Review of OCC visualization workflow

Roman Lygin (roman.lygin@cadexchanger.com)
Based on discussions with Sergey Anikin (sergey.anikin@opencascade.com) and others

December 2014

Agenda

* Problem statement
* Current and potential workflows
* Data collection and analysis
* Conclusion

Scope

Problem statement:

* Lower responsiveness of OCC-based GUI apps due to heavy computations of visual representations happening in GUI thread

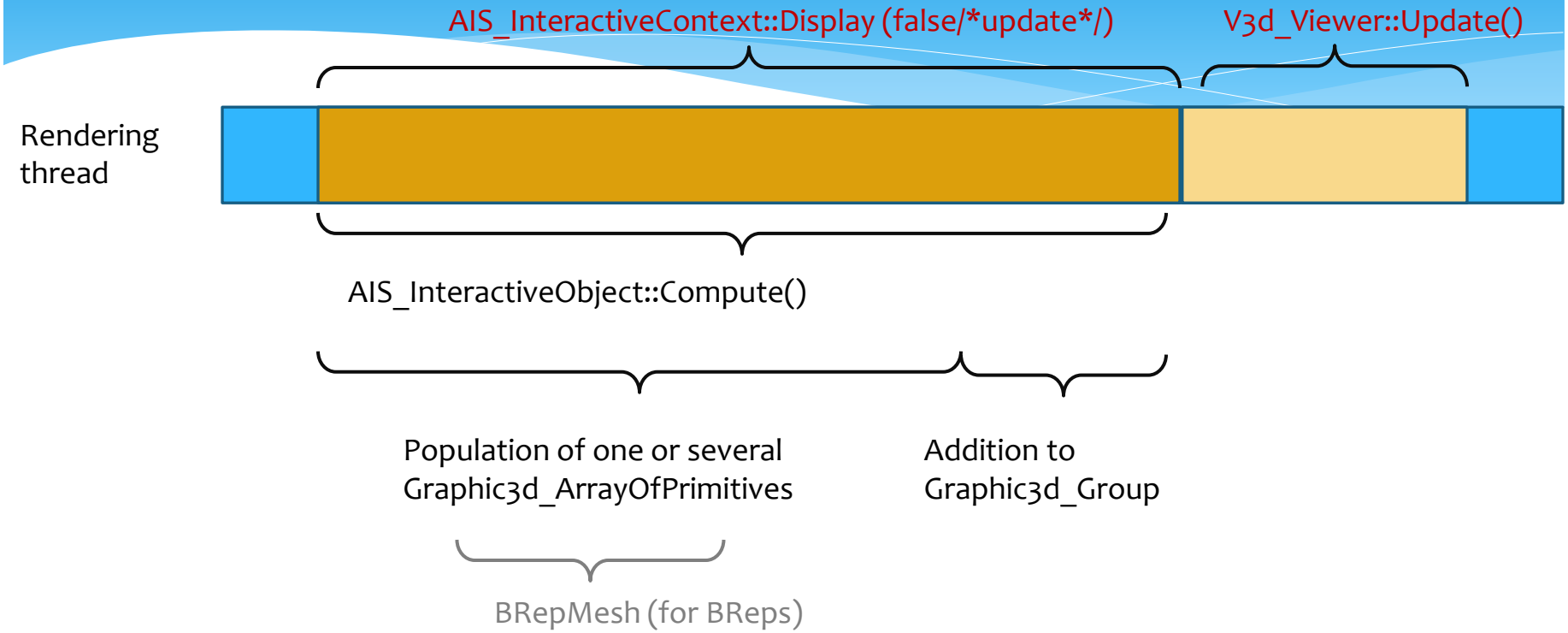
Root-cause:

* AIS_InteractiveContext computes presentations of AIS_InteractiveObjects by calling their virtual Compute() method to populate Prs3d_Presentation
  * Current OCC infrastructure and API do not allow to efficiently precompute presentations in advance

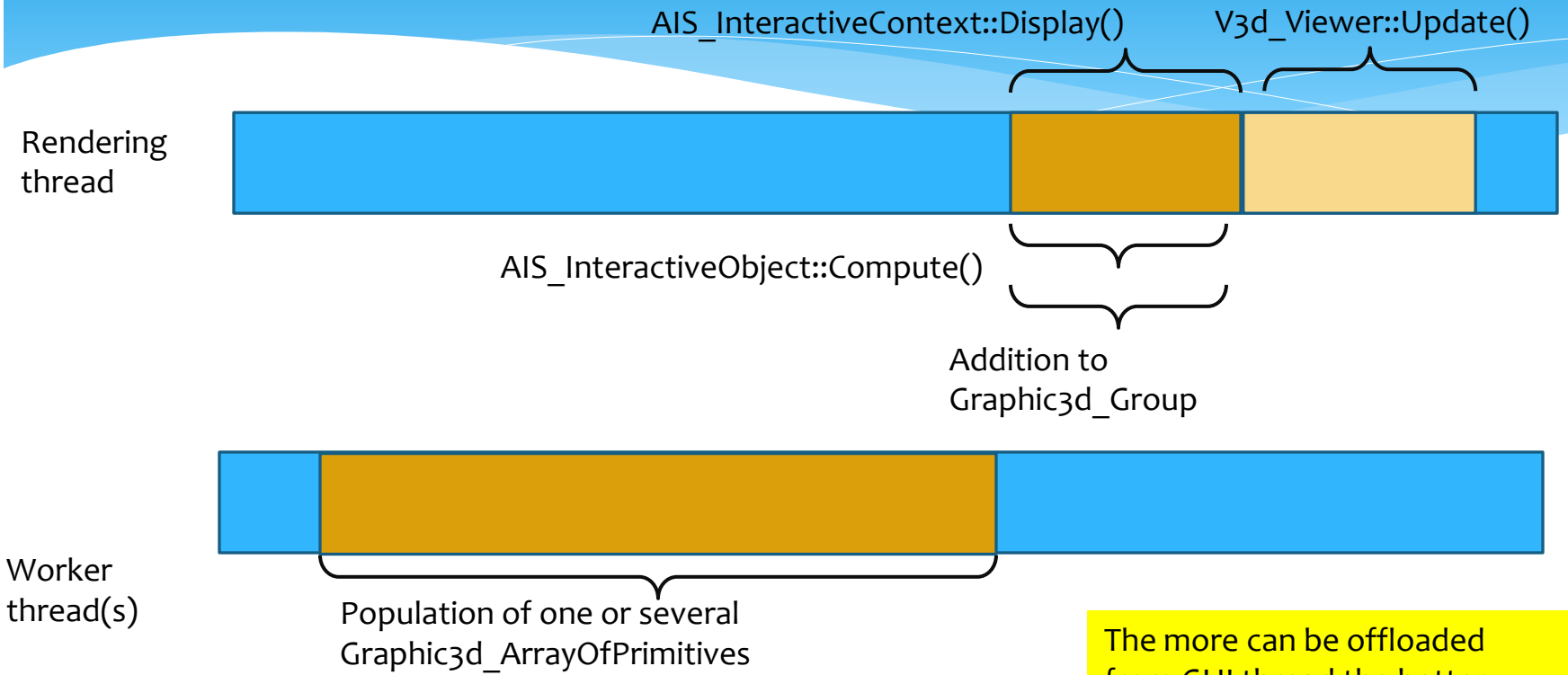
Desired workflow:

* Precompute presentations in worker threads and let GUI thread quickly display them

Current visualization workflow



Potential workflow



The more can be offloaded from GUI thread the better

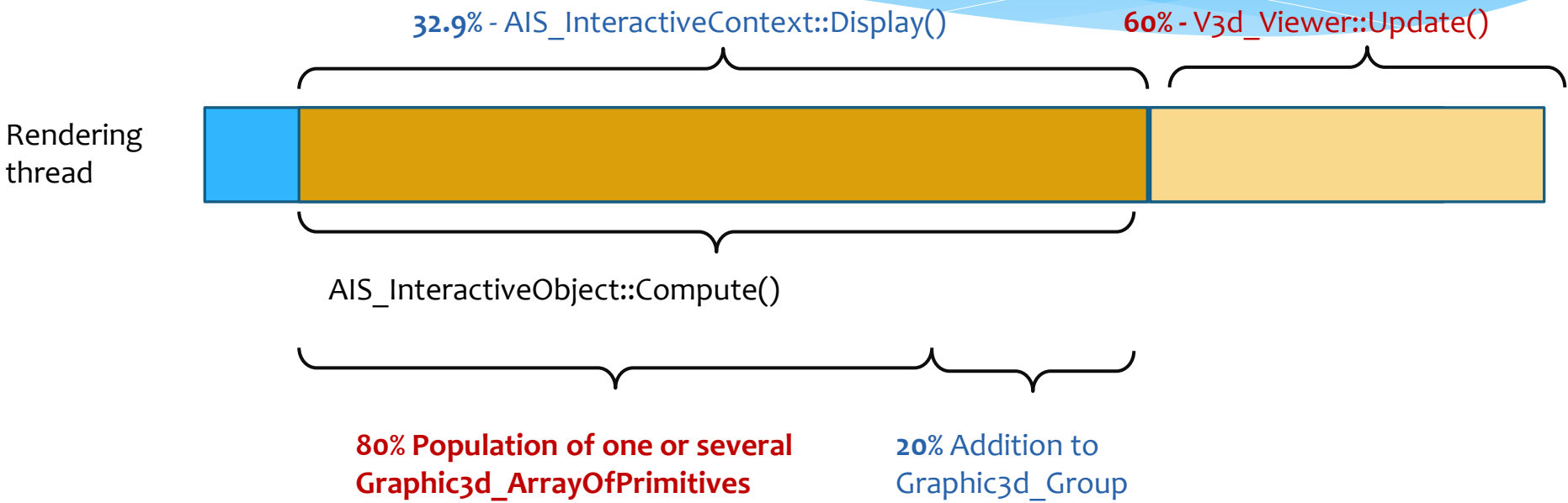
Data collection

* Two types of custom AIS_IO's from CAD Exchanger:
  * Mesh – triangle set displayed in shading mode, using Graphic3d_ArrayOfTriangles::AddVertex() with colors and normals
  * Brep - displayed in shading mode, using StdPrs_ShadedShape.

* Workloads
  * One type of objects at a time: a few dozens meshes, then a few BReps. Each group at once. Each object is relatively heavy-weight
* Open CASCADE DRAW 6.7.1
* Intel VTune Performance Analyzer XE 2015
* Win32 / vc12 /release (/O2)

Data may vary in other cases but hopefully give a representative view

1. Displaying mesh object



Data

		Total time	
cadex::internal::CadExModelXDEBrowser_RepresentationWidget::DoDisplay<class cadex::ModelObject_SceneGraph>		2.358s	
Callees			
Visual3d_ViewManager::Update	61.00%	1.44s	
cadex::internal::ModelPrs_DisplayerApplier::Apply	39.00%	0.98s	
AIS_InteractiveContext::Display()		0.775s	32.87%
cadex::internal::X3DAIS_IndexedFaceSet::Compute		0.735s	
Callees			
Self-time (related to population of arrayoftriangles)	33%		
Graphic3d_Group::AddPrimitiveArray	18.90%		
Graphic3d_ArrayOfTriangles::Graphic3d_ArrayOfTriangles	16.20%		
cadex::X3DGeometry3D_IndexedFaceSet::VertexNormal	11.80%		
Graphic3d_ArrayOfPrimitives::AddVertex	11.00%		
cadex::X3DGeometry3D_IndexedFaceSet::Coordinate	5.70%		
cadex::internal::X3DAIS_Helper::Convert	1.40%		
Graphic3d_Group::SetPrimitivesAspect	1.30%		

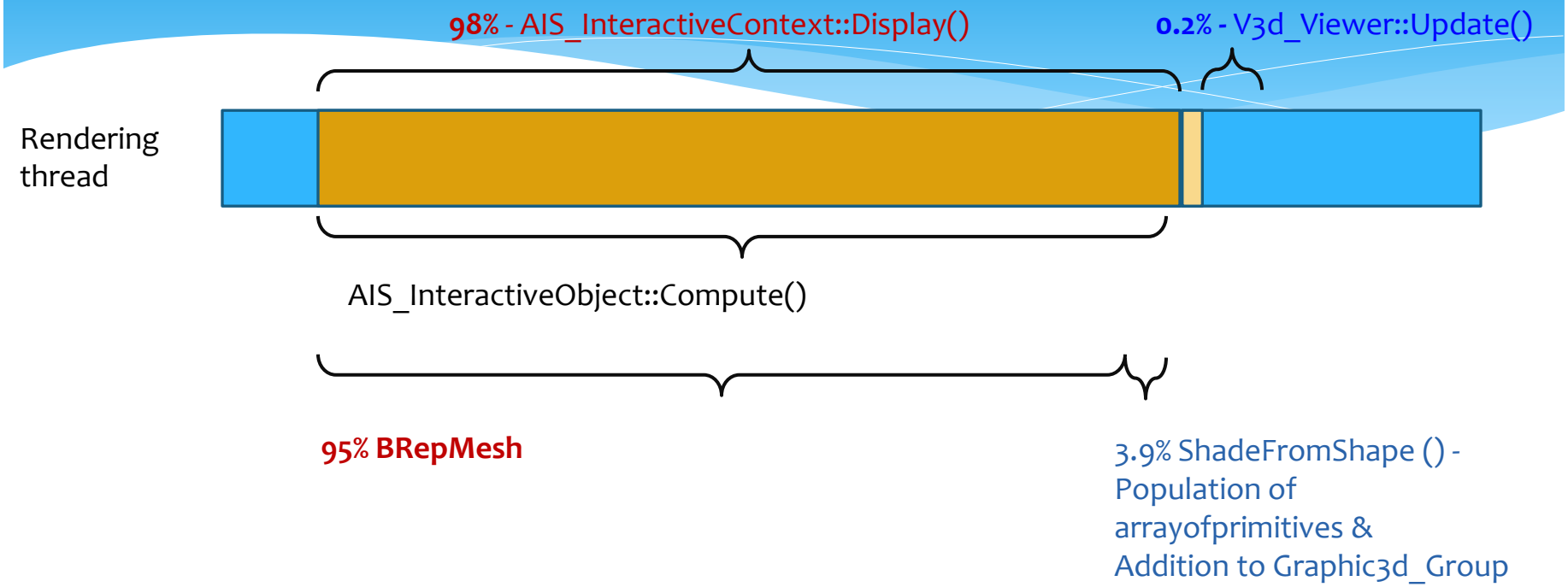
Population of arrayoftriangles

Addition to Group3d

Potential gain

* ~80% of Compute() = 0.8 * 0.735 = 0.588s
* 0.588/2.358s = 25% of total display time.
  * Moderate but still worthwhile.
  * Note: Viewer::Update took 60% on this workload. On other workloads where AIS_IO::Compute() would have greater share, relative gain would be higher

2. Displaying BRep object



Data for ShadeFromShape()

95% Population of arrayoftriangles

2.5% Addition to Group3d

Callees	
`anonymous namespace'::ShadeFromShape	100.00%
StdPrs_ToolShadedShape::Normal	47.50%
Poly_Connect::Poly_Connect	23.90%
_libm_sse2_sqrt_precise	8.80%
Graphic3d_ArrayOfPrimitives::AddEdge	3.80%
TopExp_Explorer::Next	2.80%
Graphic3d_ArrayOfPrimitives::AddVertex	2.20%
StdPrs_ToolShadedShape::Triangulation	2.10%
Graphic3d_Group::AddPrimitiveArray	1.20%
Graphic3d_ArrayOfTriangles::Graphic3d_ArrayOfTriangles	1.10%

Potential gain

* The greatest potential gain for BReps is to precompute tessellation (BRepMesh) in advance
  * This also requires that StdPrs_ShadedShape trusts and does not clean precomputed triangulation (see #23200, http://tracker.dev.opencascade.org/view.php?id=23200)
    * Currently this issue can be worked-around (see backup)
* Precomputaton - 95% of ShadeFromeShape(). Very high potential

Conclusion

* OCC visualization can be improved to allow precomputations
* Prerequisite for BReps - #23200
* To take advantage of the improvement, 3rd party app developers should implement own mechanism
  * using explicit threads or
  * Intel TBB tasks (see http://www.threadingbuildingblocks.org/docs/help/tbb_userguide/Design_Patterns/GUI_Thread.htm)
* Next steps – OCC may consider this for the OCC roadmap

Backup

Work-around for #23200

* Register a global “void” mesher which will not attempt to re-mesh shapes. See next slides.
* Always explicitly call BRepMesh before visualization.

```
BRepMesh_DiscretFactory::Get().SetDefaultName (CadExTestLib_VoidMesher::LibName()); //once in your
app

class CadExTestLib_VoidMesher : public BRepMesh_DiscretRoot
{
public:

    //! Performs meshing algorithm.
    /*! Does nothing.*/
    virtual void Perform() {}

    //! Initializes the mesher with specified parameters.
    Standard_EXPORT static Standard_Integer Discret (const TopoDS_Shape& theShape,
        const Standard_Real theDeflection,
        const Standard_Real theAngle,
        BRepMesh_DiscretRoot*& theAlgo);

    //! Returns a library name exporting the instance of this class.
    Standard_EXPORT static const TCollection_AsciiString& LibName();
};
```

```
Standard_Integer CadExTestLib_VoidMesher::Discret (const TopoDS_Shape& theShape,
                                                   const Standard_Real theDeflection,
                                                   const Standard_Real theAngle,
                                                   BRepMesh_DiscretRoot*&  theAlgo)
{
    theAlgo = new CadExTestLib_VoidMesher();
    theAlgo->SetShape (theShape);
    theAlgo->SetAngle (theAngle);
    theAlgo->SetDeflection (theDeflection);
    return 0;
}

//Holds the name of this dynamic library that contains CadExTestLib_Mesher
static const TCollection_AsciiString aLibName ("CadExTestLib"
#ifdef _DEBUG
                                               "d"
#endif
                                               );

const TCollection_AsciiString& CadExTestLib_VoidMesher::LibName()
{
    return aLibName;
}

}

#if OCC_VERSION_HEX < 0x060800
typedef BRepMesh_DiscretRoot* BRepMesh_PDiscretRoot;
#endif
DISCRETPLUGIN(CadExTestLib_VoidMesher)
```