ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL05

NOMAD 3D: AUGMENTED REALITY IN INSTRUMENT CONTROL

Y. Le Goc*, P. Mutti, F. Cécillon, Institut Laue-Langevin, Grenoble, France
I. Dagès, ENSIMAG, Grenoble, France

Abstract

The life cycle of an ILL instrument has two main stages. During the design of the instrument, a precise but static 3D model of the different components is developed. Then comes the exploitation of the instrument of which the control by the Nomad software allows scientific experiments to be performed.

Almost all instruments at the ILL have moveable parts very often hidden behind radiological protection elements such as heavy concrete walls or casemate. Massive elements of the sample environment like magnets and cryostats must be aligned in the beam. All those devices have the possibility to collide with the surrounding environment. To avoid those types of accident, the instrument moves must be checked by a pre-experiment simulation that will reveal possible interferences.

Nomad 3D is the application that links the design and the experiment aspects providing an animated 3D physical representation of the instrument while it moves. Collision detection algorithms will protect the moveable parts from crashes. During an experiment, it will augment the reality by enabling to "see" behind the walls. It will provide as well a precise virtual representation of the instrument during the simulations.

INTRODUCTION

The classical tool for instrument design at the ILL is SolidWorks [1]. Typically, the projects are realised internally but they can integrate some components from external companies. The models at the end may contain errors, have different configurations showing different parts of the model in an exclusive way. The models are very precise — every part of the instrument is designed including the screws — and can be big. The model represents the different components with their real dimensions as the real parts are built from it.

On the other side, Nomad [2], the instrument control software is providing the full control of the instruments and the experiments. The axes driven by the motors can move in parallel on request of the user and Nomad is monitoring the actual positions of the axes by reading the encoder position of the motors. A simple movement of three parts around two axes is shown in Figure 1.

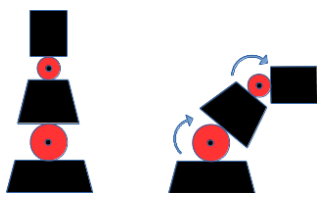

Figure 1: Diagram showing a movement with 2 axes.

* legoc@ill.eu

The goal of the project is to adapt the SolidWorks models to 3D models that will be loaded and animated into a dedicated viewer application. The positions of the axes are read from Nomad. We do not make any strong assumption on the client computer requirements so the viewer application must be scalable and be able to display big original models. Now we suppose that we are in the scope of an instrument for which we have a SolidWorks model and a Nomad configuration. To achieve our goal, we need to proceed in different steps. First we need to export the model to clean, correct and simplify it so that it is small enough to be displayed at a comfortable frame rate. Then we need to identify and map the axes of the model to the axes of Nomad, "augmenting" the data of the model. As Nomad only provides angle or distance values, a "calibration" phase is then required to position the axis in the 3D space and set the "zero". With these information, we are able to animate the 3D model precisely with the only actual values of the axis.

Nomad 3D is a cross-disciplinary project that links Computer-Aided Design (CAD) [3], instrument control and 3D graphics. The article will navigate through these different fields.

WORKFLOW

The Nomad 3D project is split into different applications for which the typical workflow is shown in Figure 2.

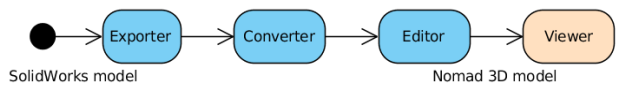

Figure 2: Workflow for an instrument model.

We provide details for each application in the following sections.

SolidWorks Introduction

Let's begin by a short introduction to CAD and the SolidWorks data model. SolidWorks and other CAD software are intended for mechanic's design.

A SolidWorks model is described as a tree hierarchy of components, each of them saved in a separate file. The component leaves also called "parts" are the geometries obtained by a combination of basic 3D geometric shapes - addition or subtraction of prisms, cylinders, spheres, etc. The components that are not the leaves, called "assemblies" are groups of parts or assemblies (called sub-assemblies in that case). They also describe how the sub-components are constrained to each other. A constraint between two components is called a "mate" and defines the degrees of freedom of the components from a relative perspective.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL05

A SolidWorks model can have several configurations describing different positions of the moveable components and their visibility, for which an example is given in Figure 3.

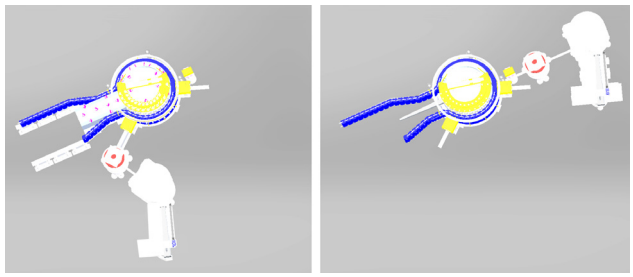

Figure 3: Model in two configurations having different positions and visibilities.

Exporter

The first step in our workflow is to obtain a file data structure adapted to common 3D engines [4], which is usually called scene graph. Indeed if the component hierarchy can be directly translated into our structure, the geometry parts must be converted into a set of triangles. As we were unable to find an open-source library, we tried the MathWorks Simscape Multibody Link [5] add-in which is a SolidWorks plugin to export assemblies to the Simscape Multibody CAD software. The hierarchy and geometries were well exported but not the mates and the configurations. Then we decided to write our own SolidWorks add-in in C# [6] which has proven to be successful. The SolidWorks C# API provides full access to the model data and offers a native function to convert the geometries into STL [7] files containing a set of triangles. Finding such a triangulation function was a key to the success because it was a too difficult task to rewrite it by ourselves. The API enabled us to save all the configurations of the model thanks to a good documentation. Our Nomad 3D add-in has a UI accessible in SolidWorks once a model is loaded and provides the desired export functionality. The first step of simplification of the model geometries is done here by parameterising the export with a threshold enabling to eliminate small parts like screws that we do not need in our final visualisation. When the export is finished, we have the desired file data structure of which a simplified data model is shown in Figure 4. Notice that there are multiple files:

- The main XML file containing the assembly hierarchy, the configurations, the material and visibility properties of each component as well as their mates.
- The STL files containing the geometries of each part.

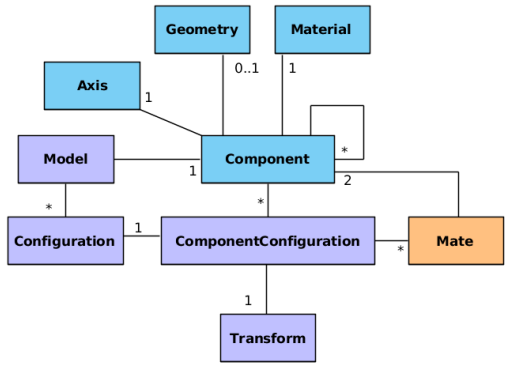

Figure 4: Simplified Nomad 3D common data model.

Note that the triangulation is a discretisation algorithm that we parameterise such that geometries are as coarse as possible, but they are still too detailed for our goal.

Converter

The Converter application is the second step in the clean-up and simplification [8] of the model. We could have extended the SolidWorks C# add-in but we preferred to write it in Java as it was easier to develop. The Converter takes as input the exported files and writes new files. The XML file format is a little bit different. The main difference being that there is no mate group as they have been converted. The STL files are converted using a Blender [9] script to perform a clean-up of the geometries — bad triangle orientation, double vertices, etc. The Blender script is also used to generate different levels of details (LODs) [10] for the geometries. In addition to the cleaned-up *exported* geometries, we generate the *decimated* geometries for which a ratio is provided by the user (0.5 is the default value) and the *convex hull* geometries. These different levels of details will be useful for the scalability of the viewer application as the number of triangles is decreasing according to them. An example of LODs is given in Figures 5 and 6.

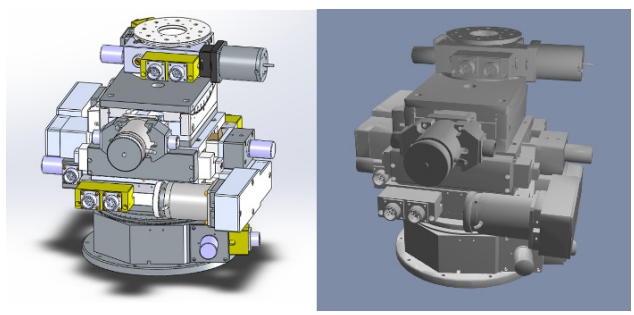

Figure 5: A SolidWorks goniometer model and its *exported* conversion.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL05

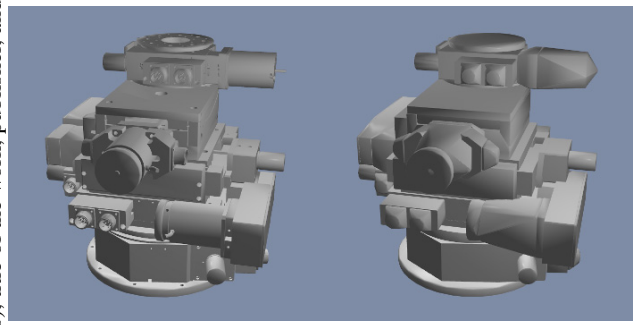

Figure 6: The *decimated* and *convex hull* conversions.

Editor

If the Exporter and the Converter applications reduce and reorganize data of the model, the Editor application on the other hand is adding information that cannot be done automatically.

The main goal of the Editor is to define the axes and the walls of the model. To do it, we must first find the connected components that move around axis. For that it is possible to do some kind of clean-up by reorganizing the model if its component hierarchy and geometry are not suitable. The Editor allows to:

- Edit the tree: add, remove, move some components.
- Subdivide some geometries: the operation must be done in an external tool, e.g., Blender.

The subdivision of geometries can be necessary when we integrate parts designed by an external company.

Then we can focus on the definition of the axes, that is essential for the animation of the model with Nomad. The Editor has different steps to add an axis:

- Define the space properties: an axis is associated to a component and describes how this component can move relatively to its parent component. The user must enter its type — rotation or translation — and define its 3D position — point and direction.
- Map to Nomad: the user must select an axis from the list provided by Nomad.
- Calibrate: the definition of the “zero” or “median” position is necessary to correctly animate the model according to the Nomad values.
- Optional limits: some limits can be added here but they are only used for visualisation as Nomad already have some.

The Editor also provides the necessary identification of the walls of the models by tagging the wanted components.

From the user point of view, the Editor written in JavaFX [11] provides a 3D view of the model and a tree view of the hierarchy of components. To interact with the different components, the user can hide or show any component, i.e. all its children and select them, which is necessary to access the “deepest” parts of the model. The selection can be done graphically in the 3D view as well as in the tree view by selecting items. Once a component is selected, its associated data can be edited, i.e., the axis, material and wall properties as shown in Figure 7.

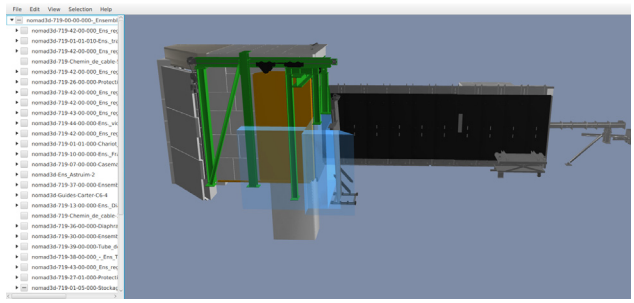

Figure 7: Screenshot of the Editor. The front wall is selected and appears transparent.

The Editor also provides features to edit the configurations inherited by the SolidWorks model. SolidWorks does not constraint the number and the content of the configurations which offers lots of freedom to the CAD designer. In our case, our Nomad 3D standard imposes to have one mandatory configuration called “median” in which all the axes are in the "median" position, i.e. the zero value in Nomad, and the components are visible. This special configuration is mandatory to correctly animate the model and the user must edit one existing configuration to achieve this.

Depending on the original SolidWorks model “qualities”, the operations in the Editor to generate a standard Nomad 3D model for the viewer are not trivial and may take time.

Viewer

We rapidly decided to choose Three.js [12] as frontend for rendering and animating our Nomad 3D models. Three.js is a 3D engine based on WebGL [13] – implementation of OpenGL ES [14] in JavaScript [15] - which offers the current best compromise between performance and portability. Three.js also has a nice API and lots of functionalities including a performant management of the LODs: it selects automatically the appropriate LOD for a given geometry or part depending on its distance from the point of view as shown in Figure 8. A “view distance” parameter controls how close from the point of view the different LODs are selected. The more the distance is small, the more the displayed geometry will be coarse. Our Nomad 3D model has been designed to be easily loadable, rendered and animated in Three.js. The resulting code is simple and easy to extend.

The user can change some display options: the opacity of the walls, the material reflection and the dynamic shadows can be tuned depending on the performance of the client computer and adapted to a comfortable framerate. As backend we use Node.js [16] for which a V8 C++ addon has been written to communicate with Nomad. The Viewer is requesting to Nomad the positions of the axes at each frame enabling a real-time animation of the model.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL05

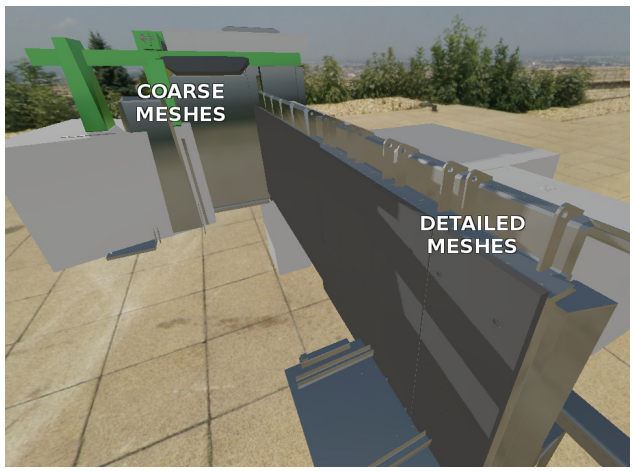

Figure 8: Screenshot of the Viewer showing the LODs depending on the distance from the point of view.

Now we have developed a cross-platform Electron [17] application, which is a desktop application (Electron embeds Chromium and Node.js without any remote requests). The choice of Three.js offers a lot of flexibility that will enable us to share the client code for further web, tablet applications.

RESULTS

We tested the Nomad 3D workflow on the model of the vertical Time-of-Flight reflectometer Figaro for which the SolidWorks model has a size of 1.1 G on disk. The conversion to the Nomad 3D model resulted in a size of 103M on disk. The triangle count of the different LODs is given in Table 1 and an overview of the model in Figures 9 and 10.

Table 1: The Level of Details of the Figaro Model

Level of details	Triangle count
Cleaned	13 769 502
Decimated	6 887 424
Convex hulls	399 414

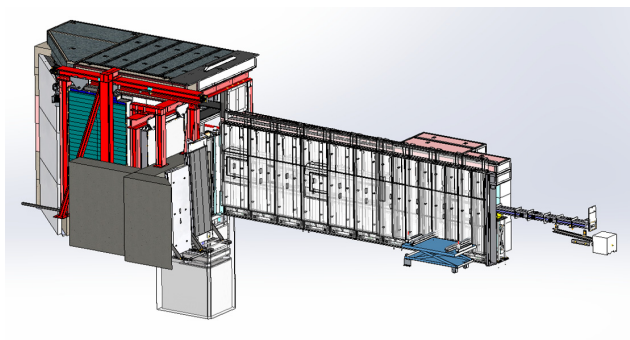

Figure 9: Overview of the Figaro instrument in SolidWorks.

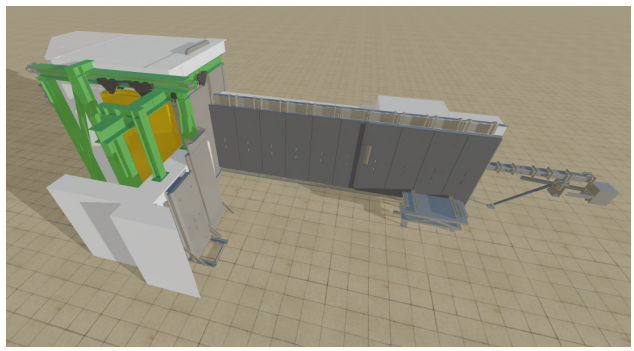

Figure 10: Overview of the Figaro instrument in the Viewer.

We developed successfully the Viewer application that enables us to animate the model in real-time by Nomad and see behind the walls. Some demonstration videos of the Figaro instrument are available on the Nomad 3D web site [18]. We obtained a 30fps frame rate on a desktop PC (dual core Intel i5 3.3GHz, Nvidia GT218 512M) for the Figaro model. However at the time we write, the collision detection and the set-up of external devices like cryostats is still ongoing.

The commissioning phase of Nomad 3D on Figaro is now completed and we are working to extend the application to other instruments and to get some feedback from the users.

FUTURE DEVELOPMENTS

Currently on some instruments of the ILL, we use a rough mechanism to check the potential collisions before moving the parts for real: we only check the final positions of the movement. We want to replace this with a more precise simulation of the movement which takes into account the full trajectory.

The implementation of the collision detection [19] will be soon implemented in Nomad 3D and will allow to solve this: we have to check if wall geometries collide with non-wall geometries during the movement. However this is not a so trivial task as different strategies may be adopted depending on the level of precision we want to reach and the computer resources we have.

Some new clients will also be developed: a web application for the ubiquity and a tablet application for a better user interaction.

These are the incoming developments but some others will follow as the Nomad 3D concept opens new perspectives.

CONCLUSION

We achieved our goal to develop tools to “see” behind the walls of the instruments without any webcam providing the first step to the Augmented Reality at the ILL.

Concretely we can take an ILL SolidWorks model of an instrument, adapt it to a standard Nomad 3D model without any modification in SolidWorks and animate it in a simple viewer. We developed successfully the Exporter,

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-THAPL05

Converter, Editor and Viewer applications. We needed to add some functionalities to the Editor as we were "exploring" the full Figaro instrument's model. We surely will need to add some new functionalities to cover all the ILL moving instruments for which a SolidWorks model exists, but definitely the way is open to the next generation of instrument control where the user experience is at the heart. Nomad 3D mixes mechanics, CAD, 3D graphics and shows that cross-disciplinary projects are the future of instrument control. Better instrument simulation with the online collision detection will allow to optimise the space and by consequence measures. Last but not least, the integration of Virtual Reality devices such as Oculus Rift for a complete immersion will come up soon.

REFERENCES

[1] SolidWorks, http://www.solidworks.com/

[2] Nomad, P. Mutti *et al.*, "NOMAD - More than a simple sequencer'', in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, pp. 808-811.

[3] *CAD: Handbook of Computer Aided Geometric Design*, Gerald Farin, Josef Hoschek, Myung-Soo Kim, Josef Hoschek and Myung-Soo Kim, ISBN: 978-0-444-51104-1

[4] 3D Graphics Introduction, https://www.gamedev.net/articles/programming/graphics/the-total-beginners-guide-to-3d-graphics-theory-r3402/

[5] MathWorks Simscape Multibody Link, https://fr.mathworks.com/help/physmod/smlink/index.html

[6] C#, https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/index/

[7] STL format, https://all3dp.com/what-is-stl-file-format-extension-3d-printing/

[8] 3D mesh simplification, https://software.intel.com/en-us/articles/3d-modeling-and-parallel-mesh-simplification/

[9] Blender, https://www.blender.org/

[10] LOD, http://computer-graphics.se/TSBK07-files/pdf/PDF09/10%20LOD.pdf

[11] JavaFX, https://docs.oracle.com/javase/8/javase-clienttechnologies.htm/

[12] Three.js, https://threejs.org/

[13] WebGL, https://www.khronos.org/webgl/

[14] OpenGL ES, https://www.khronos.org/opengles/

[15] JavaScript, https://javascript.info/

[16] Node.js, https://nodejs.org/

[17] Electron, https://electron.atom.io/

[18] Nomad 3D, http://docs.sites.code.ill.fr/nomad-3d/

[19] Collision Detection, https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection/