Generative design approach

Throughout this section, the code written for the purposes of this thesis is discussed in detail. The code is extensive and lengthy. Details considered trivial or irrelevant are left out of the discussion. The entirety of the code is accessible to the reader and is extensively commented for clarification.

Template

A template model is initially created. The template can be altered according to the specified design methodology applied. The template contains all necessary information and all appropriate FEM settings have been applied. This decreases the need for alterations later.

Template parameters need to be specified by the user. An overview of the user-specified template parameters is given in Table below.

|  |  |
| --- | --- |
| Variable | Description |
| case | The template case identifier |
| x\_e | The number of elements in the x-direction |
| y\_e | The number of elements in the y-direction |
| e\_s | The side length of the element in mm |
| b | The number of elements reserved for the unit boundary width |
| n\_steps | The number of analysis steps in the second of analysis |
| table\_name | The name of the function applied to the template |
| app | The dimensions of the applied boundary conditions |

The template case identifier refers to one of three deformation cases applied to the template. These cases are discussed in detail in Section.

The user-specified template parameters are used to define a template class object. The template class object calculates more parameters used in the construction of the template. The relevant calculated parameters are outlined in Table below.

|  |  |  |
| --- | --- | --- |
| Variable | Calculation | Description |
| x\_s | e\_s\*x\_e | The side length of the template in mm in the x-direction |
| y\_s | e\_s\*y\_e | The side length of the template in mm in the y-direction |
| ogd\_mat |  | The non-linear Ogden material model |
| x\_n | x\_e + 1 | The number of nodes in the x-direction |
| y\_n | y\_e + 1 | The number of nodes in the y-direction |
| n\_e | x\_e\*y\_e | The total number of elements in the template |
| n\_n | x\_n\*y\_n | The total number of nodes in the template |
| e\_internal |  | The list of internal element IDs that are allowed to be removed |
| n\_external |  | The list of external node IDs that are allowed to be removed |
| t\_id | <case>\_<x\_e>x<y\_e>  \_<x\_s>\_<y\_s> | The template ID |
| grid |  | A representative grid of ones |

The template is then created in MarcMentat. The nodes are created starting at the global origin on the XY-plane. The nodes are incrementally added in the positive x-direction. The nodes are spaced apart as defined by e\_s. Once a row of nodes is completed as defined by x\_n, the y-coordinate is positively incremented as defined by e\_s. A new row of nodes is created. This process is repeated until completed as defined by y\_n.

Four nodes are used to make square 2D elements. Starting at the global origin, elements are incrementally added in the x-direction until completed as defined by x\_e. All rows are added until completed as defined by y\_e.

The graph used to apply the boundary conditions is applied. The boundary conditions are applied according to the case identifier. The boundary conditions related to each case are detailed in section.

Mechanical planar strain geometric properties are added to all elements. The Ogden material model for Mold-Star 15 is applied to all elements. A single contact body is defined containing all elements. The loadcase containing the fixed and forced displacement boundary conditions is created. The job for the loadcase is created.

The template is saved at this point. All units created during a simulation are built from this template.

The template job is run and its success evaluated. The process of running a simulation and evaluating its success is outlined in Section.

The template model is saved again. Meaningful template parameters and data obtained from the template is written in a human-readable manner to a log file. The template class object is saved to a file that can be accessed later.

Analysis approach

The analysis approach is specified by the user. Two analysis methods are available.

A Monte Carlo-styled analysis is available. The user must specify the unit generation method and the number of units to be generated.

An analysis making use of a genetic algorithm is also available. The user-specified parameters are detailed in Table below.

|  |  |
| --- | --- |
| Variable | Description |
| gen | The number of generations |
| prob | The probabilities of an occurrence of crossover, random mutation, and biased mutation |
| point | The potential number of occurrences per unit of crossover, random mutation, and biased mutation |
| meth | The unit generation method |

Monte Carlo-styled analysis

A population of units is generated according to the specified unit generation method. The unit generation process is outlined in Section. The unit generation process results in a list of unit class objects.

The list of unit class objects is used to create and run the population of units. The process of running a list of units is outlined in Section.

The units are ranked according to performance metrics specified by the template case. A ranked list of the unit IDs is provided in descending order in a text file. The ranking process is discussed in more detail in Section. Further evaluation of unit performance may be carried out by manual inspection.

Unit Generation

Unit generation refers to the process of automatically specifying internal geometry of a unit based on the predefined template.

Three methods of unit generation are implemented and investigated. All three methods provide a list of internal element IDs provided to MarcMentat as elements to be removed from the template file.

Random generation

Random generation is implemented as a baseline to compare with the other two unit generation methods.

A random generation seed is used to allow for replicability. A number of elements to be removed is first selected. This number has a possible range of zero to all internal elements. A list of unique internal element IDs is then selected and sorted.

L-Systems

An L-System-like generation method is implemented due to the L-System features of efficiency, compactness and scalability. L-Systems are defined using class objects containing all necessary information. L-System class object parameters are detailed in Table below.

|  |  |
| --- | --- |
| Variable | Description |
| vocab | The L-System vocabulary |
| gramm | The L-System grammar |
| axiom | The initial axiom of the L-System |
| n | The number of iterations to apply to the L-System |
| seed | The random generation seed used to generate the L-System (if applicable) |
| word | The resulting word of the L-System |

The L-System vocabulary is itself a class object. The vocabulary is predefined. It consists of variables and constants. The vocabulary is outlined in table below. The vocabulary interpretation differs from traditional L-System interpretations. The interpretation is required to fill in elements of a grid. Traditional L-System interpretations result in lines of varying lengths and angles being drawn. A unique interpreter was designed and implemented for the purposes of this thesis.

|  |  |
| --- | --- |
| Variable | Interpretation |
| F | Create an element at the current position and increment the current position in the current direction |
| f | Increment the current position in the current direction |
| + | Rotate the current direction by 45° clockwise |
| - | Rotate the current direction by 45° counterclockwise |
| Constant |  |
| [ | Push the current position to the position memory stack |
| ] | Pop the latest position on the position memory stack and return to that position |
| ( | Push the current position to the position memory stack  All directional variables, i.e. + and -, in the word following this constant are reversed until the ) constant is encountered |
| ) | Pop the latest position on the position memory stack and return to that position |

Randomly generated L-Systems are manipulated using a random generation seed. Randomly generated L-Systems have requirements and specifications implemented to ensure the validity of the L-System.

At least one L-System rule must be defined. This rule must apply to the variable “F” and must itself contain at least one instance of the letter “F”. Up to three additional rules may be defined, one for each of the remaining variables.

Rule components are selected from a predefined list included in Table below. Rule components were selected for various reasons. Rule components containing more than one character allow for variability in rule length. Rule components containing two identical directional variables specify 90° rotations. Rule components enclosed within square brackets allow for branches to exist within the L-System.

During initial trials, many issues were encountered in allowing rules to be generated with single or unevenly matched brackets. The predefined L-System axioms contain square and round brackets. If rules contain single or unevenly matched brackets, the interpretation of the resulting word will not follow the specified symmetrical axis.

Twelve axioms are predefined and outlined in Table below. Axioms were defined according to desirable symmetry conditions.

|  |  |  |
| --- | --- | --- |
| Axiom | Rotation/Reflection | Axis |
| [F]++++[F] | Rotation | Horizontal |
| --[F]++++[F] | Rotation | Vertical |
| [F]++[F]++[F]++[F] | Rotation | Horizontal and vertical |
| +[F]++++[F] | Rotation | Diagonal |
| -[F]++++[F] | Rotation | Negative diagonal |
| +[F]++[F]++[F]++[F] | Rotation | Diagonal and negative diagonal |
| [F]++++(F) | Reflection | Horizontal |
| --[F]++++(F) | Reflection | Vertical |
| [F]++(F)++[F]++(F) | Reflection | Horizontal and vertical |
| +[F]++++(F) | Reflection | Diagonal |
| -[F]++++(F) | Reflection | Negative diagonal |
| +[F]++(F)++[F]++(F) | Reflection | Diagonal and negative diagonal |

Running a Job

The command to run the job is sent to MarcMentat. Jobs may take anywhere from 0.01 seconds to 300 seconds to complete. This depends on the complexity of the model and the number of cut-backs during calculation required to accurately solve for the model behavior.

MarcMentat creates several files during the process of running a job. The log file specifies the exit condition of the job. The log file is not created at the start of the job.

A model is always saved just before a job is run. The timestamp of this saved model is used for evaluation of the log file. It is first determined if the log file exists. If it does not exist, the code waits for 1 second before checking again. This repeats until the log file is found to exist. If the log file is found, its timestamp is compared to the model file’s timestamp. If the log file is older than the model, i.e. it is a log file of a previous run of the model, the code waits 1 second before checking if it has been updated. If the log file is newer than the model, it is inspected for the exit number string or the access violation string.

If the exit number string is found, the exit number is evaluated. Two exit numbers and an error case are identified and defined in Table below.

|  |  |
| --- | --- |
| Exit number | Description |
| 3004 | A successful run |
| 67 | A license server connection time out or failure |
| Other | An unsuccessful run |

If exit number 3004 is found, the model is recognized as having run successfully. The model output file is opened. All relevant data is read from the model output file and written to clearly labeled CSV files. Any relevant data that must be calculated externally from MarcMentat is calculated and also written to clearly labeled CSV files. Relevant data is outlined and motivated in Section .

If exit number 67 is found, the job is rerun and the entire process as detailed above is repeated until the exit number can be rerun again. If an access violation string is found, it is treated identically to exit number 67.

If any other exit number is found, an error message with the number is displayed. No results are obtained from the model. The model ID is logged appropriately. The code continues on to the next model.

Cases

Case 1 – Pure elongation

Case 1 is a case of pure elongation as defined by … This case has no rigid body modes. This case has applications in causing extension.

Four boundary conditions are applied. They are outlined in Table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Boundary | Constraint | Direction |
| bc\_fd\_yy1 | Bottom edge | Fixed | y |
| bc\_fd\_yy2 | Top edge | Forced displacement | y |
| bc\_fd\_xx1 | Left edge | Fixed | x |
| bc\_fd\_xx2 | Right edge | Fixed | x |

The boundary conditions as applied in MarcMentat are illustrated in Figure below. The resulting deformation is illustrated in Figure below.

Case 2 – Pure shear

Case 2 is a case of pure shear as defined by … This case has no rigid body modes. This case has applications in causing angular extension or deformation.

Four boundary conditions are applied. They are outlined in Table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Boundary | Constraint | Direction |
| bc\_fd\_xy1 | Bottom edge | Fixed | x |
| bc\_fd\_xy2 | Top edge | Forced displacement | x |
| bc\_fd\_yx1 | Left edge | Fixed | y |
| bc\_fd\_yx2 | Right edge | Fixed | y |

The boundary conditions as applied in MarcMentat are illustrated in Figure below. The resulting deformation is illustrated in Figure below.

Case 3 – Elongation of one side

Case 3 is a case of elongation applied to one side. The opposite side is kept at a fixed length. This case has applications in causing expansion or curling.

Six boundary conditions are applied. They are outlined in Table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Boundary | Constraint | Direction |
| bc\_fd\_yy1 | Bottom edge | Fixed | y |
| bc\_fd\_yy2 | Top edge | Fixed | y |
| bc\_fd\_xf1 | Bottom left corner | Fixed | x |
| bc\_fd\_xf2 | Bottom right corner | Fixed | x |
| bc\_fd\_xn | Top left corner | Forced displacement | x |
| bc\_fd\_xp | Top right corner | Forced displacement | x |

The boundary conditions as applied in MarcMentat are illustrated in Figure below. The resulting deformation is illustrated in Figure below.