Immersive Visualization of Rail Simulation Data

A Master of Science Project Report

Submitted by,

Vaibhav Govilkar

UIN:678491876

Advisor: Jason Leigh

Secondary committee member: Andrew Johnson

Abstract

The prime objective of this project is to create scientific, immersive visualizations of a Rail-simulation. This project is a part of a larger initiative that consists of three distinct parts. The first step consists of performing a finite-element analysis to construct 'mode-shapes' consisting of a railway track and its substructure[5]. The second step involves running a simulation using these 'mode-shapes' to analyze the effects of a locomotive passive over this sub-structure. The third step is involves creating an 'Immersive Visualization of the Rail Simulation Data' using the CAVE2™[10], the central scope of this project.

Glossary

CAVE2™

The CAVE2™, the next-generation large-scale virtual-reality environment, is a hybrid system that combines the benefits of both scalable-resolution display walls and virtual-reality systems to create a seamless 3D environment that supports both information-rich analysis as well as virtual-reality simulation exploration at a resolution matching human visual acuity[10].

Mode-shapes

Mode-shapes are a result of 'finite element analysis' of objects. These mode-shapes describe the deformation of the object based on excitation by specific frequencies. This data is used to run the simulation. Apart from visualizing the simulation itself, the mode-shapes that constitute the simulation input are also useful to a mechanical engineer as visualizing them allows the engineer to assess the role of each individual mode in the final simulation.

OmegaLib

OmegaLib is a middleware designed to ease the development of applications on virtual reality and immersive systems[6]. An open-source library that is used in this project to implement the scientific visualizations in the CAVE2™[10]. It is an extension of the OpenSceneGraph library customized for large virtual reality systems like the CAVE2™[6]. It allows the programmer to freely make OpenSceneGraph calls along with OmegaLib functionalities.

OpenSceneGraph

OpenSceneGraph is a 3D graphics application programming interface used in visualization, virtual reality, simulation and computer games. Its architecture is based around the scene graph data structure where every object in the 3D scene is treated as a node in a graph and the effects of modification of a single node propagate throughout the graph[1].

Shader

A shader is a program or a code fragment that run the Graphics Processing Unit(GPU) and renders a scene based on lighting and coloring models[3].

Table of Contents

Introduction

The first two sections of the RailSim initiative consist of work done by students of the mechanical department. The visualization itself was handled by me as a from the Electronic Visualization Lab in the cross functional team(2 mechanical engineering grad students+1 Computer Science grad student) involved in the RailSim initiative. A large part of the project involved converting large data-sets into scientific visualizations that a mechanical engineer may observe, assess, analyze in a meaningful way with the goal of drawing conclusions aided by a superior understanding of how the objects in the simulation interact.

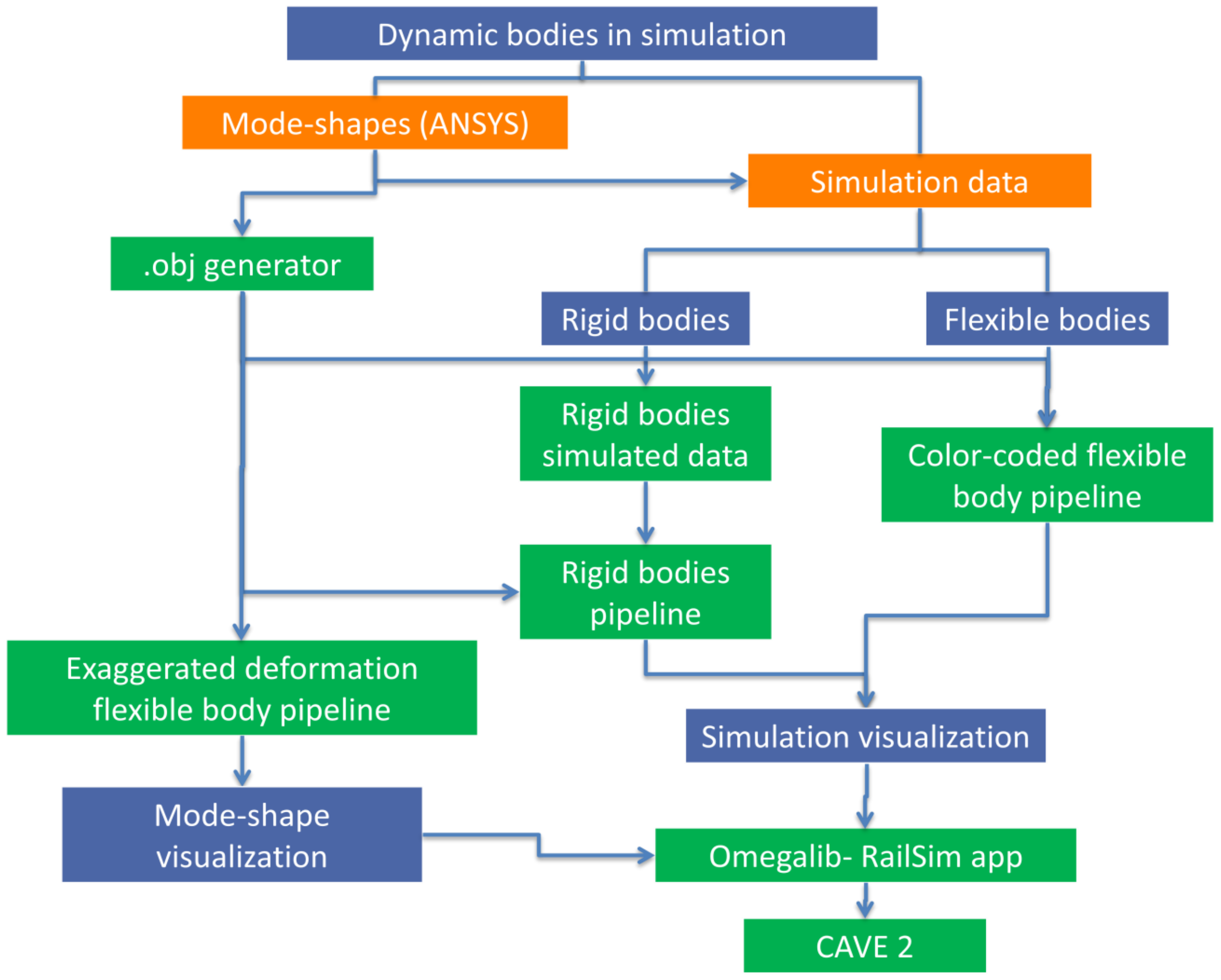
The simulation itself is run on 'mode-shapes' constructed in the finite element analysis tool 'ANSYS'. It consists of the following major bodies[5]:

- Rigid rails
- Deformable rail
- Rigid sleepers
- Ballast (Upper gravel)
- Sub-ballast(Middle soil)
- Sub-grade(Lower soil)
- Fasteners(Spring elements)
- Suspended wheel-set (representing a loaded locomotive)

The data from 'ANSYS' is used as an input to a mechanical simulation software SAMs to run 7 to 10 second simulations. These simulations in turn produced new simulation data-sets.

The data-sets from ANSYS and the simulation were used to create visualization. They are put through several stages of pre-processing before being passed into the CAVE2™[10] application.

Work-flow



Where:

🟩 Project work

Green: Work done as a part of the visualization project.

🟧 External work

Orange: Tasks completed externally primarily by the mechanical engineers to provide the input data for the visualization.

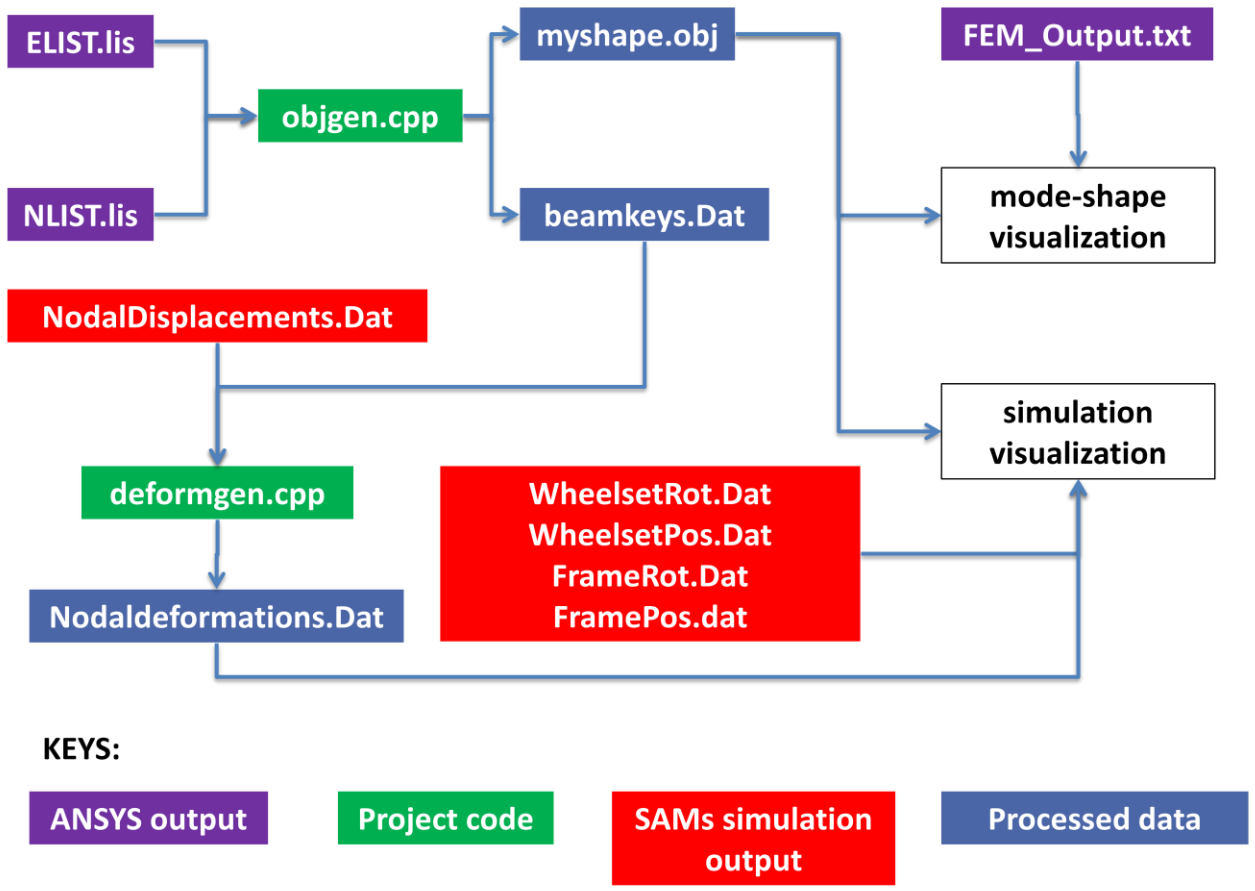
Description of the work flow

- Mode-shapes are generated in 'ANSYS' and its data is passed on to SAMs to run simulations.
- The ANSYS data is passed into the .obj generator to create 3d models in the form of .obj files[9].
- The .obj files in conjunction with the 'ANSYS' mode-shape data is passed to the 'Exaggerated deformation flexible body pipeline' which will be used in the Mode-shape visualization.
- The .obj files in conjunction with the SAMs simulation data is passed to the rigid body, and color-coded flexible body pipeline to achieve a visualization of the simulation itself[8][9].
- The OmegaLib RailSim app for the CAVE2™ handles these two visualizations to be displayed on the CAVE2™.

Code Base

The code base to achieve the given workflow consists of the following:

Pre-processing code



- **NLIST.lis**
  This file contains the list of nodes that the mode-shape is comprised of in the form : *Node number, x, y, z coordinate of that node (space separated)*.
- **ELIST.lis**
  This is file that contains element definitions as a part of the finite element analysis. These consist of the definitions of each element as a cuboid structure of six nodes or vertices or 2 vertex points for beam elements. It exists in the following form:
  Element number, node numbers the element is made up of.

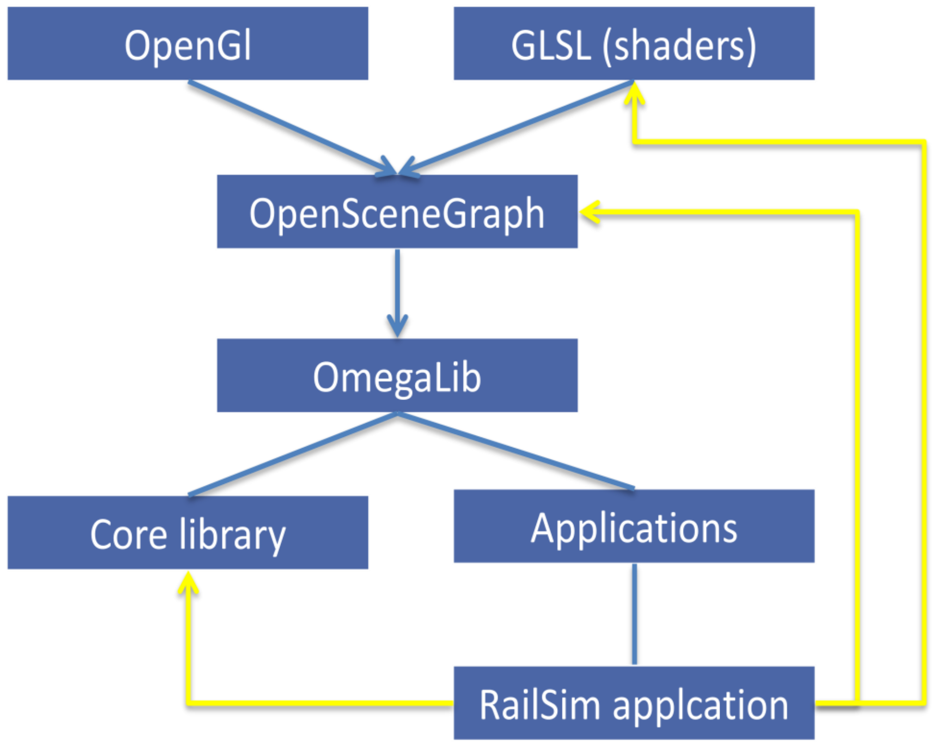
- **objgen.cpp**
  This file is used to generate the mode-shape itself. The mode-shape consists the railway track and the substructure below it. It uses the two files – NLIST.lis, ELIST.lis the .obj file[9]. It also constructs an intermediate file 'beamkeys.Dat'. This files is used to generate deformation data for extra vertices added during the construction of the rail and sleeper cross-sections[7][8].
- **myshape.obj**
  This is the .obj file constructed[9] by the objgen. This file is a universal format for defining 3d objects. It is in the format:
  To define vertices: ‘v’, x, y, z coordinate(space separated)[9]
  To define surfaces: ‘f’, all the vertices the face is comprised of(space separated)[9].
- **beamkeys.Dat**
  This is an intermediate file used by the deformgen.cpp, specifically to generate beam-element cross sections. Since the original deformations received contain only deformations for 2 end-points of a beam this file is used to replicate the deformation information for newly added vertices as a part of the cross-section.
- **NodalDisplacements.Dat**
  This file is generated by the SAMs simulation and contains a list of deformations in 3 point vector form for every vertex in the same order as the original node-list file.
- **deformgen.cpp**
  This file is used to generate a file called 'Nodaldeformations.Dat' using the output from the SAMs simulation. This file uses 3d vector deformations of every node to generate a file of the net magnitudes[7][8]. This output is finally passed into the color coded flexible body pipeline to visualize the simulation. The intermediate data file 'beamkeys.Dat' is used to add deformation values for the newly generated vertices in the .obj files[9] which are a part of the sleeper and beam cross-sections.

- **Nodaldeformations.Dat**
  This file is generated by the deformgen.cpp program easy to parse for OmegaLib and it also contains the extra nodes added to create the rail and the sleeper cross-sections for the beam-elements.
- **FEM_Output.txt**
  This file is generated by ANSYS and contains the deformation values for every possible mode in the finite element analysis.
- **Wheel-set data**
  These files consist of the wheel-position and rotation for the frame and the wheels as they move along in the simulation
- The **mode-shape visualization** uses the .obj file and the 'FEM_Output' or the Finite Element Mode output from ANSYS for its pipeline[6] as inputs.
- The **simulation visualization** uses the .obj files. The suspended wheelset/frame's rotation and position generated by SAM along with the processed file 'Nodaldeformations.Dat' as inputs to its pipeline[6].

The OmegaLib app

- The RailSim application uses the OmegaLib library along with several OpenSceneGraph calls[2] to visualize the mode-shape as well as the simulation.
- The app codebase also include custom shaders to manipulate attributes of individual vertices within the mode-shape.

The application makes calls from several sources, best described in the following chart.



The yellow lines represent the sources from which the application derives its code.

- The OmegaLib library is built upon OpenSceneGraph which in turn is written in C++ using OpenGl[6]. OpenSceneGraph also allows direct access to shaders through several of its inbuilt function calls[1].
- The OmegaLib library as it currently exists is divided into the Core Library which contains all the functionalities, and the applications library which contains applications that use the core library[6].

- OmegaLib allows the application developer to make direct OpenScenGraph calls if required.
- The RailSim application(contained in the applications library) uses primarily the core library code. But also uses several OpenSceneGraph functionalities, as it requires the manipulation of geometries.
- The manipulation of geometries requires altering specific vertex attributes.
- Hence the application uses custom shaders that allow vertex manipulation written in GLSL[4] [3] that receive input from the OpenSceneGraph calls. [1] [2]

The application itself consists of the following prime classes and functions over and above existing OmegaLib and OpenSceneGraph calls:

- **Findgeometry class**
  This class uses the osg::NodeVisitior [2] within its constructor to find the geometry node within the current node in the OpenSceneGraph node hierarchy[1].
- **objloader class**
  This class uses the .obj file data to load a geometry into the scene and add it to the scene node using OpenSceneGraph geometry construction[1] and manipulation calls.
- **OmegaViewer**
  This is a class defined in every OmegaLib application and its instance is created when the application is launched[6]. It contains most of the application specific code. It contains the following methods:
  - **initializeData()**
    This method is used to parse load all the pre-processed data from the simulation and ANSYS into the application. It makes calls to each file's loader functions with specific parameters and populates local variables.
  - **loadobj(..)**
    This function takes the file path and the references of the two vectors to populate. The function parses the files and loads the file data into the two vectors used for geometry construction[1] by the

objloader class. These vectors are the vertex list and the surface list.

- **loaddeform(..)**
  This function takes the filepath and a reference the deformation data vector. It parses the file and populates the vector[8] with the loaded data.
- **loadData(..)**
  This function parses the wheel-set information from the files and populates the wheelset vectors which include the position and rotation for the wheels and the frame.
- **loadcolors(..)**
  This function loads the specific colors-per-material based on a file that contains each nodes material type which include every element possible in the ANSYS output including the rails, sleepers, ballast, sub-ballast and sub-grade. This is mainly useful for the non-color-coded mode-shape visualization.
- **initialize()**
  This function is used to initialize the members of this class. The constructor is not used as all the data is unavailable at the time of creation of OmegaViewer. It makes calls to various initializers including the critical initializeData() function. It sets up the lighting, the skybox and the camera position. It also loads the custom shaders that the application uses.
- **onMenuItemEvent(menuitem)**
  This is an enforced virtual function[6][7] definition that is automatically executed everytime there is any kind of menu-event including selection and deselsction of menu-items. The function provides definition for what is to be executed when a menu even occurs.
- **handleEvent(event)**
  Another even handler called when any button on the controller is pressed. This handles the navigation, key-framing and camera-view switching[6].

- **update(..)[6]**
  This function is called at every frame and is used to update the geometry of the mode-shape as well as the position of the wheel-set. Depending upon the visualization currently being viewed the update function passed application data, like deformation values to the vertex shader so that it may update the geometry accordingly.
- **Slerp, Lerp, toquaternion**
  These three methods allow for spherical interpolation, linear interpolation and a converter to quaternion. They can be used to perform smooth camera transitions.
- **camRot and camTrans**
  These methods allow the user to implement camera rotation and camera translations to the main scene camera.

Custom Shaders

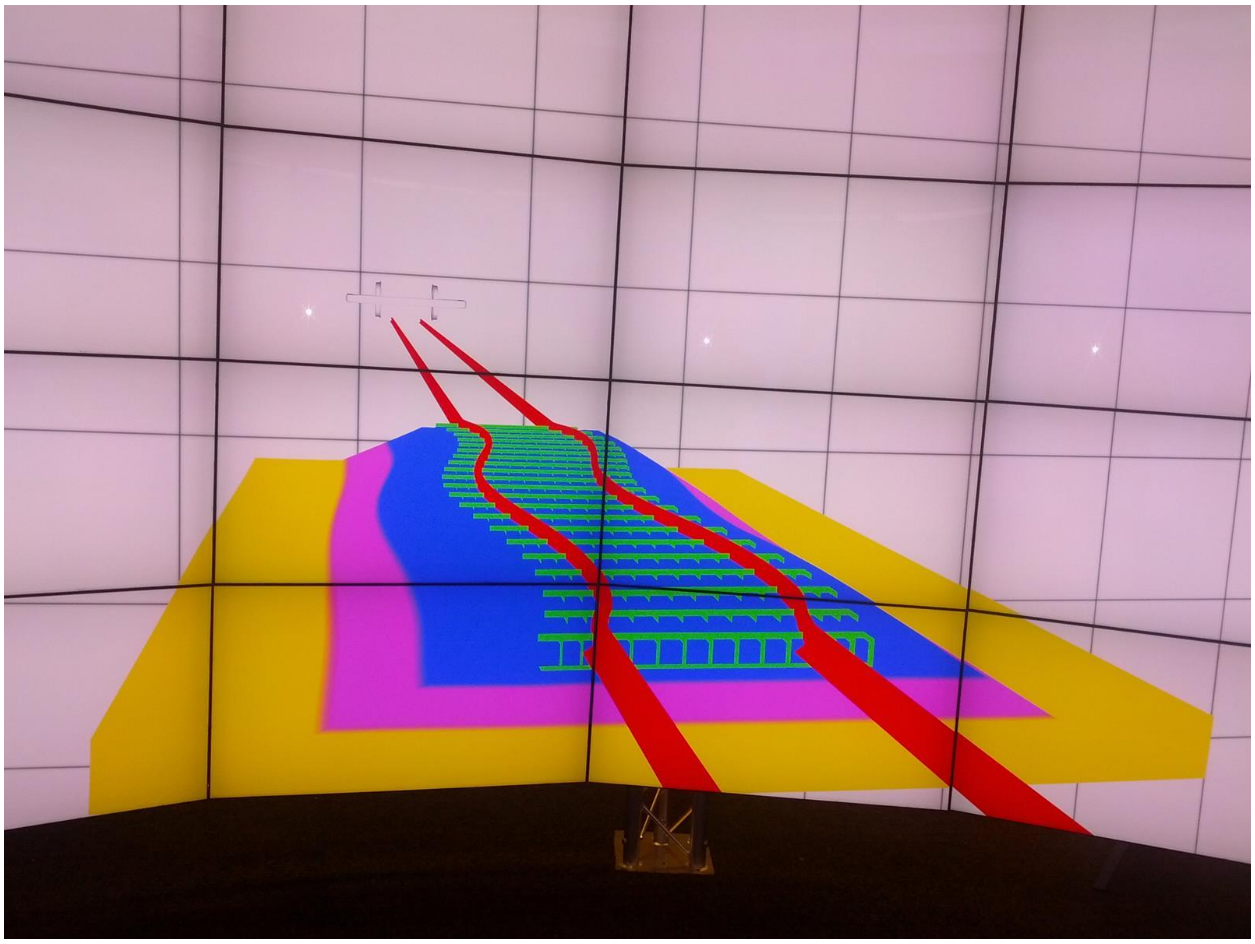
The application uses three custom shaders to create the visualizations:

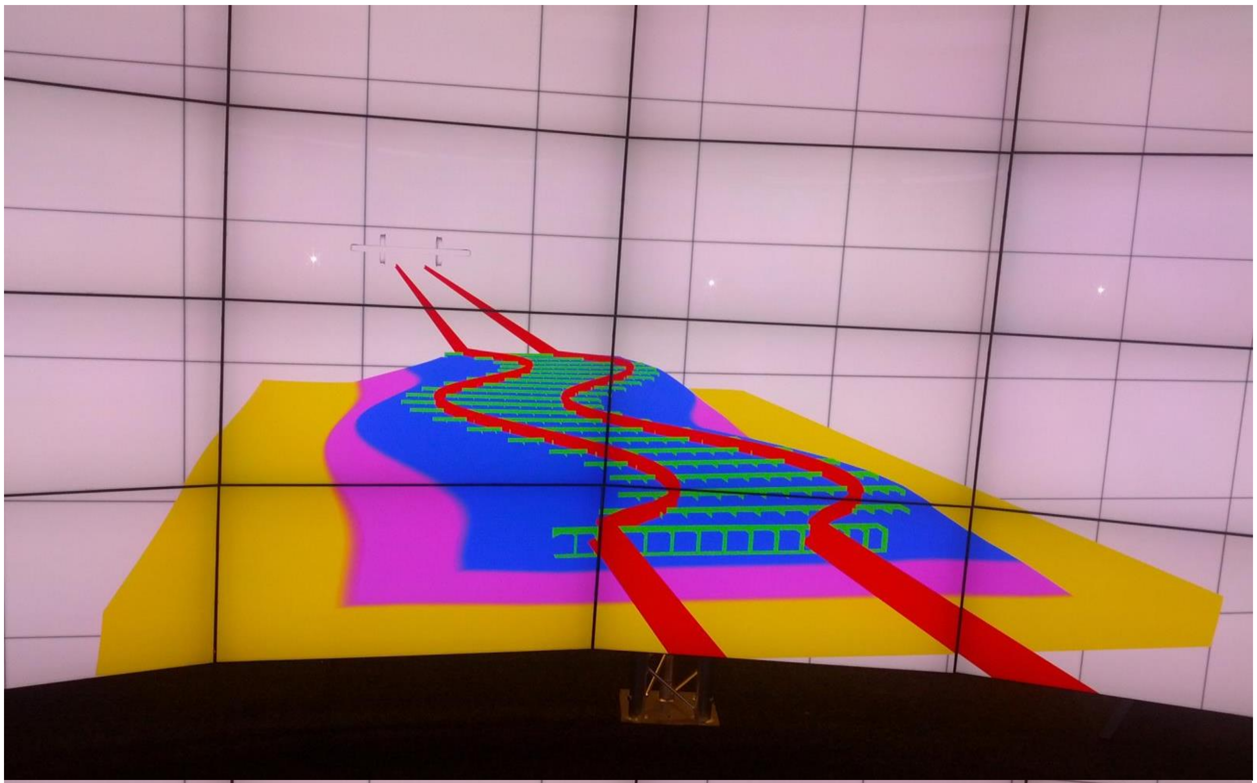
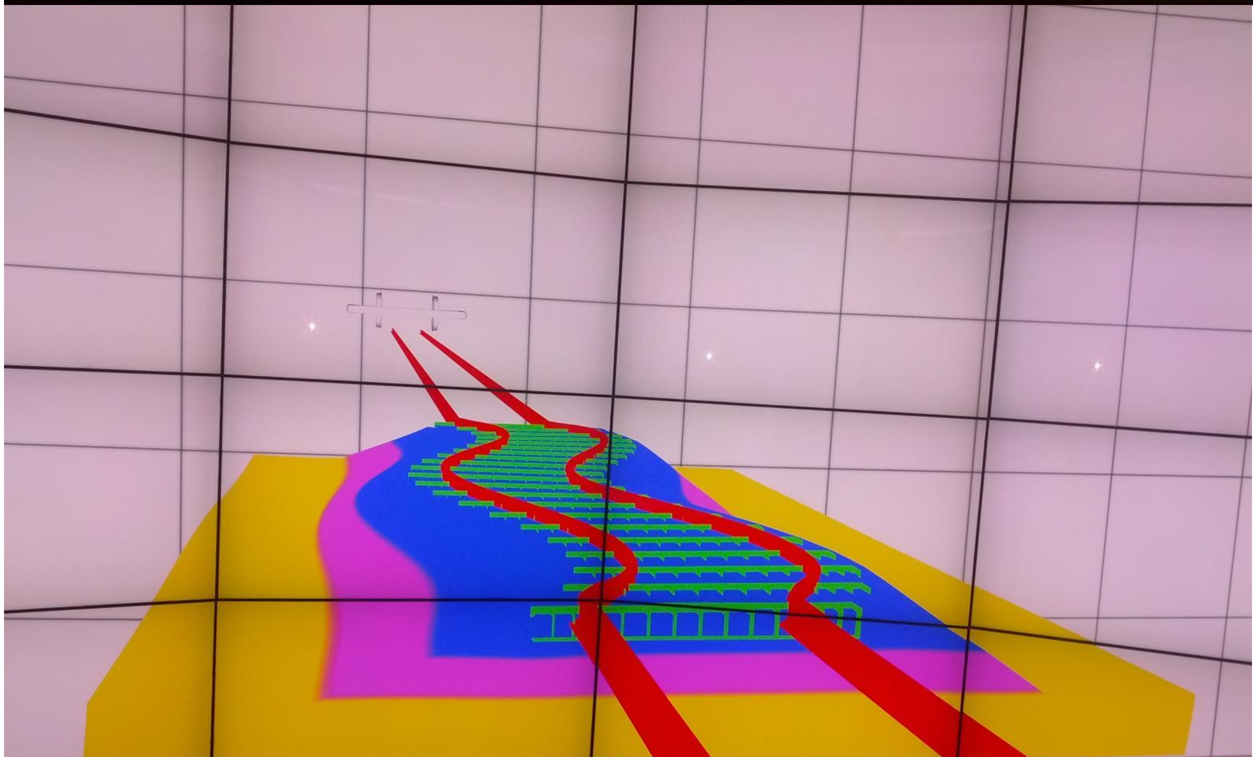
- **Textured.vert (simulation vertex shader)**
  This shader takes the deformation data-set and creates a color-code per-vertex depending on the deformation values. Each vertex is colored using this shader and its values and interpolated between vertices using the fragment shader[3][4].
- **Texturedmode.vert (modeshapes vertex shader)** This shader takes the deformation data-set for each mode and exaggerates it. This exaggerated displacement is added to every vertex in the geometry[3][4].
- **Textured-border.frag (fragment shader)** [3]
  This shader is used by both the vertex shaders to render the geometry. This shader remains common to both the modes of visualization since most of the code remained same in both the modes. The fragment shader interpolates between vertices to generate a surface of the geometry[3]. The interpolation includes the positions and the color of the vertices in question.

Mode-shape visualization

The mode-shape visualization uses the 'Exaggerated deformation flexible body pipeline'**.** After the application loads the .obj file of the mode-shape the pipe-line passes the deformations-per-vertex for the mode-shapes into the vertex shader[3] and displaces them with a multiplicative magnifying factor that increases from 0 to 150 over time and then loops back to 0. This distorts the mode-shape in an exaggerated fashion to describe the current 'mode' selected.

Following are a few images of a mode-shape with varying degrees of exaggeration in its deformation:

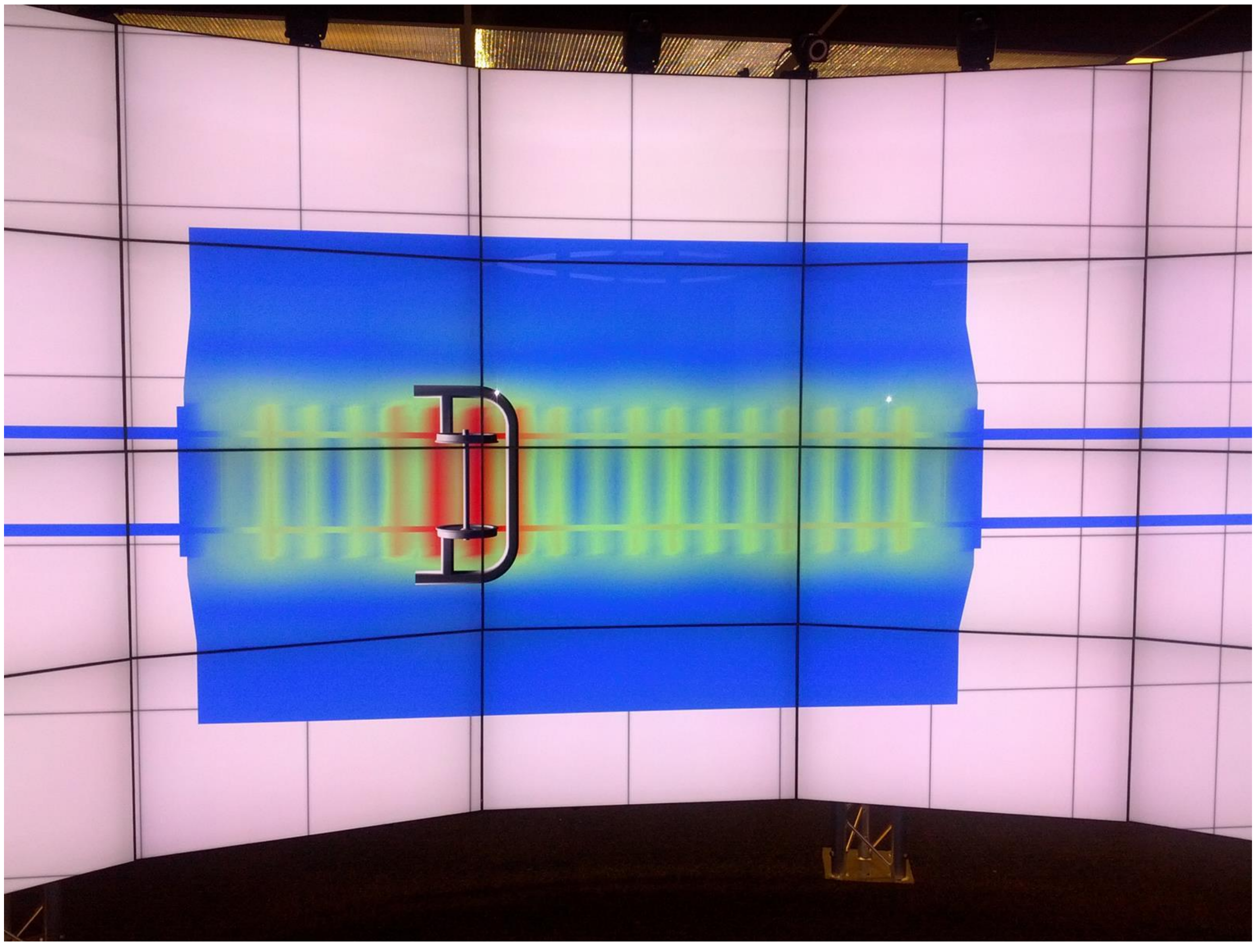


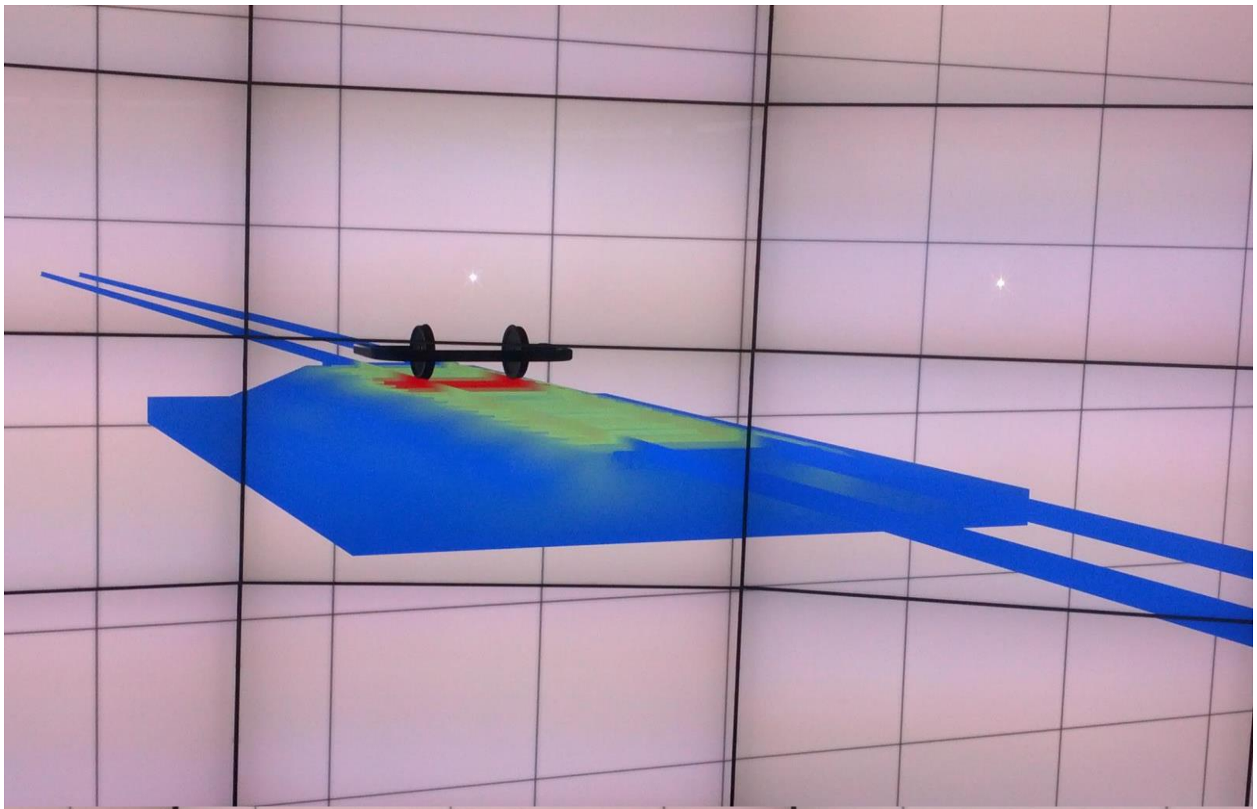



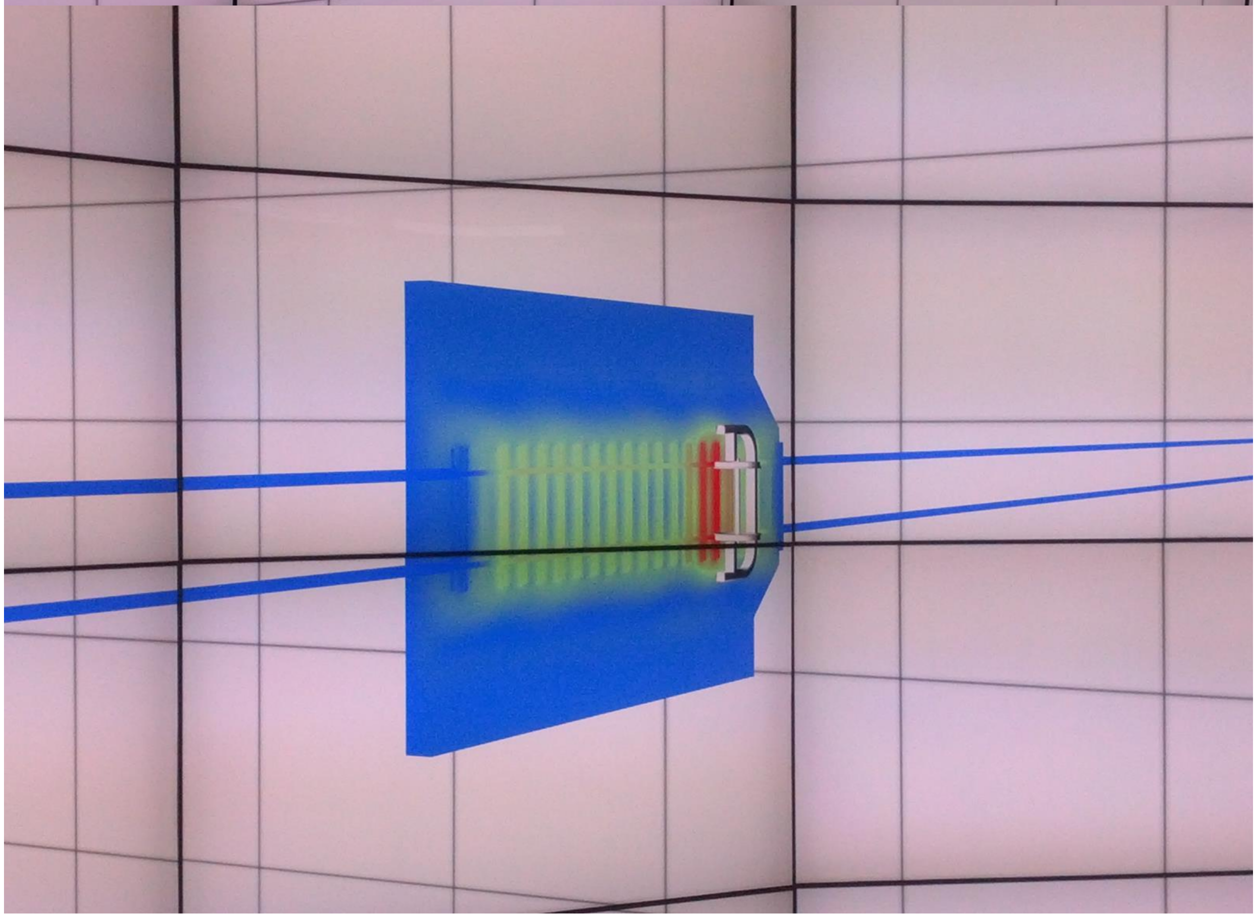
Simulation visualization

The simulation visualization uses the 'Color-coded flexible body pipeline'. The application loads the .obj file of the object(same as the mode-shape visualization). It also loads the wheel-set positions and rotations per-time-step and the nodal-deformations of the object(rails and substructure) as the wheels pass over it from 'Nodaldeformation.Dat'. The per-vertex deformations of the object as time progresses are encoded in the form of per-vertex coloring. The colors vary from blue(for static nodes) to red(maximum deformation). This spectrum allows us to see the effects of the loaded-wheelset on the rails and the sub-structure as it passes over-head.

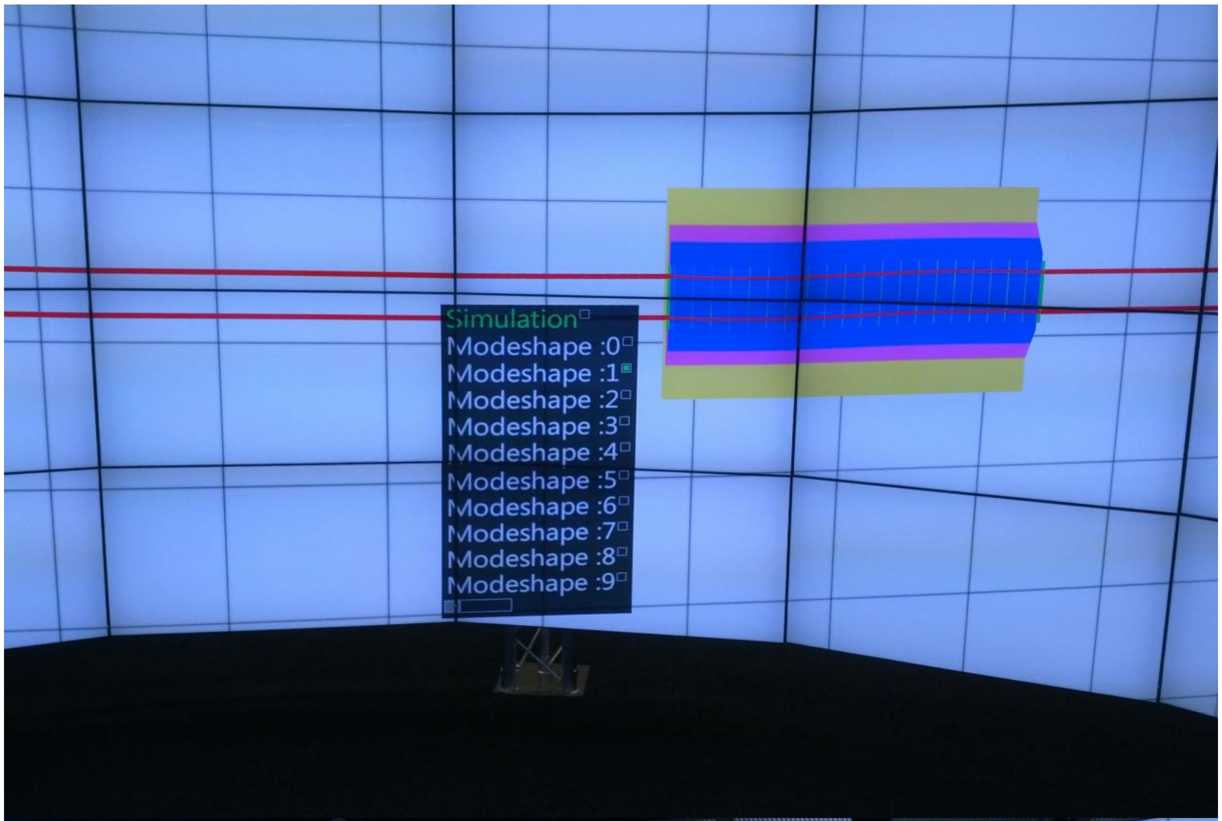
Following are a few images of the simulation from various perspectives. As can be seen the region on the rail-substructure in contact and closest to the wheel-set shows maximum deformation signified by the red color.

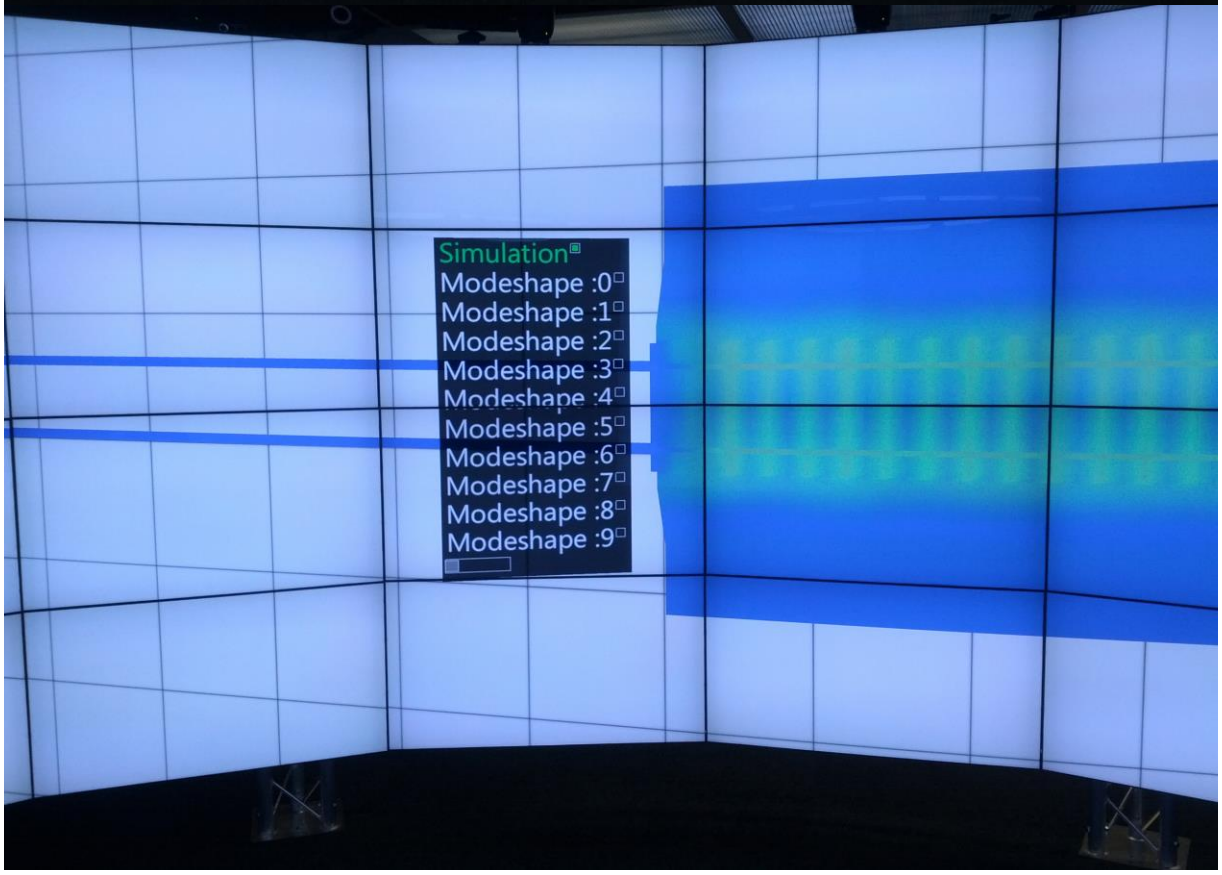






Following are a few screenshots of the menu system[6]:

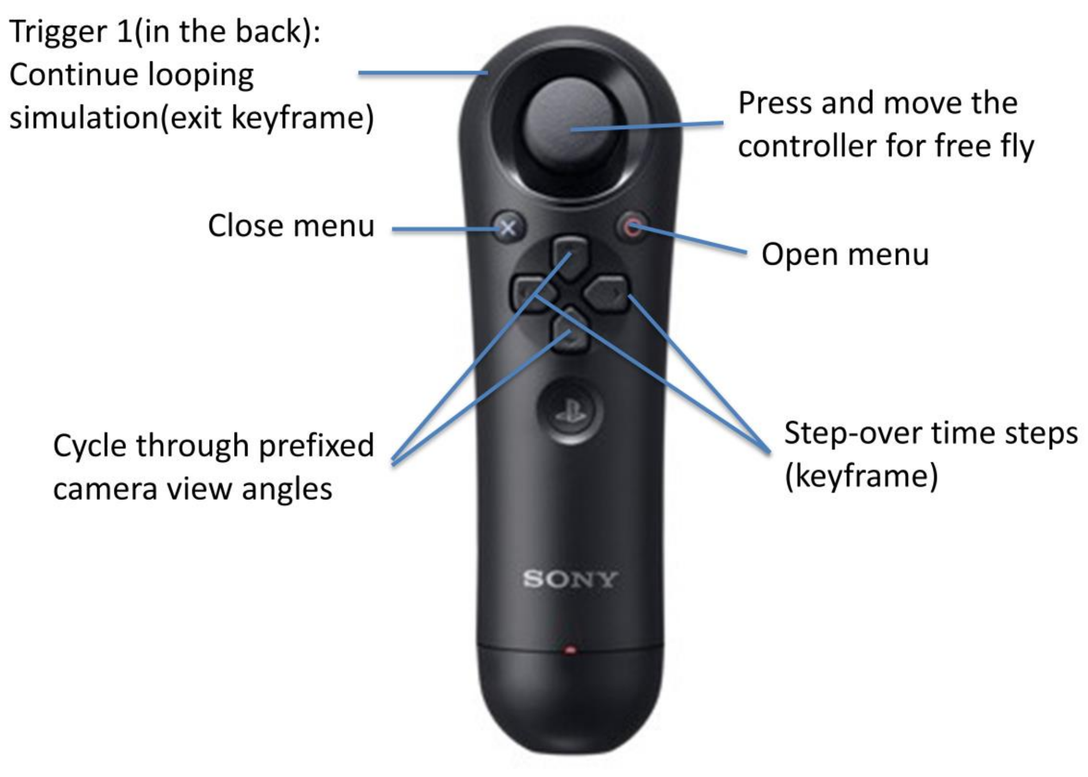




The application also includes the following features:

- Two modes: Simulation mode, mode-shape mode.
- Fully dynamic camera with free flying as well as smooth transitioning to pre-fixed positions[6]
- Pause, Play, Step-over individual time-steps.
- An interactive menu-system[6].

The playstation-move controller is used to control the simulation. Each button can be custom-mapped to a required functionality[6]. Currently the controls are as follows:



Conclusion

The project code has been documented to allow future students to build upon the existing code base as the larger RailSim initiative grows in scope. The existing code can be used for new simulation data as the data-form remains consistent.

References

[1] Rui Wang **and** Xuelei Qian**,** 2010, *OpenSceneGraph 3.0: Beginner's Guide,* Packt Publishing

[2] OpenSceneGraph library, Documentation: http://trac.openscenegraph.org/documentation [Online; accessed 2012-2013]

[3] Randi J. Rost, Bill M. Licea-Kane, Dan Ginsburg, John M. Kessenich, Barthold Lichtenbelt, Hugh Malan, Mike Weiblen, 2009, *OpenGL Shading Language (3rd Edition),* Pearson Education

[4] GLSL OpenGL shading language, Online Reference: http://www.opengl.org/sdk/docs/manglsl/ [Online; accessed 2012-2013]

[5] John. H. Armstrong, 2008, *The Railroad What it is, What it Does, 5 edition,* Simmons Boardman Pub Co

[6] OmegaLib, Documentation: https://code.google.com/p/omegalib/ [Online; accessed 2013]

[7] Stanley.B. Lipman, 1999, *Essential C++ edition 1,* Addison-Wesley Professional

[8] C++/ STL, Online Reference, Documentation: www.cplusplus.com/reference/stl [Online; accessed 2012-2013]

[9] Paul Bourke, Online Reference, *Object files(.obj),* http://paulbourke.net/dataformats/obj/ [Online; accessed July 2012]

[10] Electronic Visualization Laboritory, UIC, Website, http://www.evl.uic.edu/ [Online; accessed March 2013]