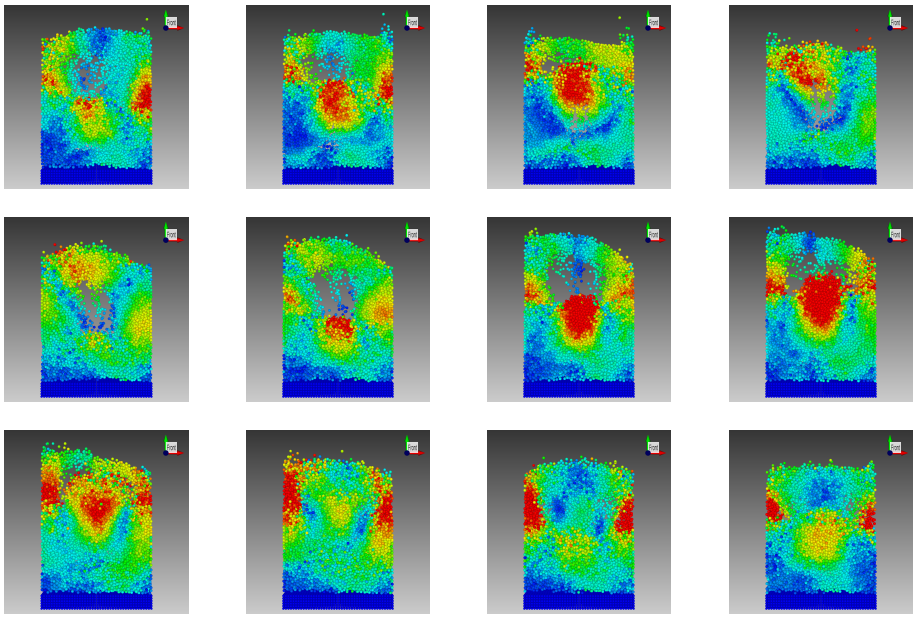


Figure 1: **Particles in different timesteps** are queried from HBase database. The velocity vectors for all particles are queried separately. The particles are color-mapped based on their velocity magnitude.

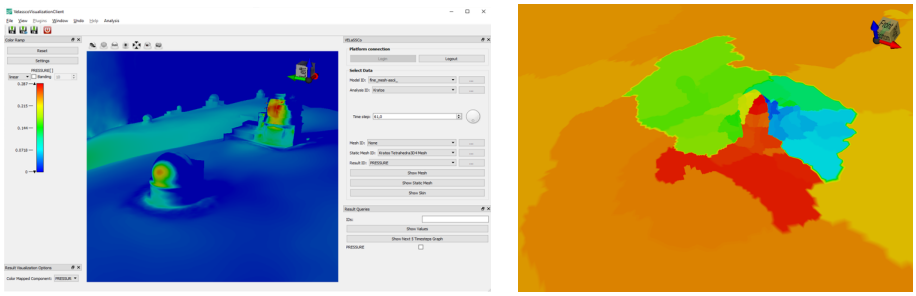


Figure 2: **Telescope simulation boundary mesh** with color-mapped pressure values in left figure and partition IDs in right figure. First the telescope boundary mesh computation is queried from the HBase database. It computes the boundaries of the mesh and the result mesh is sent back to the client. The client query specific results on the boundary mesh vertices. The colormapping is done on the clinet side.

**VELaSSCo project**. "Visualization for Extremely Large Scale Scientific Computing" known as VELaSSCo is an EC funded project which deals with

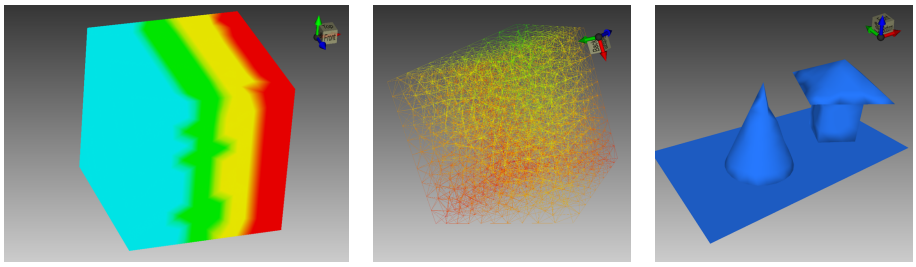


Figure 3: **Querying simulation mesh** from HBase database. The results on the simulation mesh vertices are queried separaretly and visualized using colormapping.

end-user visualization of big data. The idea behind this project is to store the datasets on a distributed database like HBase (Hadoop) or EDM. The user can query a specific part of data in the database or a specific operation on the data in the database. The user's query is processed on the HPC and the result is sent back in a GPU-friendly format for the client to be visualized e.g. Figure 2 shows the boundary mesh of telescope dataset which the pressure values on its surface are color-mapped. First, the client queries for the mesh boundary. The HPC computes the boundary of the mesh and sends it back to the client. The client sends another query to request for pressure values of the vertices on the boundary mesh. It uses received data to visualize the boundary with pressure values color-mapped on its surface. Figure 1 shows the particles in different timesteps which their velocity magnitude is color-mapped. The client queries the particle's position and radius in each timestep. The velocity vectors for the particles are queried afterwards. The velocity magnitude is computed on the client side, and used to color-map the particles' visualization.

In this project, I mainly focused on implementing two direct result queries on the HPC side. They are used to return the result values on a list of vertex indices, and mesh drawing data. Mesh drawing data query result requires to be returned in a GPU friendly format. Therefore, it can be copied directly directly into OpenGL buffers to draw. I implemented these two queries using Thrift C++ API. On the client side, I implemented a plug-in to support VELaSSCo Plugin for indoor application called Rapid Prototyping Environment (RPE). This plugin supports:

- query for available model information, their analysis, results, meshes, and static meshes.
- Query for computing boundary mesh of a specific dataset (Figure 2).
- Query for computing discrete-to-continuum surface for DEM cases.
- Query for Finite Element Meshes (FEM) / Discrete Element Meshes (DEM) and render them (Figure 3).

- Query for results on a specific node in a specific timestep or a vertex's results evaluation in all time steps.
- Query for particles in different timesteps for DEM cases and create an animation which can be play-backed.

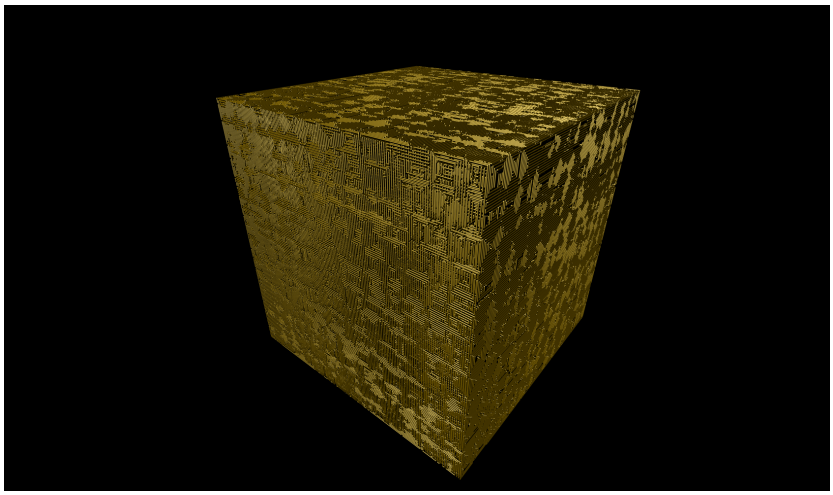


Figure 4: **Pipecube** made of more than 15.6 million pipes, ray-traced on a 1920x1200 display on 17 fps using Intel Embree on an Intel Core i7-4960X.

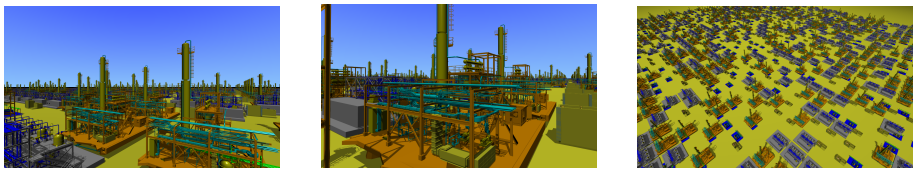


Figure 5: **Factory ray-tracing with Embree**.

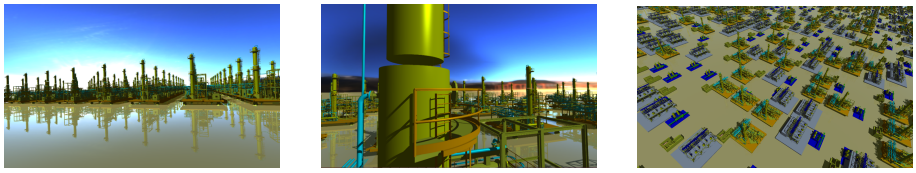


Figure 6: **Factory ray-tracing with Optix**.

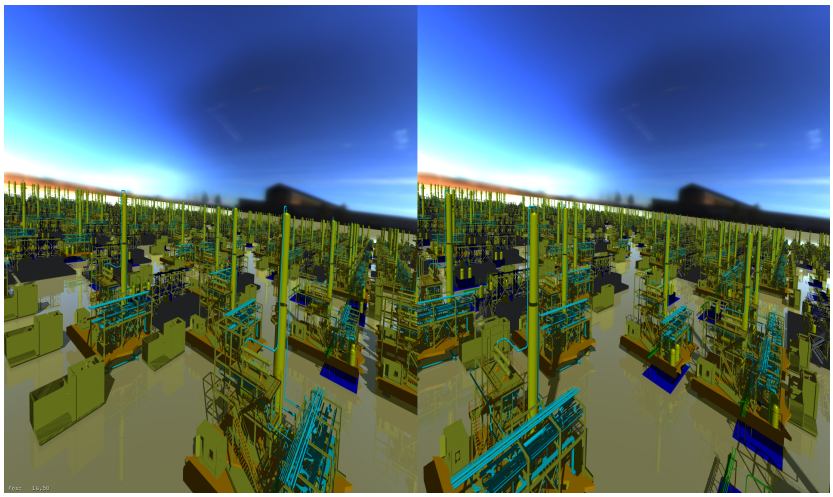


Figure 7: **Factory ray-tracing (Stereoscopic view)**.

**Ray-tracing higher-order primitives**. This task starts with a simple example which was rendering of a pipecube using pipes which are made of two spheres and one cylinder. To speed up renderer, and to get a smooth final image instead of triangulating the primitives, the ray-primitive intersection is computed analytically. The GPU version was implemented by Andreas Dietrich using NVIDIA Optix and I implemented the CPU version using Intel Embree. The CPU version was able to handle very big and smooth pipecubes (Figure 4). This was an starting idea for implementing a version which uses the AVEVA's RVM files to produce the factory models with higher order primitives (Sphere, Box, ...) and then ray-trace it on CPU or GPU. I implemented a factory viewer in two version (NVIDIA Optix and Intel Embree one) which ray-traces a group of factories(figure5 shows the output image achieved by utilizing Intel Embree on Intel Core i7 CPU. Figure 6 shows the NVIDIA Optix output image on a NVIDIA Geforce GTX 980). Additionally, the GPU version was supporting 3D stereoscopic output for 3D projectors (figure 7).

**PCB board software rasterizer**. Implementing a software rasterizer which determines if a pixel on the screen is in the PCB board, which is going to be printed, or it is part of holes. The application gets a group of positive and negative polygons as input. Figure 8 shows the output of the application for sample case. For each pixel, a counter is assumed which is initialized to zero. For each polygon, the pixel is examined. If the pixel is inside a negative polygon the counter is decremented, if it is inside a positive polygon the counter is incremented. Finally, pixels with positive counter value are inside the PCB board (shown as green), pixels with negative values are inside the holes (shown as red) which means the board should be cut to achieve them, and pixels which the counter's value is zero are not in the PCB board area (can be encountered as hole). To speed up point-inside-polygon process, a grid acceleration structure

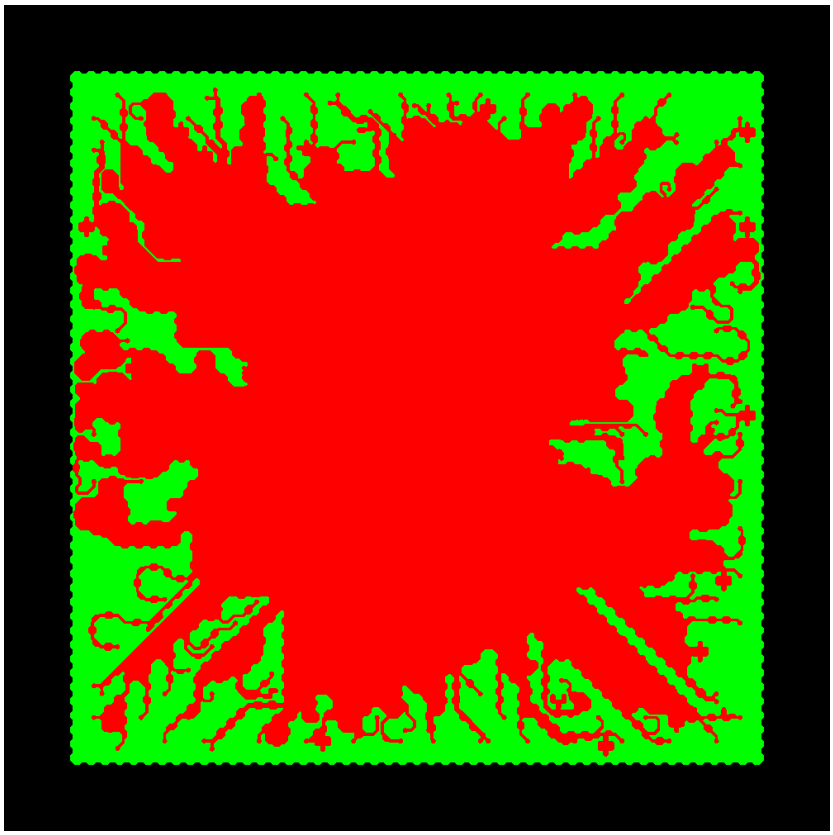
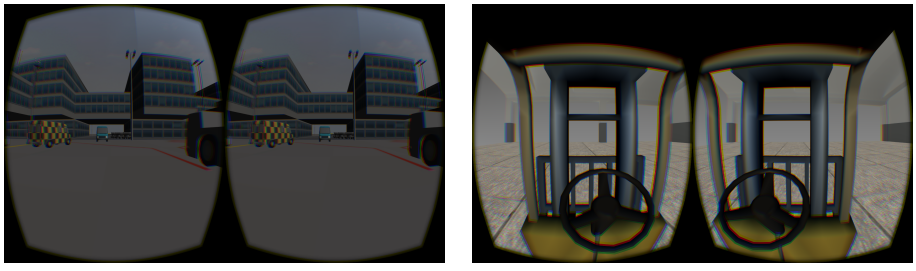


Figure 8: **Software PCB rasterization** done 107 fps on Core i7-4960X.

is used which its elements store the polygon references that overlap them. The point-inside-polygon check is done only for polygons which their references is stored in the grid's element which the pixel resides in.



(a) Airport Scene (b) Forklift Driving

Figure 9: **Virtual Reality** Projects.

**Virtual Reality**. I fixed some bugs in Oculus Rift plug-in of indoor visualization software called Rapid Prototyping Environment (RPE). Additionally, I extended the plug-in for specific projects, to support a player camera inside the scene. The user was able to move through the scene. Furthermore, I extended the plug-in to add leap motion support. The user is able to control the camera with specific gestures (rotating with swiping hand, zooming in/out by points hand inside/outside). Figure 9 shows two screenshots of the VR projects that I did at Fraunhofer IGD.

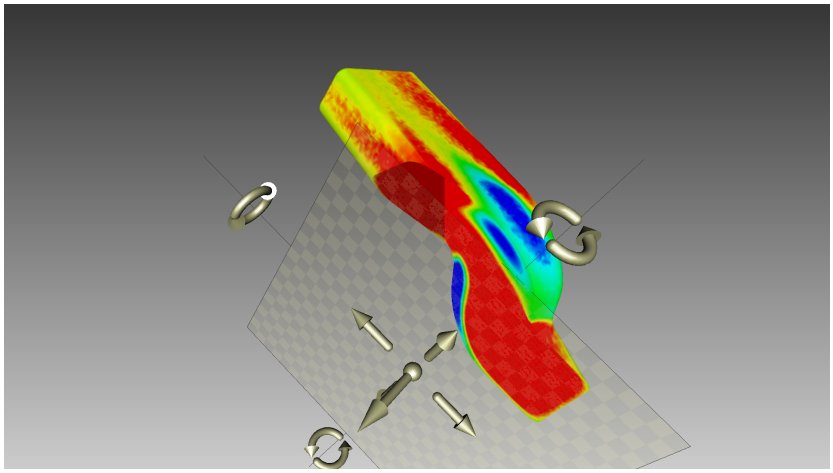


Figure 10: **Cross section** of wind tunnel dataset.

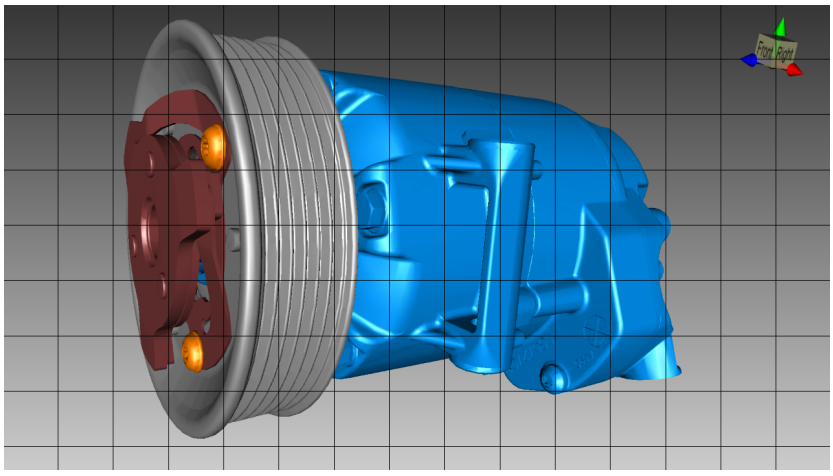


Figure 11: **Extremely high resolution screenshot capturing**.

**Other Tasks**.

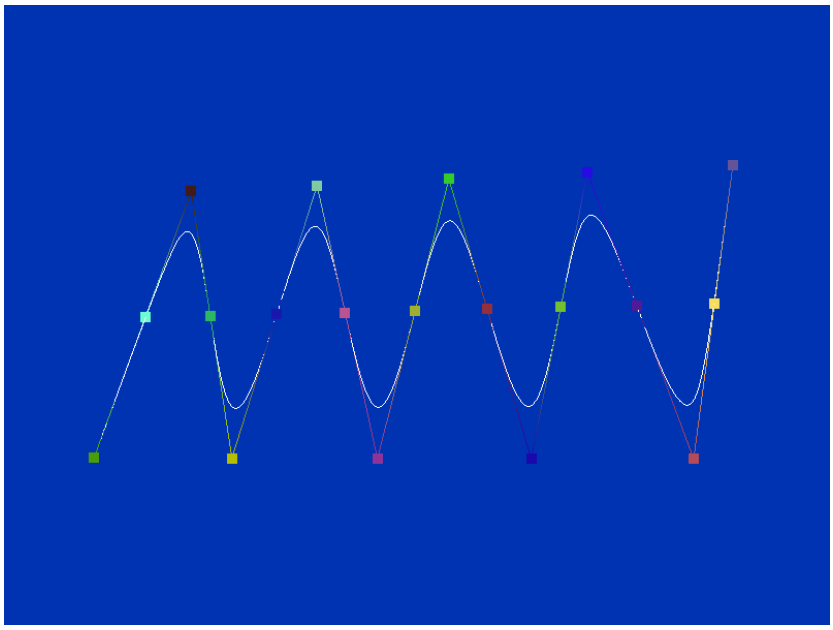


Figure 12: **Evaluating B-Spline on GPU** using a tessellation shader.

- Implementing cross-section algorithm for unstructured simulation mesh types (Figure 10).
- Adding very large screenshot capturing feature to RPE (Figure 11).
- Evaluating B-Spline on the GPU using tessellation shader (Figure 12).