Generative design approach

In this section, the software pipeline is discussed in detail. The software pipeline is complex, and many components are interrelated. Details considered trivial or irrelevant are left out of the discussion. The entirety of the code is available to the reader if they would wish to investigate or make use of it. The code is extensively commented for any clarification needed.

Template

The generative design approach as implemented requires large populations of similar units to be created and run. The units need to be similar to meaningfully draw comparisons between them and select the best performing units. A template unit is created containing all necessary and non-unique specifications. The template has all appropriate FEM settings applied. The template can be altered according to the specified design methodology.

The template is a 2D grid of square elements. 2D quad shell elements are used. The template has boundary conditions applied causing deformation according to a specified case. Cases are outlined in Section.

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 |
| ogd\_mat | The Ogden material model parameters |
| n\_steps | The number of analysis steps in the second of analysis |
| table\_name | The name of the function applied to the template |
| d\_mag | The dimensions of the applied boundary conditions |
| p\_mag |  |

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 defined. 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 user-specified. Two analysis methods are available and discussed below.

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. Both analysis methods use the same approach to the initial population generation.

A list of maximum and minimum parameter values is defined. The parameter values are appropriate to the unit generation method specified. The different parameters, ranges and motivations are outlined in table below. The range values are inclusive.

|  |  |  |
| --- | --- | --- |
| Parameter | Range | Motivation |
| Random | | |
| Seed | 1 | The seed is limited to the number of units requested by the simulation |
| Number of units to be generated |
| Number of elements removed | 0 | The complete range of elements which may potentially be removed is included |
| Number of internal elements |
| L-System | | |
| Seed | 1 | The seed is limited to the number of units requested by the simulation |
| Number of units to be generated |
| Axiom ID | 1 | The complete list of axioms is included |
| Number of predefined axioms |
| Number of rules | 1 | At least one rule is required for a valid L-System |
| Number of L-System variables | The number of rules is limited to the number of L-System variables |
| Rule length | 2 | A rule must be longer than one character |
| 5 | Potential complexity is limited |
| Number of iterations | 1 | At least one iteration must be applied for a valid L-System |
| 5 | Potential complexity is limited |
| CPPN | | |
| Seed | 1 | The seed is limited to the number of units requested by the simulation |
| Number of units to be generated |
| Model ID | 1 | Potential complexity is limited |
| Scale | 1 | Potential complexity is limited |
| Number of hidden layers | 2 | At least two layers are required to have a valid network |
| 10 | Potential complexity is limited |
| Size of the initial hidden layer | 2 |  |
| 32 |  |
| Element removal threshold | 0 | The complete range of elements which may potentially be removed is included |
| 100 |

A population is randomly generated. Each population member is a list of parameters within the bounds specified in Table above. If the unit generation method is random, the two parameters are used to generate a list of elements to be removed directly.

If the unit generation method is specified as L-Systems, the parameters are used to generate a random L-System. The L-System word is interpreted to obtain a list of elements to be removed.

If the unit generation method is specified as CPPNs, the parameters are used to generate a random CPPN. The CPPN is set to generate the maximum number of models specified by the parameter. A CPPN model class object is defined with the model ID specified by the current parameters. The CPPN model class object is interpreted to obtain a list of elements to be removed.

Each list of elements to be removed is paired with the relevant class object or set of parameters. Each pair is added to the population list. The population list is used to create the units in Marc Mentat and run them.

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.

Genetic Algorithm

Genetic algorithm parameters are specified by the user. The parameters are outlined in Table below.

|  |  |
| --- | --- |
| Variable | Description |
| gen | The number of population generations to be run |
| prob | A list of the probabilities of crossover, random mutation, and biased mutation occurring |
| point | A list of the number of potential points of occurrence of crossover, random mutation, and biased mutation |

An initial population is generated as discussed in Section. A population will be run, evaluated and evolved for the number of generations specified.

Two parent population members are selected for evolution. Parents are ranked in descending order according to their fitness. The ranking procedure is discussed in detail in section. The parent population is iterated through from the most fit to the least fit parent until a parent has been selected. A random number between 0 and 1 for selection is generated. A fitness weighting function is used to determine the selection of a parent. The fitness weighting function is defined as Equation

Where

is the fitness weighting

is the population size

is the rank of the selected member

is a user-defined fitness constant. is set to 1.

The cumulative fitness weighting of all units before and after the current population member is calculated. If the random number is between the two cumulative fitness weightings, the member is selected as the parent.

Population members may evolve in three ways.

Crossover may occur between two parents. For every crossover point, a random number between 0 and 1 is generated. If the number is below the specified probability, crossover occurs. A random index in the list of parent parameters is selected. The parameters are swapped between parents from this index onwards. The two resulting children are potentially evolved further.

Random mutation of a child parameter may occur. For every random mutation point, a random number between 0 and 1 is generated. If the number is below the specified probability, random mutation occurs. A random parameter of the child is selected. The parameter is randomly changed to any allowed value for that parameter.

Biased mutation of a child parameter may occur. For every biased mutation point, a random number between 0 and 1 is generated. If the number is below the specified probability, biased mutation occurs. A random parameter of the child is selected. Another random number between 0 and 1 is generated. If the number is below 0.5, the parameter is decreased by 1. If the number is above or equal to 0.5, the parameter is increased by 1. The parameter is bounded by the allowed ranges.

All children are added to the population list. The new population list is used to generate the next population of units.

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.

|  |  |  |
| --- | --- | --- |
| Number | Rule Component | Result |
| 1 | F | Create an element at the current position and increment the current position in the current direction |
| 2 | f | Increment the current position in the current direction |
| 3 | + | Rotate the current direction by 45° clockwise |
| 4 | - | Rotate the current direction by 45° counterclockwise |
| 5 | ++ | Rotate the current direction by 90° clockwise |
| 6 | -- | Rotate the current direction by 90° counterclockwise |
| 7 | fF | 2, then 1 |
| 8 | Ff | 1, then 2 |
| 9 | [F] | Push the current position to the position memory stack, then 1, then pop the latest position on the position memory stack and return to that position |
| 10 | [f] | Push the current position to the position memory stack, then 2, then pop the latest position on the position memory stack and return to that position |
| 11 | [+F] | Push the current position to the position memory stack, then 3, then 1, then pop the latest position on the position memory stack and return to that position |
| 12 | [+fF] | Push the current position to the position memory stack, then 3, then 7, then pop the latest position on the position memory stack and return to that position |
| 13 | [-F] | Push the current position to the position memory stack, then 4, then 1, then pop the latest position on the position memory stack and return to that position |
| 14 | [-fF] | Push the current position to the position memory stack, then 4, then 7, then pop the latest position on the position memory stack and return to that position |

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 adhere to the specified symmetrical conditions.

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

|  |  |  |
| --- | --- | --- |
| 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 |

The true axiom is a single “F”. All rules are applied for the specified number of iterations. The resulting word is placed within the axiom at all positions occupied by an “F”. This allows for the symmetry conditions to be maintained across iterations while retaining “+” and “-” as variables.

The final word is interpreted according to the specified internal dimensions of the template. The word is interpreted character by character.

The number of open round brackets in the word is counted. For every open round bracket, the first subsequent closed round bracket is identified. Each string segment starting and ending with an open and closed round bracket is extracted. The round brackets are replaced with the equivalent square brackets. Pluses and minuses are swapped to invert all directional changes within the round brackets. This allows for reflection symmetry conditions to be applied. The original string segment is replaced with the adjusted string segment.

Every character in the word is interpreted from the initial character. The interpretation applies to a grid of squares identified by x- and y-coordinates. The origin square is defined by the coordinates (0, 0). Positive and negative directions are applied as per standard convention. The interpretation starts at the origin with the direction at 0°, i.e. in the positive y-direction.

If the character is “F”, a check is done to determine if the stack is at its initial value or if the current element is the initial element. If it is either, the element coordinates are set to the origin. If it is neither, the x- and y-coordinates are incremented in the current direction. The new element coordinates are added to the list of coordinates.

If the character is “f”, the procedure is identical to the procedure for the character “F”, except that the new element coordinates are not added to the list of coordinates. Thus

If the character is “+”, the current direction is incremented by 45°. If the character is “-”, the current direction is decremented by 45°.

If the character is “[”, the current position and direction is added to the stack. If the character is “]”, a check is done to determine if the stack is at its initial value. If it is, the initial coordinates and direction are fetched. If it is not, the latest coordinates and positions are popped from the stack. A check is then done to determine if the stack is at its initial position. If it is, a flag is set to indicate as such.

The complete list of coordinates is sorted in ascending order. All duplicate element coordinates are removed. The coordinates are shifted from centering around the origin to centering around the central element of the unit. All coordinates outside of the internal space of the unit are dropped. The remaining coordinates are interpreted as element indices. The element indices are compared to the list of all internal element indices and the list of elements to be removed is obtained.

CPPNs

An CPPN-like generation method is implemented due to the CPPN features of image generation and scalability. CPPNs are defined using class objects containing all necessary information. CPPN class object parameters are detailed in Table below.

|  |  |
| --- | --- |
| Variable | Description |
| seed | The random generation seed used to generate the CPPN |
| mod\_n | The number of models to be generated by the CPPN |
| scale | The scale of the focus on the model |
| hl\_n | The number of hidden layers in the network |
| hl\_s | The size of the initial hidden layer |
| thresh | The rounding or element removal threshold |
| x | The number of elements in the x-direction |
| y | The number of elements in the y-direction |

A random generation seed is used to determine the CPPN’s initial layer, hidden layers, and activation functions. The random generation seed allows for replicability of CPPNs.

A single CPPN may generate a number of models. A CPPN model is defined as a class object containing the relevant information. CPPN model class object parameters are detailed in Table below.

|  |  |
| --- | --- |
| Variable | Description |
| cppn | The CPPN class object used to generate the model |
| mod\_id | The model ID |
| grid | The binary model grid |

A CPPN model may be scaled inwards or outwards. The scale parameter was set at 1, i.e. no scaling, in order to reduce complexity. The functionality of the scaling was left implemented to allow for future tests.

The threshold parameter allows for two approaches to the removal of elements. The CPPN outputs 2D arrays of values ranging from 0 to 1. If the threshold parameter is set from 0 to 1, it is interpreted as a rounding threshold. All values above or equal to the threshold are set to 1 and all values below or equal to the threshold are set to 0. If the threshold parameter is set above 1 to 100, it is interpreted as a percentage of elements to remove. The lowest values making up the specified percentage of all values are set to 0 and the rest of the values are set to 1.

The x- and y-dimensions are specified initially. All hidden layers barring the initial layer have a number of nodes equal to the x-dimension multiplied with the y-dimension. The CPPN will result in a model that fits perfectly within the internal space of the unit.

CPPN models obtained for the purposes of this this thesis are at much lower resolutions than traditional CPPN models. A reduction in complexity was deemed appropriate. Traditional CPPNs allow for unique activation functions to be applied to each node. This was not implemented in order to reduce complexity. Five activation functions are available for the hidden layers of the CPPN. Activation functions are applied to an entire layer. The activation functions are:

* Sin
* Cos
* Tanh
* Sigmoid
* Smooth ReLu

The last two activation functions are available for the output layer of the CPPN. They result in values ranging only from 0 to 1.

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 model is saved and the template model is reopened. The code continues on to the next model.

Cases

Case 1 – Uniaxial elongation

Case 1 is a case of uniaxial 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 1 – Biaxial elongation

Case 1 is a case of biaxial elongation. This case has applications in causing expansion.

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 | Forced displacement | x |

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

Case 3 – Pure shear

Case 3 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 4 – Elongation of one side

Case 4 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.

Performance Measures

Performance measures are reliant on the case being evaluated.

Case 1

Two performance measures are identified for case 1. The height to width ratio is inspected. The larger the ratio, the greater the relative extension of the unit. The change in width is inspected. It is desired that the width remain as constant as possible.

The height and width of the deformed shape are determined by fitting a horizontal and vertical curve through the external node displacement coordinates respectively.

Case 2

Four similar performance measures are identified for case 2. The height and width of the deformed shape are determined by fitting a horizontal and vertical curve through the external node displacement coordinates respectively. The Hausdorff distance for each side’s coordinates to the fitted curves is calculated. It is desired that the unit retain its square shape while extending in the x and y directions. The smallest Hausdorff distance from a straight line indicates the smallest deviation in shape.

Ranking

Unit performance measures are weighted equally. In order to compare them equivalently, all performance measures are normalized according to Equation

Where

is the fitness value

is the performance measure

is the mean of all units’ performance measure values

is the standard deviation of all units’ performance measure values

All fitness values applicable to a unit are summed together. Units are ranked in descending order according to the fitness value. The unit with the largest fitness value is deemed the most fit.