VELaSSCo

***V**isual Analysis for **E**xtremely **La**rge-**S**cale*
***S**cientific **Co**mputing*

D4.2 – First Prototype of the High Performance Visualization / Scalable Visual Analysis Client(s)

Version #1.2

Deliverable Information

Grant Agreement no	619439
Web Site	http://www.velassco.eu/
Related WP & Task:	WP4, T4.1, T4.2
Due date	September 30, 2015
Dissemination Level	PU
Nature	P
Author/s	Andreas Dietrich, Miguel Pasenau
Contributors	Frank Michel, Abel Coll

Approvals

	Name	Institution	Date	OK
Author	Andreas Dietrich	**FRAUNHOFER**		
Task Leader	Andreas Dietrich	**FRAUNHOFER**		
WP Leader	Frank Michel	**FRAUNHOFER**		
Coordinator	Abel Coll	**CIMNE**		

Change Log

Version	Description of Change
Version 0.1	Topics outline
Version 1.0	First version with description of iFX usage
Version 1.1	Review by Abel and Miguel
Version 1.2	Review by Heidi

Table of Contents

1 Introduction

A central aspect of the VELaSSCo architecture is the employment of client side visualization engines that leverage the capabilities of modern GPUs. Visualization clients are separated from the database infrastructure, and communicate with the platform to send VELaSSCo Queries (VQueries), receive results, and visualize them using one or more GPUs.

This document serves as a companion manual for deliverable D4.2, which is part of the first prototype of the VELaSSCo architecture due in M21. D4.2 includes the implementation of the visualization client infrastructure, which has been described in D4.1 – "Specification of the GPU-Driven Representations and Architecture of the GPU-Based Scientific Visualization Pipeline".

In the following, we will provide a brief overview of the visualization workflow in the VELaSSCo platform, followed by a description of the current prototype implementation of the visualization client and how to operate it in order to perform a simple query.

2 Overview of the Visualization Workflow

The high-level workflow between the visualization client and the VELaSSCo platform is depicted in Figure 1. Operating the graphical user interface of the visualization engine can trigger different types of VQueries, i.e., session queries, direct result queries, or analysis queries. In all cases, the visualization engine calls functions of the platform access library, which will then send a query command to the engine layer that retrieves the requested result from the data layer. More detail of how this works can be found in D2.4, D4.1, D3.4 and D3.2.

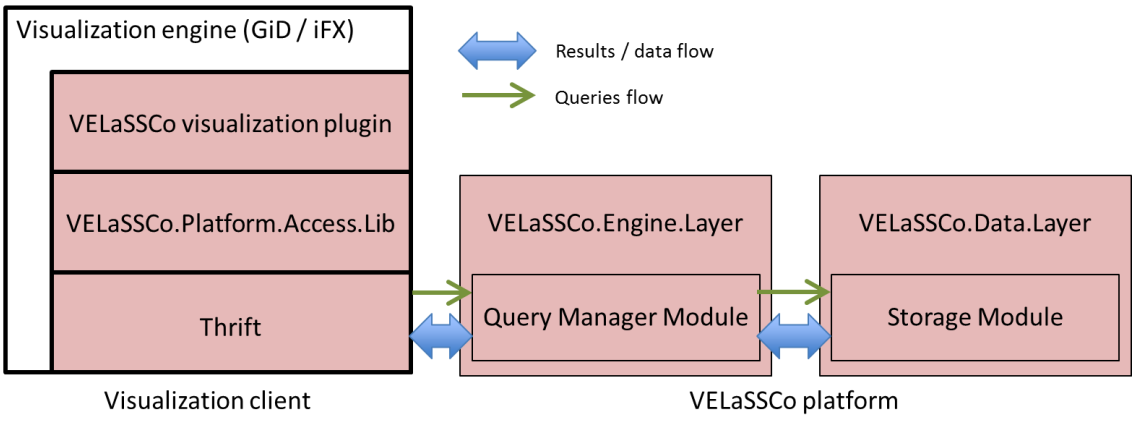

Figure 1. Query workflow triggered by the visualization engine.

The visualization engine serves as a rendering and post-processing framework, and is responsible for generating the final images that are presented to the user. Depending on the specific use case, this can either involve rendering of data that has been pre-computed by the VELaSSCo platform (e.g., a triangle mesh representing an iso-

surface), or post-processing simulation data on the client side (e.g., generating an iso-surface from a volume dataset). In the VELaSSCo project both GiD (CIMNE) and iFX (Fraunhofer) are used as visualization client, and the architecture of the platform is designed to be visualization client independent: a third party client could easily be integrated using the access library.

3 Using the Prototype

In this chapter, we will outline how the visualization engine is used in a first prototype implementation of the VELaSSCo platform. Here, we will focus on iFX; the use of GiD is similar since both make use of the access library, which provides transparent access to the platform.

We will assume that the VELaSSCo platform is running and available on the server side, i.e., the Query Manager module, the Storage module, and database management systems (HBase or EDM) have already been started.

3.1 Activating iFX and the VELaSSCo plugin

After starting up the visualization client (by clicking “Start_VELasSSCo.bat”), the main RPE viewer appears in iFX (see Figure 2).[1] The viewer consists of three main parts: menu and button bars for operation the application, a rendering area used for the actual visualization, and a window for displaying status messages.

[1] The Rapid Prototyping Environment (RPE) is the framework iFX builds on. It allows the system designer to quickly build new applications by providing a modular system of components (such as user interface elements, scene graph handling, GPU rendering, etc.). It also provides a plugin system for easy integration of extensions.

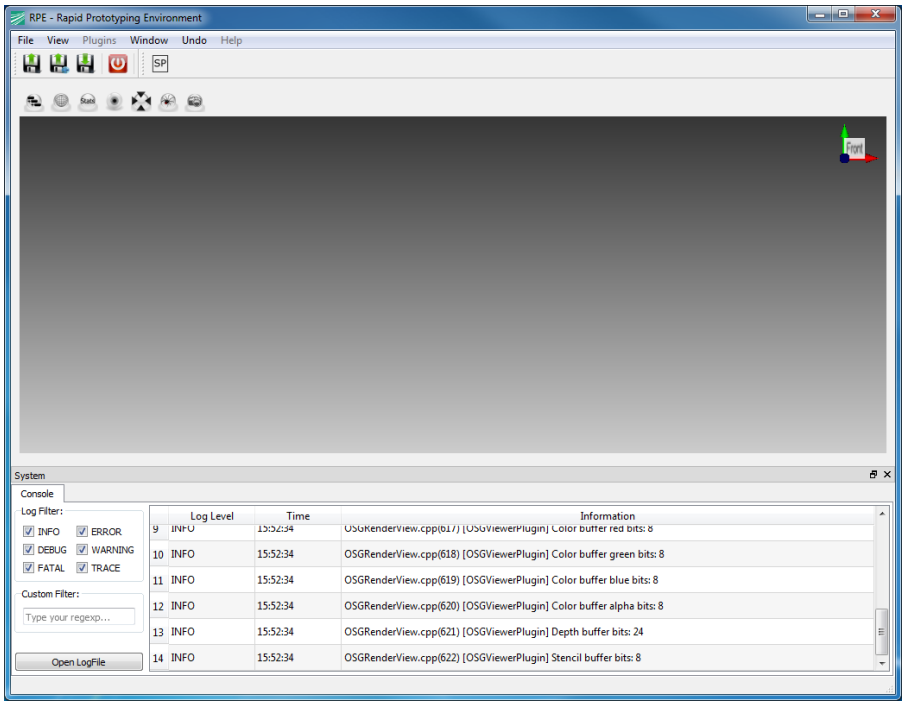

Figure 2. The Rapid Prototyping Environment (RPE) viewer after startup.

The VELaSSCo platform is accessed with the help of a "VELaSSCo plugin". To show the user interface of this plugin we have to select "View→VelasscoPlugin→VELaSSCo" from the drop-down menu (Figure 3). The user interface has two areas, one for connecting to the platform, and a second one for setting parameters for a query. The first prototype supports only three VQueries at the moment of writing, namely VQ-001, "UserLogin", VQ-006, "UserLogout", and VQ-100, "GetResultsFromVerticesID" (see also D3.4).

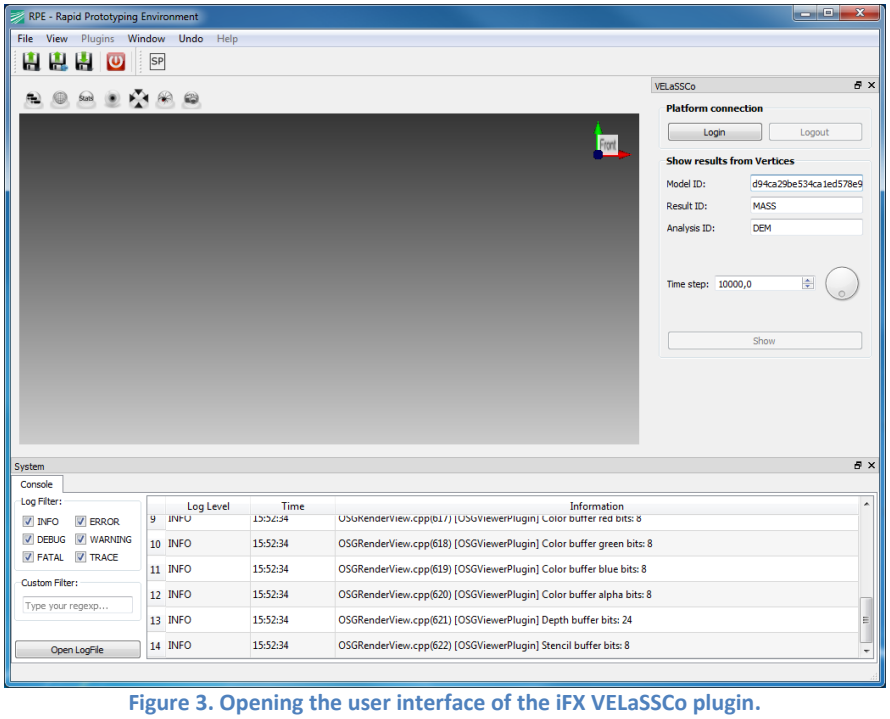

Figure 3. Opening the user interface of the iFX VELaSSCo plugin.

The current function uses VQ-100 to retrieve result values of a given type for a list of vertices from a specific model. In the prototype version, the model ("Model ID"), result type ("Result ID"), and analysis type ("Analysis ID") are hardcoded. For now, only different time steps ("Time step") influence the query.

3.2 Login to the VELaSSCo platform

To log into the VELaSSCo platform, simply press "Login". This opens a login dialog (Figure 4). User and password are preconfigured. For security reasons we currently need to tunnel all platform communication through an SSH connection, therefore the URL has to point to a local port (which is forwarded through SSH to the platform gateway server).

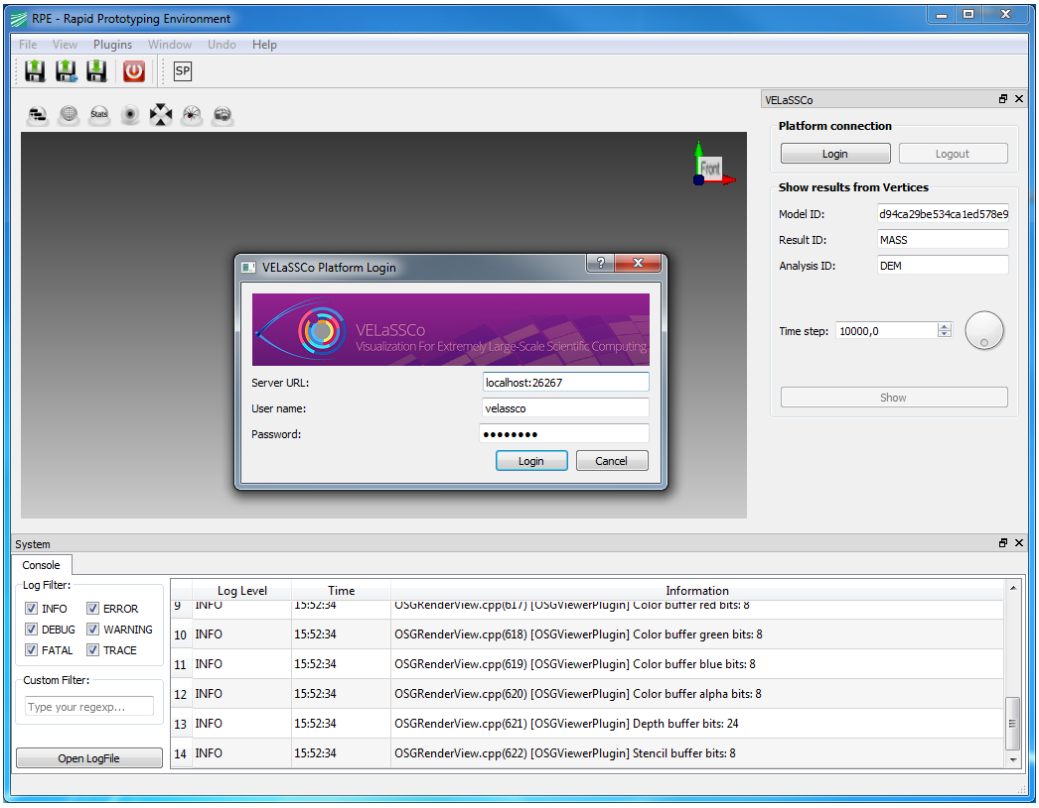

Figure 4. The VELaSSCo platform login dialog.

3.3 Issuing a query

After pressing "Login" (and a successful login operation) the "Show" button becomes selectable in the VELaSSCo pane. Pressing "Show" will now issue the actual query and trigger sending a query command to the platform. This should almost immediately (depending on the quality of the available network connection) result in a display of particles (Figure 5).

This first query requests the spatial position of a list of particles (in the prototype all particles of a specific time step) from the database. In iFX these particles are visualized as spheres. In the viewer, a user can now freely navigate this scene and inspect it. In order to navigate, use moving the mouse and

- the **left mouse button** to **rotate** the scene,
- the **middle mouse button** to **pan**,
- the **right mouse button** to **zoom**.

Another configuration of particles for a different time step can be inspected by changing the “Time step” value and pressing “Show” again.

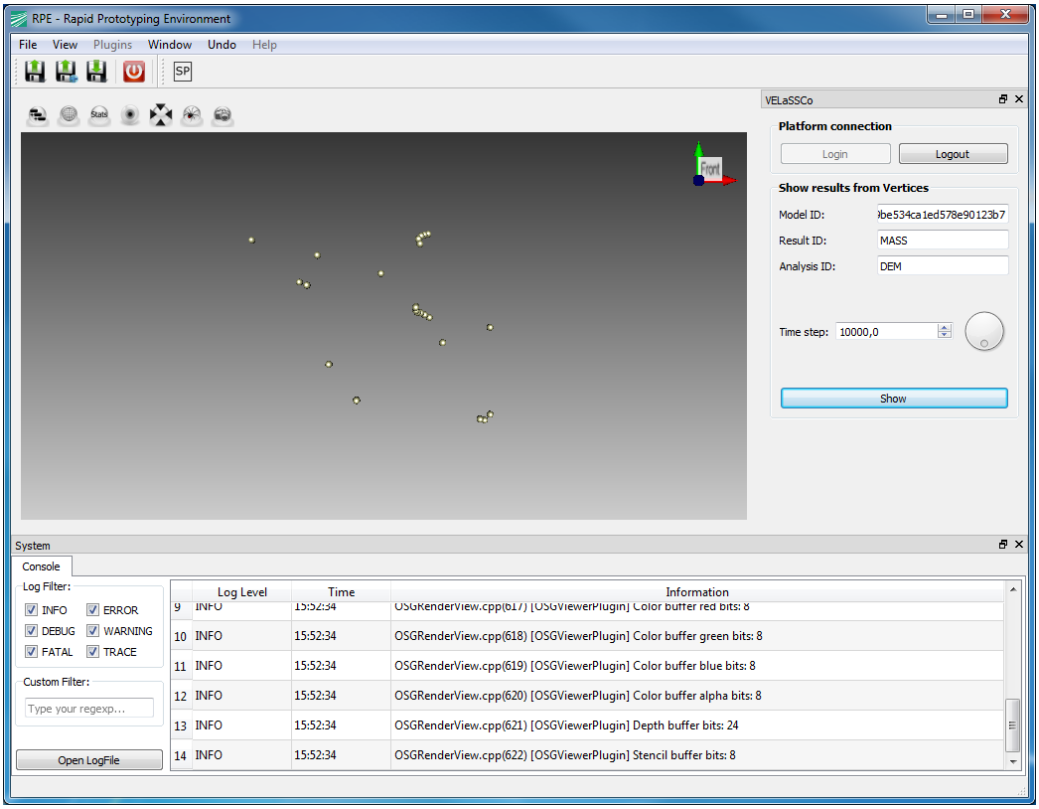

Figure 5. DEM particles resulting from a "GetResultsFromVerticesID" query.

3.4 Logging out

Logging out of the VELaSSCo platform is straightforward, simply press the “Logout” button. After that the “Login” button will be available again.

4 Annex (technical/implementation details)

To communicate with the platform, the iFX plugin sits on top of the VELaSSCo Access Library, which transparently issues queries to the VELaSSCo platform and receive their results.

4.1 Access Library Module

The access library module is a static library that is linked into the iFX plugin. It acts as a client for the Apache Thrift remote procedure call interface (see D4.1). From a front-end the access library is controlled by a “C” application programming interface. This design choice was taken for flexibility, in order to easily generate bindings for other

languages (e.g., Python, etc.), for an easier distribution of the library in binary form (a "C++" interface may require the same compiler version for the front-end as was used to compile the library). Below the API layer, all implementations were done in C++.

The source of the current version of the API C header is listed below.

```
#ifndef VELASSCO_ACCESS_LIB_H
#define VELASSCO_ACCESS_LIB_H


#ifdef __cplusplus
#  include <cstdint>
#  include <cstddef>
#else
#  include <stdint.h>
#  include <stddef.h>
#endif


#ifndef VAL_API
#  if defined(_WIN32)
#    define VAL_API /* __declspec(dllimport) */ // no DLL for now
#  else
#    define VAL_API
#  endif
#endif


/**
 * List of function result codes. Non-zero codes indicate an error.
 */
typedef enum
{
       VAL_SUCCESS                         = 0,

       /* General */
       VAL_UNKNOWN_ERROR                   = 0x0001,
       VAL_INVALID_SESSION_ID               = 0x0002,
       VAL_INVALID_QUERY_PARAMETERS        = 0x0003,

       /* UserLogin */
       VAL_WRONG_URL                       = 0x0100,
       VAL_USER_NOT_ACCEPTED               = 0x0101,
       VAL_SYSTEM_NOT_AVAILABLE            = 0x0102,

       /* UserLogout */
       VAL_LOGOUT_UNSUCCESSFUL             = 0x0200,

       /* GetResultFromVerticesID */
       VAL_RESULT_ID_NOT_AVAILABLE         = 0x0300,
       VAL_SOME_VERTEX_IDS_NOT_AVAILABLE = 0x0301

} VAL_Result;


typedef int64_t VAL_SessionID;


#ifdef __cplusplus
extern "C" {
#endif
```

```
/**
 * The user provides credentials to get access to VELaSSCo data.  The
 * credentials are verified, and, if approved, a session identifier is
 * returned.
 */
VAL_Result VAL_API valUserLogin(                /* in */
                                               const char*      url,
                                               const char*      name,
                                               const char*      password,

                                                /* out */
                                                VAL_SessionID  *sessionID );

/**
 * Logout from the VELaSSCo platform.
 * After successfully logging out, the session identifier is invalid.
 */
VAL_Result VAL_API valUserLogout(               /* in */
                                                VAL_SessionID   sessionID );

/**
 * Given a list of vertices IDs from the model, get the result value of a
 * given type of result for each vertex id of the list.
 */
VAL_Result VAL_API valGetResultFromVerticesID( /* in */
                                               VAL_SessionID   sessionID,
                                               const char*     modelID,
                                               const char*     resultID,
                                               const char*     analysisID,
                                               const int64_t*  vertexIDs,
                                               double          timeStep,

                                               /* out */
                                               const int64_t* *resultVertexIDs,
                                               const double*  *resultValues,
                                               size_t         *resultNumVertices );

/**
 * Translate a numerical result code into an error message string.
 * The memory for the string does not need to be released by the user.
 */
VAL_Result VAL_API valErrorMessage(             /* in */
                                                VAL_Result      error,

                                               /* out */
                                                const char*    *message );

/**
 * API testing.
 */
VAL_Result VAL_API valStartTestServer( const int server_port);

#ifdef __cplusplus
}
#endif

#endif
```