Software application for verification of equipment access

Christian Tverås

May 2016

Master Thesis

Department of Engineering Design and Materials

Norwegian University of Science and Technology

Supervisor: Professor Bjørn Haugen

Acknowledgment

I would like to thank associate Professor Bjørn Haugen for all of his support and help in guiding me in this project work. He has been a great help especially in giving me a push in the right direction and giving me a smooth start by showing me knowledge about technology that was new to me.

I also need to acknowledge Marius Hansen Røed who worked on the same problem earlier and have put a lot of effort into it. His master thesis was the motivation for this continuation of the problem.

C.T.

Summary and Conclusions

Contents

List of Figures

Chapter 1

Introduction

This chapter is an introduction to the Master thesis, describing the problem, objectives, motivation and the background of this project work. It will explain what the problems are with the current problem and how they solve it, as well as the idea of how to solve the current problem.

1.1 Background

Many jobs involves handling and transportation of heavy equipment. Work area is often cramped and verification of equipment access before a transportation is important. Knowing in advance if the transportation is possible before moving the equipment to the required destination is both cost and time efficient.

The goal of this project is to create a software application in .NET using dynamic relaxation in order to check the feasibility of transporting the equipment. Dynamic relaxation is a numerical method that consist of a set of equations to find equilibrium of all forces in a pseudo-dynamic system. It validates the path during movement and clash. It changes the path accordingly if the transportation object clashes. This could potentially save a lot of time and add value by allowing the engineer to run several iterations with a low cost.

This is a continuation on a master thesis written fall 2014, but nothing more than his research in the master thesis will be contributing to my new take on this software application. This new attempt will use dynamic relaxation as well, but will focus on trying to making a license free application made from scratch. The project work autumn 2015 was the start of this thesis and gathering more research and to understand the problem, as well as trying some of it out, was the focus then.

1.2 Objectives

A set of list with objectives involved in this project are

1. Document theoretical basis for the application.
2. Define the input parameters for the core numerical algorithm of the application.
3. Implement quasi-static movement of the equipment along a predefined path.
4. Presentation of closest possible path and possible failed path.
5. Define a workflow for the modeling process geared towards use of the developed algorithm.
6. Develop a user-friendly application, which visualizes and verifies access of equipment and access ways.

Gathering research has been important to create this application. This document consist of research of everything from how to present input parameters and data, research on how to use ParaView and other tools to visualize the movement of the trolley. It also explains dynamic relaxation and about the calculations and theory behind. There also is explanations of the most important implementations for the application.

1.3 Continuation on previous work

This is a continuation on my project work done last semester and on the master thesis "Verification and visualization of equipment access on offshore platforms" done fall 2014 by Marius Hansen Røed. He made a plugin application during this master thesis, which used Navisworks. Navisworks is a project review software that review integrated models to gain better control over project outcomes [1]. Some of the problems with Navisworks though is that it is quite expensive with an annual fee of €2.700 for the most advanced version that included the necessary functions for this plugin. One of the reasons he used Navisworks for the plugin was because of a clash detection feature which is very helpful, but it showed that this feature was very time consuming [8]. This is some of the reasons for the new take on this software application. It also means that some of the theory behind is the same, as well as the calculations.

The project from last semester had the focus on testing out different technologies, as well as doing some research for later. This is why some of the research gathering for this master thesis was finished already and the focus has been on creating the application itself.

1.4 Manual verification

Engineers have to verify manually if it is possible to access the passageways with the different trolleys. They first design the model given some criteria's based on the NORSOK (the Norwegian shelf's competitive position) standards. Then they import the model into PDMS or other 3D tools to validate the transportation from start point to destination. The engineers have to manually move and rotate the transportation vehicle to check for clash with the environment with their own eyes only. Changes of rotation and translation happens separately for each step. Repetition of the verification must happen if there is a clash or if there are some changes to the design or change of the size of the transportation size. This makes it a time consuming process.

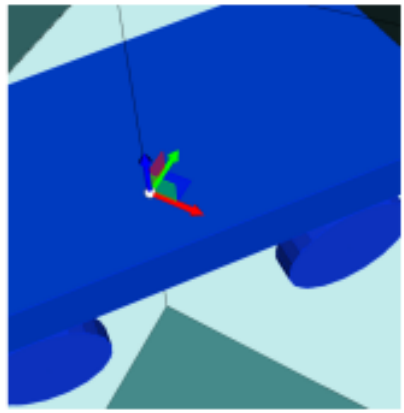

Figure 1.1: This show how the vehicle clash with the wall in a given position [8].

Figure 1.1 show an example of a transportation vehicle trying to turn a corner, as it clashes with the corner. This is just one of many steps an engineer have to check and validate. As it clashes in this example, the engineer has to either redesign the model of the work area or try out another movement and rotation to get past the corner.

1.5 Structure

This is a small explanation about what each of the chapters are about:

Chapter 1 - Introduction:

> Consist of the background of the problem, the motivation and previous work of the same type of work.

Chapter 2 - Input parameters:

> What type of parameters there are and how the application initialize and uses them.

Chapter 3 - Visualization:

> Different visualization parts of this application from file formats visualizing different models to actual visualization ways.

Chapter 4 - Dynamic relaxation:

Calculations and theory of dynamic relaxation. How to solve the verification problem with dynamic relaxation.

Chapter 5 - Implementations of application:

Different implementation parts of the application.

Chapter 6 - Conclusion:

Conclusion with a summary and ideas for further work.

Chapter 2

Input parameters

This chapter will look at the input parameters of this project and how they are initialize and used.

2.1 Input Parameters

The transportation object are supposed to follow a path, through a work environment. This application are going to create a visualization of the movement and automate the verification check. The environment is a 3D geometry model consisting of walls, floors and other obstacles that the transportation object should avoid. This shows that there are three input parameters: A model of the work area, the transportation path and a transportation object. This chapter will look more at these three parameters.

This is a standalone application and do not use any applications to do the calculations, reading and writing the input parameters. It uses ParaView though to visualize the results, but it calculates the results and data before opening it in the visualization tool.

2.1.1 Geometry model

The work area is a 3D model of the surroundings, such as walls or other objects that can work as boundary restrictions for the transportation object. The goal of the whole application is to verify that the model design is suitable for the given transportation. That is why this is maybe one of the most important input parameters, since managing collision will depend on the difficulty to work with the data.

Normally are 3D models of the work area made with PDMS, or other 3D CAD applications. STL-files (STereoLithography) has a common file structure, and is a fundamental and widely used file format. It present the data as multiple vertices sorted in vectors of triangles. This file format makes it possible to visualize the geometry model in ParaView, as well as reading the data in the application for clash detection. Figure 2.1 Shows an example of a simple model opened in ParaView, showing the surface with the edges of triangles that STL-files consist of. Later on in the thesis is more information about STL-files and the usage of it in the application.

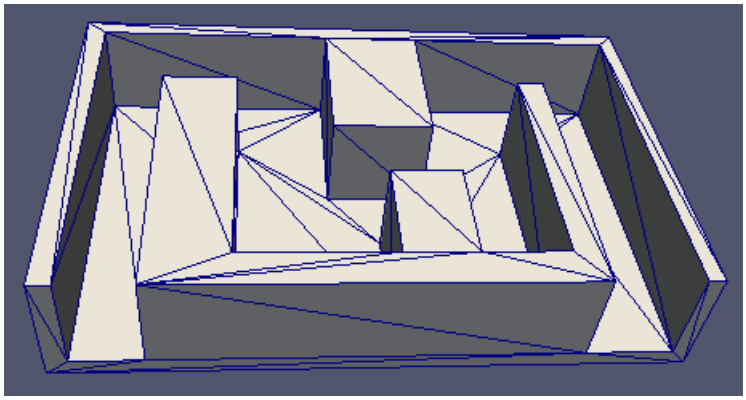

Figure 2.1: A simplified test example of a geometry model, with a pathway.

2.1.2 Path

A path is necessary to validate the transportation. The path is crucial when using dynamic relaxation as validation method, as the trolley has to follow the path. The path should be easy to create, so that validating a geometry model is effective.

Some of the restrictions is that the path has to be continuous, consisting of multiple nodes. It needs a fixed start and end position. Corners should consist of arcs or nodes close together to get a more realistic movement, but this is up to the engineer to choose. It is important that creating a path is easy, with as few clicks as possible and easy to open in the application. The user should have the possibility to choose the number of nodes needed and how close together they are in the different parts of the path.

This application reads the path as a VTK-file (Visualization ToolKit) with points, which is a file format it is easy to read. Using ParaView is possible to create the VTK-file, by creating a Poly Line Source, where you can add points, click and drag to change the position or manually write the coordinates. Figure 2.2 shows a path line created in ParaView. The points is not visible when opening vtk-files with the path in ParaVew as seen at the figure.

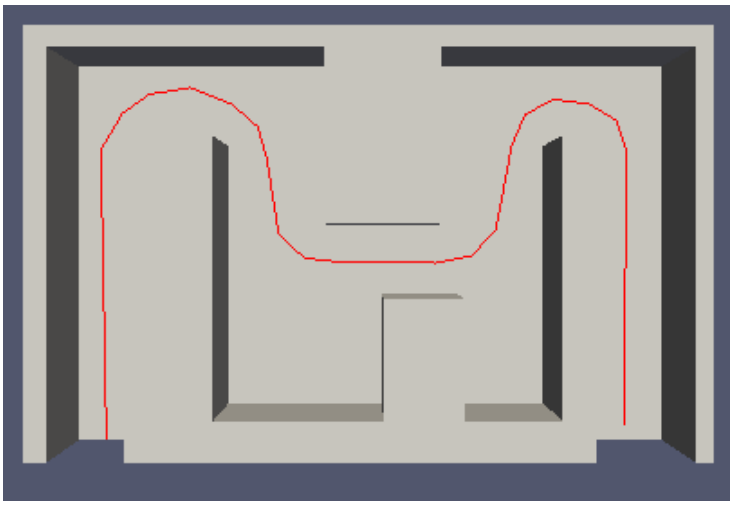

Figure 2.2: Example of a path made out of multiple nodes with change of precision.

2.1.3 Object

The transportation object is usually a trolley or other type of transportation vehicles. The path might change for different transportation objects, as the geometry change. The geometry is simplified during this case, where the object is a rectangular object consisting of vectors as sides and nodes as edges. This makes it easier to check for clash with the geometry model, and to initialize and create the visualization of its movement. It also needs a center that should follow the nodes of the path. The center could change depending on the transportation vehicle, but will in this case just have a center point in the middle. It should be possible to customize the object by choosing size and mass, or give a selection of different objects.

Chapter 3

Visualization

This chapter will look more at the visualization part of the project. This will go in depth of the STL-file format, VTK-format and ParaView.

3.1 STL-file format

An STL-file is a triangular representation of a 3-dimensional surface geometry [5]. The file format is a common CAD-related file type, where multiple triangles made out of three vertices each with a normal vector representing the surfaces. These triangles represent the geometry, by following a set of rules.

3.1.1 Format specifiactions

Each of the vertices in a triangle consist of x-, y- and z-coordinates. Each of the triangles consist of 12 values, with coordinates for each of the vertices and the normal vector (a vector perpendicular to surface with length one).

Two things decides the orientation of each of the triangles. The first thing is the unit normal vector, which is always facing outward of the surface, while the other thing is

the order of the vertices listed in the file format. The order follows a counterclockwise order, outward of the surface.

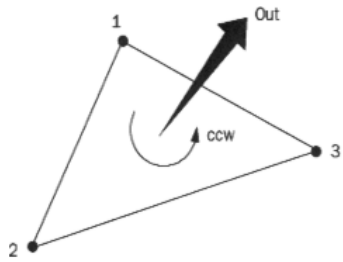

Figure 3.1: A triangle showing direction from unit normal and order of vertices [5].

There are some rules on how the triangles form the geometry. Every triangle joined together has to share two vertices. This means that one vertex from a triangle cannot be in between of two other vertices of another triangle. Figure 3.2 show examples on how violating and following this rule looks like.

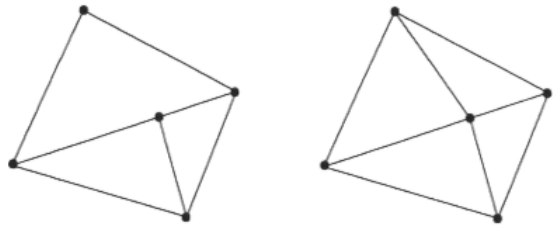

Figure 3.2: Figure to the left is a violation while picture to the right show a correct representation [5].

3.1.2 File Formats

There are two different ways to represent STL-files, as an ASCII-file or binary-file. ASCII-files are not common because of the size of the file, but gives a clear overview of the format of the file. The representation itself is clear and just as explained before. The only difference between the ASCII format and the binary is that all of the values is stored as binary numbers.

```
char[80] (80 bytes) – Header
unsigned long (4 bytes) – Number of triangles

foreach triangle
   float[3] (12 bytes) – Normal vector
   float[3] (12 bytes) – Vertex 1
   float[3] (12 bytes) – Vertex 2
   float[3] (12 bytes) – Vertex 3
   short (2 bytes) – Attribute byte count
end
```

Figure 3.3: an overview of the binary file format [4]

Figure 3.3 shows the structure of an STL-file. It start with a header of 80 bytes, and then 4 bytes of the number of triangles in the file. Each triangle follows, consisting of 50 bytes each. Every value has a size of 4 bytes, so each vertex, with x-, y- and z-coordinates is 12 bytes. It is not normal to use the last 2 bytes of these 50 bytes.

3.2 VTK (Visualization Toolkit)

VTK is a software system for 3D computer graphics, modeling, image processing, volume rendering, scientific visualization and information visualization. It is a C++ toolkit, but also supports wrappers of the toolkit into Python, C# and Java among others. The origin of VTK was in 1993 originally from the textbook "The Visualization Toolkis An Object-Oriented Approach to 3D Graphics". The three graphics and visualization researchers worked in it on their own time, with the motivation was to collaborate with other researchers to create an open framework for creating leading-edge visualization and graphics applications. Users and developers around the world improved and applied the system to other problems when the core of VTK was finished. The three founders left their jobs and founded Kitware Inc. in 1998, to support the huge VTK community. [6].

3.2.1 Data model

The fundamental data structure in VTK is data objects. These data objects is either datasets, such as different grids and meshes, or graphs and trees. Each data objects consist of mesh and attributes. The main datasets is image data, rectilinear grid, structured grid, polydata and unstructured grid.

Building blocks

Even though the data structure of a mesh depends of the type of datasets, some abstractions are the same for everyone. A mesh usually consist of vertices and cells. A cell consist of multiple vertices and maps a region or surface in different ways.

Attributes are discrete values that are not a part of the mesh. These attributes are for example pressure temperature, velocity or mesh color. Attributes are stored as data arrays with a random number of components.

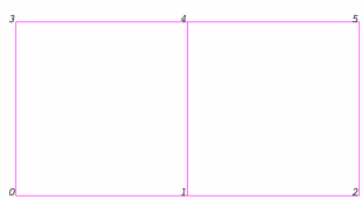

Figure 3.4: Mesh structure of 2 cells from 6 vertices [7]

Datasets

In the beginning of this chapter names the different types of datasets and grids. Uniform rectilinear grid (image data) consist of multiple cells of the same type, dimension and size. This is the most normal type of dataset and takes the least size of the datasets. This is why most algorithms in VTK takes advantage of this kind of grid. Adaptive Mesh Refinement (AMR) datasets are almost the same, except that it consist of a collection of multiple Image Data. Rectilinear grid consist of cells with the same geometry but with different size. Curvilinear grid (structured grid) consist of different geometry and size. These are the structured grids and uses the least memory.

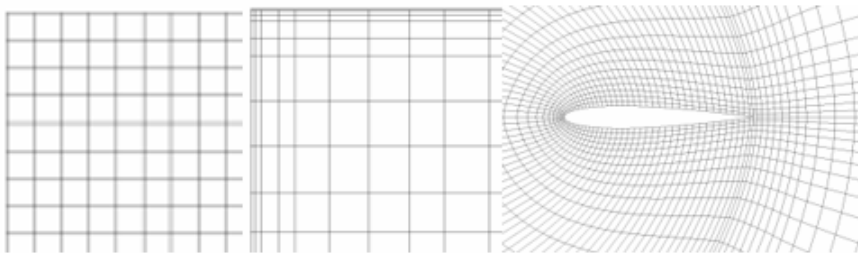

Figure 3.5: Unstructured grid and polygonal grid [7]

An unstructured grid is the most primitive dataset type. This kind of dataset uses more memory, which is why anyone should use this kind of dataset only when no other datasets can represent the data. This kind of datasets can consist of all the cell types VTK supports, in different dimensions and sizes. Polygonal grid (polydata) is a more specialized version of the unstructured grid, who works as a more efficient rendering option. It consist of all cell dimensions except 3D cells.

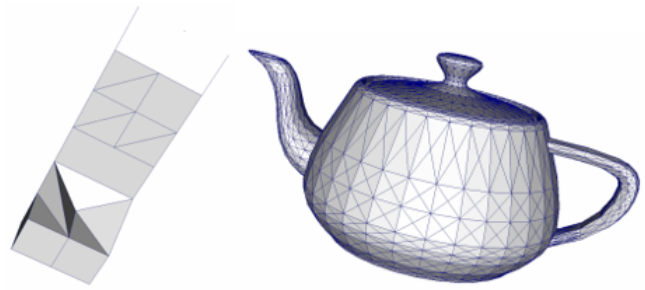

Figure 3.6: Unstructured grid and polygonal grid [7]

3.3 ParaView

ParaView is an open-source data analysis and visualization application. Users can analyze extremely large datasets through visualizations by using different types of techniques. It is a general-purpose, end-user application you can run on your computer or on remote parallel computing resources. It is also a framework with different tools and libraries for applications such as scripting, web visualization and on site analysis (in-situ analysis).

ParaView was a collaborative effort between Kitware Inc. (who also made VTK) and Los Alamos National Laboratory in 2000. Kitware Inc. also started working on its own independent web based visualization system, which contributed to the development of paraView. By collaborating with many other governmental and academic institutions. One of the goals was to create an open, flexible and intuitive user interface, which is user-friendly. It is possible to open in a lot of different visualization files (STL and VTK is just a few), which makes paraView flexible to use in almost any settings. This has been one of the thing focused on during development of new versions. ParaView is downloaded about 100 000 times every year and won the HPCwire Readers' choice award in 2010 and 2012, and editors' Choice award for best HPC Visualization product of technology in 2010. This chapter will look at how ParaView works and give an intro to how to use it and its tools. This part will be important for understanding how this application uses paraView to visualize the given data, as well as giving an idea on how some of its technologies is possible to use in further development of this application if wanted.

3.3.1 basics

ParaView uses VTK as the basic visualization and rendering algorithms. The GUI is just a small part of all functionalities in paraView with many libraries available. This makes it possible to create your own GUI if wanted. It also comes with a Python interpreter (pvpython) where you can automate visualization and post-processing data with python scripts. Pvbatch is just the same as pvpython, except that it is not using paraView. It only take commands from input scripts and run in parallel if it uses the

MPI (message passing interface). Pvpython and pvbatch takes exactly the same inputs. There is also other executables for running on server and doing remote visualization. ParaView is flexible for customization.

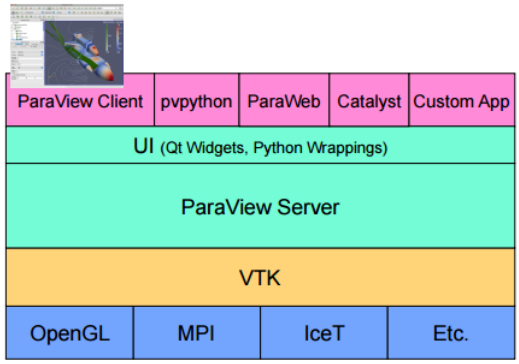

Figure 3.7: The ParaView system [3]

It uses a data-flow paradigm (pipeline structure), which means that data flows through different modules of algorithms, where the output of each element is the input of the next. There are three different types of algorithms. First type is sources, which create data into the system such as readers that read data from files. Filters is algorithms that process datasets and generate, extract or derive features from the data. The last is sinks that saves the data with writers. ParaView includes multiple readers and writers that makes is possible to use all kind of file formats.

The GUI mainly consist of a menu bar and toolbars as almost every other programs, with access to almost every features in ParaView with shortcuts to the most used features. It also consist of a 3D view to visualize the data, a pipeline browser and a property panel. The pipeline browser show the structure and makes it possible to select the pipeline object. The property panel allow you to view and change the parameters of the current pipeline object. There is also possible to add and remove various GUI elements.

Paraview visualize and analyze big datasets so when creating sources or opening a file, the source creates an instance without rendering anything in the 3D view. Since sources (and filters) has parameters, it waits to apply the data on the 3D view if you need to change any of the parameters. It is an apply button to render the view or use the filter on the data already added to the pipeline, as ingesting big data into the system can be

time-consuming processes.

3.3.2 Scripting in ParaView

As mentioned before, ParaView comes with a python interpreter through pvpython. ParaView comes with multiple python packages that provide functionalities in ParaView. This makes it possible to create python scripts to visualize and analyze what you need without using the GUI. However, the GUI also comes with a python shell (Tools – Python Shell) to run scripts and python commands there as well. When writing commands in the shell the GUI interface will update after every commands.

Tracing

ParaView has an option to trace all your actions in the GUI (Tools – Start Trace), and creates python scripts to do the same actions. When you stop tracing (Tools – Stop Trace), ParaView opens an editor window with the generated python code. This is one great way of learning the basics of the python packages, which comes with ParaView. It is also a nice way of automate actions you do multiple times in ParaView to be more efficient.

3.3.3 Temporal file series

Time in ParaView is handles in different ways depending on the file format. Several file formats makes it possible to save multiple time steps in one file, while others save each step in a file series (with each time step as a new file). ParaView recognizes time series in the Open File dialog and shows them as a grouped element with the filename end with "..vtk" (see figure 3.8). It is also possible to open one of the files individually if necessary.

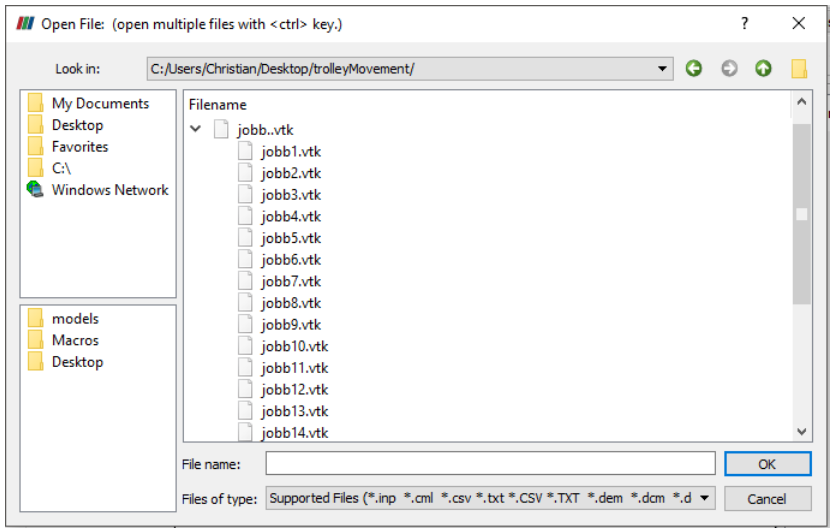

Figure 3.8: an example of a temporal file series in ParaView

Chapter 4

Dynamic Relaxation

All of the calculations are the same as the master thesis of Marius Hansen Røed. This makes it unnecessary to start all over again with this topic. That means that this chapter will mostly be a summary from the master thesis "Verification and visualization of equipment access on offshore platforms" [8] with a few differences. The chapter will explain the method, as well as giving a brief explanation on how to solve the verification problem.

Dynamic relaxation is a numerical method that consist of a set of ordinary differential equations to find equilibrium of all forces in a dynamic system as mentioned in the introduction. Form-finding is one of many ways of using dynamic relaxation [2], by finding a geometry where all forces are in equilibrium. The geometry consist of multiple nodes, each with a mass. An applied force will lead to oscillation in the system, which will always try to maintain equilibrium. Simulations of this dynamic process checks through all the iterations and looks at the geometry in all time steps during the external loads.

4.1 Method

ynamic relaxation originated from the equation of motion, which for a system like in figure 4.1 is:

$$\mathbf{P}_{ij} = [\sum K\delta]_{ij} + \mathbf{M}_{ij}\ddot{\delta}_{ij} + C\dot{\delta}_{ij} \tag{4.1}$$

$\mathbf{P}_{ij}$ is the external load vector while $[\sum K\delta]_{ij}$ and $C\dot{\delta}^n_{ij}$ is the internal load vectors. K is the node stiffness while δ is the displacement. C is the damping coefficient, $\ddot{\delta}$ is the acceleration and $\dot{\delta}$ is the velocity. ij refers to node number i in direction j.

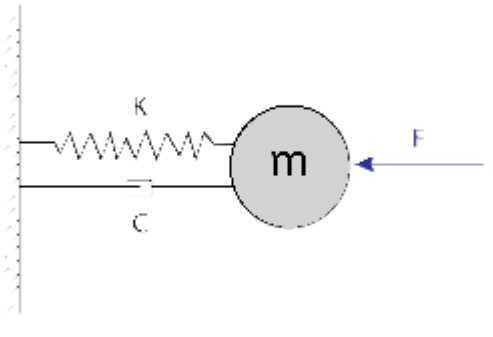

Figure 4.1: A dynamic system with a mass m, connected to a spring with a stiffness K, a damper with a coefficient C and a external force F [8].

By introducing the residual force, which is the difference between the external ($\mathbf{P}_{ij}$) and the internal forces ($[\sum K\delta]_{ij}$ and $C\dot{\delta}^n_{ij}$), we get that:

$$\mathbf{R}_{ij} = \mathbf{P}_{ij} - [\sum K\delta]_{ij} - C\dot{\delta}^n_{ij} \tag{4.2}$$

This give us a residual force equation like this after substituting equation 4.1 into equation 4.2:

$$\mathbf{R}_{ij} = \mathbf{M}_{ij}\ddot{\delta}_{ij} \tag{4.3}$$

The equation above results in equations for acceleration and velocity for each iteration,

In order to calculate the displacement at next time step (n+1):

$$\ddot{\delta}_{ij}^{n+1} = \frac{\mathbf{R}_{ij}^{n}}{\mathbf{M}_{ij}} \tag{4.4}$$

$$\dot{\delta}_{ij}^{n+1} = \dot{\delta}_{ij}^{n} + \ddot{\delta}_{ij}^{n}\Delta t \tag{4.5}$$

$$\delta_{ij}^{n+1} = \delta_{ij}^{n} + \dot{\delta}_{ij}^{n}\Delta t \tag{4.6}$$

These equations are used until $\mathbf{R}_{ij} \rightarrow 0$, which means that the system is approximately in equilibrium.

Mass, stiffness and time increment has to be set so that the method is realistic, where assumptions of stiffness and time increment is important. The mass needs to be realistic compared to the stiffness. The damping coefficient is set to be critical damping, which makes the system return to equilibrium the fastest possible way.

$$C_{critical} = 2\sqrt{mK} \tag{4.7}$$

4.2 Solving the Verification Problem

When the transportation object is following the path, the object moves from one path node to the other (see figure 4.2). This movement is one iteration. The dynamic relaxation method verifies every step of the path. Modification of the path will occur when a clash occurs if possible, so that the new path do not involve clash.

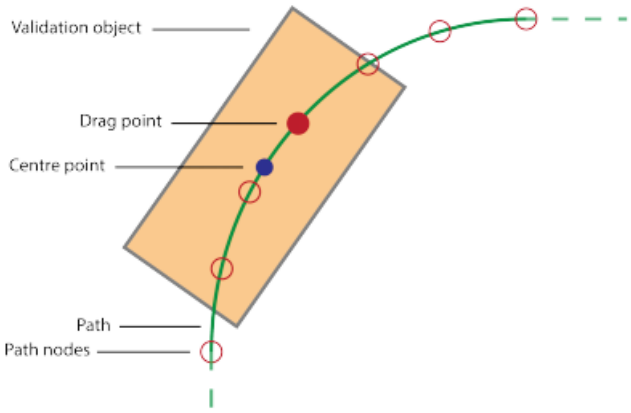

Figure 4.2: This represent how the transportation object follow the path [8].

Between the center point of the transportation object and the drag node, is a damper (see figure 4.3). The distance between them is the displacement δ. Velocity $\dot{\delta}$ and acceleration $\ddot{\delta}$ is a result from δ and the damping coefficient C. The transportation object moves closer to the drag node caused by the damper, during each time step. This method will continue to the next iteration (drag node will move to the next path node), when the system reaches equilibrium (Residual force $\mathbf{R}_{ij} \rightarrow 0$).

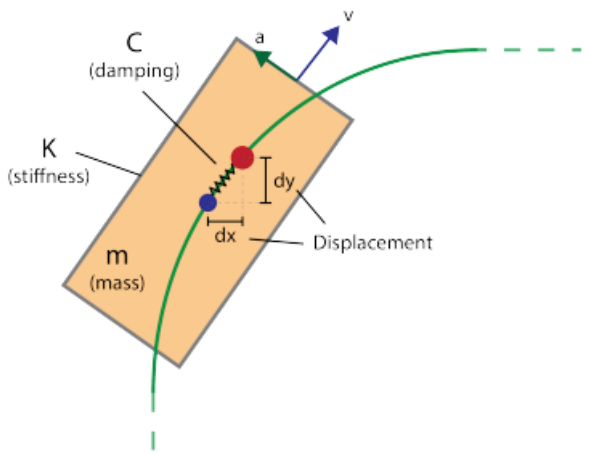

Figure 4.3: This represent how the transportation object follow the path [8].

Clash with the geometry model adds an external force to push the transportation object back away from the model. The force creates unbalance in the system, where the dynamic relaxation method tries to push the system back into balance. It will return false if the system cannot reach equilibrium after it reached a given tolerance iteration (maximum number of iterations to reach equilibrium). This means that it is not enough space for the object to move through this area of the geometry model. If the system reaches equilibrium ($\mathbf{R}_{ij} \rightarrow 0$), the new node position is stored and the drag node moves to the next path node.

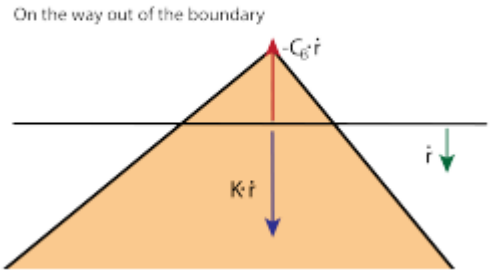

Figure 4.4: Transportation object during clash, with the velocity vector $\dot{r}$. Damping opposite direction of the applied force. Applied force and damping has same direction in the moment it clashes (velocity direction into the geometry model).[8]

Chapter 5

Implementation of application

This chapter will explain how some of the main implementations of this application uses the theory found. There are also some explanations on how the application deals with movement of the transportation object. The application is a .NET application written in C and XAML (Extensible Application Markup Language). Everything except of the GUI (Graphical user interface) components is in C.

5.1 Reading STL-files

Figure 3.3 shows the overview of a binary file format and is mainly what the implementation looks like. A binary reader opens the file in C#. The header is not important and skipped, while the number of triangles helps to iterate through all of the triangles. The normal vector and all of the vertices in the triangles is stored in a list of vectors. Since all list elements has to be stored as the same type of variable, vector was chosen, even though point would make more sense. This is because it is more functions and possibilities for the normal vector. As long as needing these functions is uncertain, the implementation stores the variables as vectors.

5.2 Reading Path

The application can read paths in VTK-format in ASCII (American Standard Code for Information Interchange) format only. You should use ParaView when creating the path as it is the easiest option and since you have to use ParaView to visualize the job anyway. Corners of the path should be arcs so that the movement is as realistic as possible, but this is not possible when using dynamic relaxation. Nodes close together is therefore the only way to make movements similar to an arc.

The headers is pre-defined, the structure has the polydata section with all the points first and then it defines the order of the points. As the order of the point is from zero and up the last part is not important for reading the path. Reading the points from the file, when everything else is pre-defined, is easy itself.

5.3 Transportation object

A new instance of the transportation object class defines the object and all of its movement. It initializes the parameters from the width, length, height, mass, a start position and the point it faces in the beginning. Center point of the trolley is set as the start position, while calculations from the pre-defined stiffness variable and the mass finds the damping coefficient of the trolley. It sets the position of all the box edges with the width, length and height from the center point, before rotating them facing the right direction.

There are two types of movement for the trolley, and that is rotation and translation. The trolley faces the next node in the path, where a function consisting of a matrix transformation rotates the edges of the box around the center. This transformation does only rotations around the z-axis, as it did not work for multiple rotations. This means that it completely works for cases where the trolley moves in the xy-plane, and not in other planes. To implement rotations among multiple planes, quaternions is the best way to go. For every step the transportation object moves, a function update the position of the center point and the edges of the box, with the distance it moves during the dynamic relaxation algorithm.

5.4 Writing trolley visualization

A temporal file series with different positions of the trolley creates the animation when opened in ParaView. These files are VTK-files, so the way the files is structured is similar to the file with the path. To create the trolley in the VTK-file, the same structure with polydata of the 8 points the box consist of, then the structure of how the points forms the cells instead of the line order in the path. The file structure then look like this:

 # vtk DataFile Version 4.0
vtk output
ASCII
DATASET POLYDATA
POINTS N float
x(0) y(0) z(0)
x(1) y(1) z(1)
...
x(N-1) y(N-1) z(N-1)

POLYGONS M K
4 0 1 3 2
4 2 3 5 4
...

The number of points, *N*, is in this case 8 where all of the points have x-, y- and z-coordinates. The polygons has *M* number of surfaces, or 6 for a box, where K is the total of numbers the polygon part consist of. Each surface has a first number that explains how many point the polygon consist of and then which of the points following.

Last part of this file is the color attributes of the box. The box is green when not colliding and red when it does. This is how to give each of the surfaces a color:

CELL_DATA M

SCALARS $cell_scalarsint1$

LOOKUP_TABLE default

0 1 2 3 4 5

LOOKUP_TABLE default M

red green blue alpha

...

First up is defining each of the surfaces (cells) in the lookup table, and then the lookup table define the color of each surfaces in the lookup table with RGBA (red, green, blue and alpha) values between 0 and 1. Alpha defines the opacity of the color.

5.5 Dynamic Relaxation

To iterate from one position to the next, every internal variables in the algorithm is set to zero, while the displacement between the current position of the transportation object and the next node is set. A while loop runs while the residual force is not close to zero. The residual force is equal to clash force minus internal force minus the damping force (this depends on the directions of the forces). It increases in the beginning as the damping velocity is zero and therefor the damping force as well. The new acceleration, velocity and displacement updates the new position of the transportation object before doing the same calculations over again in the loop. When the collision detection discovers a clash, it finds the force it takes to push it back out of the wall and adds it to the residual force. It adds the current position to the new path and continues to the next node, when the residual force is close to zero.

Chapter 6

Conclusion

This chapter will summarize my work and give a conclusion to what I have done. It will discuss some of the findings and some of their problems and strengths.

6.1 Conclusions

6.2 Discussion

6.3 Further Work

Bibliography

[1] Autodesk naviswork software. http://www.autodesk.com/products/navisworks/overview/. [Online; accessed 17-December-2015].

[2] Dynamic relaxation. https://en.wikipedia.org/wiki/Dynamic_relaxation. [Online; accessed 20-December-2015, last modified 22-August-2013].

[3] The paraview tutorial. http://www.paraview.org/Wiki/images/5/56/ParaViewTutorial44.pdf. [Online; accessed April-2016].

[4] Reading an stl file with c++. http://www.sgh1.net/b4/cpp-read-stl-file. [Online; accessed October-2015].

[5] The stl format - standard data format for fabbers. http://www.fabbers.com/tech/STL_Format. [Online; accessed 17-December-2015].

[6] VTK - overview about visualization toolkit by kitware. http://www.vtk.org/overview/. [Online; accessed 11-May-2016].

[7] Ayachit, U. (2015). *The ParaView Guide - Community Edition, Updated for ParaView version 5.0*. Kitware Inc.

[8] Røed, M. H. (June 2014). Verification and visualization of equipment access on offshore platforms.