilterKey nodes used by the RenderPassFilter nodes of the framegraph

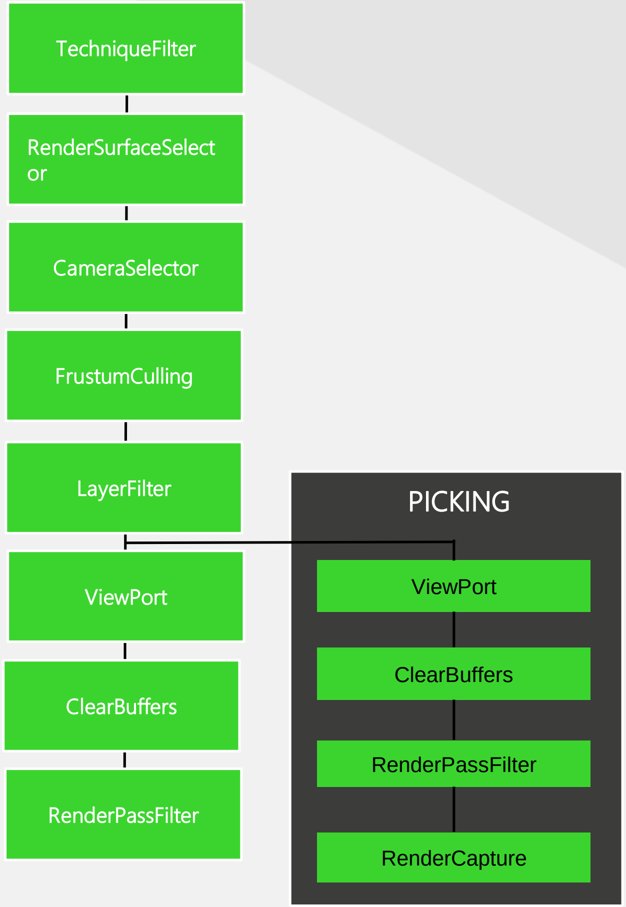


ELT BIT Color Picking implementation

- On each mouse move event a capture request is sent to the RenderCapture node

- The **RenderCapture** reply tell us the coded color of the picked entity

- The **Bit3DView** will emit the mouse events according to decoded identifier



Demo time!