Table of Contents

Table of Figures

Introduction

Additive manufacturing is an area of design and manufacture that has seen rapid progress in recent years. There are promising applications for this technology in various industries including Automotive, Aerospace and Medical where there is currently the biggest drive for research and development in an industrial context. The proliferation of many low-cost consumer 3D printers on the market has also helped this technology to penetrate into the mainstream, which further demonstrates that interest is rapidly increasing in all commercial sectors.

However, the development of parts and assemblies that fully leverage the advantages of additive manufacturing have been hindered partly by traditional design paradigms that are used by designers, which are more geared towards conventional manufacturing methods such as milling and casting. In addition, optimal Design for Additive Manufacturing (DfAM) imposes certain limitations and constraints that are not immediately obvious to designers. It is difficult to retrain the intuitions of designers to recognise best practices in DfAM, especially when it comes at the cost of limiting the designer's freedom and creativity.

Generative Design is a design paradigm which involves generating many (sometimes many thousands of) variations on a baseline design for a part, while still incorporating its key functional constraints and requirements. The most optimal designs can be chosen either by the user, or automatically using a fitness algorithm. This is an also a novel area that is rapidly growing in interest, particularly due to the rapid proliferation and commoditisation of powerful computing hardware which has made this more accessible to the end user.

The goal of this project is to develop a CAD tool or plug-in that leverage generative design in a way that supports designers in producing designs which are better optimised for Additive Manufacturing. This is envisioned to be a kind of plug-in tool for a CAD package which supplements a designer's workflow and empowers them to improve their designs in a flexible way that doesn't limit their freedom. The hope is that this will build a foundation to demonstrate how generative design has a firm spot in the future of mechanical design. In particular, this potential will shine when new opportunities and constraints are presented from industry paradigm shifts.

Background Literature Review

Additive Manufacturing

Additive manufacturing has immense potential to produce complex parts in more efficient ways than traditional processes, however this is highly dependent on how Design for Additive Manufacturing (DfAM) is incorporated into the design process. Seepersad (1) states that in order for this to be successful, DfAM tools need to be more accessible for both expert and novice designers. Designers have a powerful tendency to adhere to designs they have seen previously, which have been subject to traditional Design for Manufacturing (DfM) guidelines. In particular, most CAD tools are not well suited to representing AM parts, which usually involve starting with primitive objects like cubes and cylinders then subtractively removing features. It is suggested that CAD software needs to be supplemented by tools that can support DfAM knowledge.

Constraints

The material used for additive manufacturing methods is comparatively more expensive than the bulk material it is made from. This is because it needs some processing before it is suitable for AM (e.g powder for Laser Bed Fusion, curable liquid for Stereolithography, thermoplastic filament for Fused Deposition Modelling).

Additionally, printing parts is generally more time consuming as small amounts of material are being expelled layer-by-layer. Both factors means that part volume is a critical factor in efficiency and cost-effectiveness, especially in comparison to subtractive methods. Lower volume means lower cost of materials and lower fabrication time equalling a higher rate of production.

Parts are almost always constructed layer-by-layer from the bottom upwards. For most AM processes, this is a disadvantage for parts that have overhanging features. If the feature is inclined from the horizontal at an angle less than a critical value (usually around 45 degrees), material is required underneath to support it. If done inefficiently, this can be costly as it consumes time and material. Furthermore, the removal of support structures after printing adds another manufacturing step which increases process complexity and introduces another. Choosing a build orientation that results in low-lying and minimal overhanging features helps to mitigate this, but this can come at the cost of build quality (2).

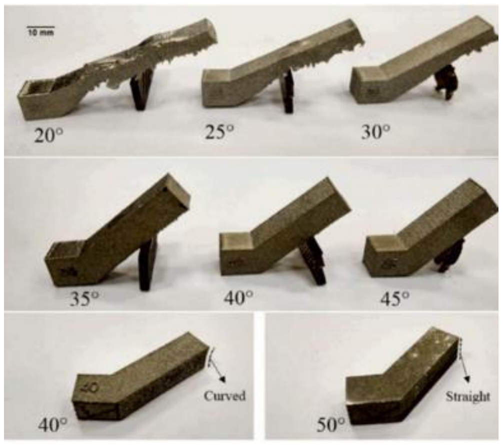


Figure 1 - 3D printed overhang at different inclination angles (3)

Both of these characteristics are not usually kept in mind when designing parts. Contemporary parametric CAD design does not encourage optimisation of part volume and overhanging features. Thus, one of the goals of this project will be to drive optimal support structure design.

Opportunities

Salem et al (2) discusses the use of lattice structures to enhance mechanical performance. They can provide high strength-to-weigth ratios, high heat transfer capacity, thermal insulation, and energy absorption. Latticing is especially suited to additive manufacturing since the small struts are much easier to produce by depositing small pieces of material than through subtractive methods. Foam, honeycomb, lattice, and other constructs of cellular structures were explored. This method has been used successfully in aerospace, biomedical and automotive industries.

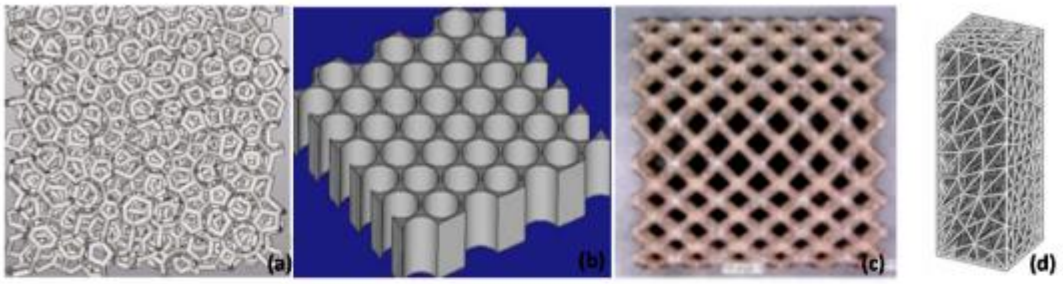


Figure 2 - Different types of cellular topologies (4)(5)(6)(7)

Part consolidation is a useful strategy for reducing the number of parts in an assembly by combining them into a single part. AM is particularly well suited to this because of its ability to manufacture a wide range of material properties through functional structures and composites (2).

Part complexity is something that also scales well with additive manufacturing. Adding new parametric features to a part design generally introduces complexity to traditional processes like milling and casting, thus introducing new steps to the manufacturing pipeline. Extra care often needs to be taken to also make sure it is still manufacturable. Conversely, the addition of pockets and extrusions to parts in AM is much more efficient as this simply involves printing or not printing material around adjacent material. This opens a wide range of opportunities. Structures and features can now be built inside enclosed walls and shells which is impossible with subtractive manufacturing and die casting.

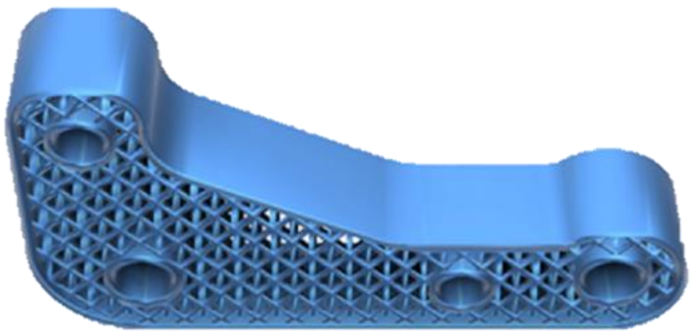


Figure 3 - Latticed part surrounded by thin walls (8)

Structural complexity in AM parts has also been leveraged for a variety of other functions. Some designers have used them to make aesthetic and ergonomic products (2). Features that have been incorporated include integrated air ducts (2), compliant mechanisms (9), and conformal cooling channels to minimise deformation during manufacturing (10).

Generative Design

One of the biggest challenges with integrating generative design into CAD software is making it flexible, intuitive, and accessible for designers. Like additive manufacturing, rapid progress had been made in the field in the last decade but there is still a lot of work to be done to ensure that designers embrace this.

Commercial solutions

Autodesk has been the most steadfast commercial company in embracing generative design in its software suite. Their lightweight CAD package Fusion 360 has a generative design utility that works by constructing obstacle geometry space in red and preserved geometry space in green. The mechanical loads and constraints are then marked normally like they would be in an analysis software package like ANSYS. The solver then iteratively computes generations and tries to converge to an optimal solution.

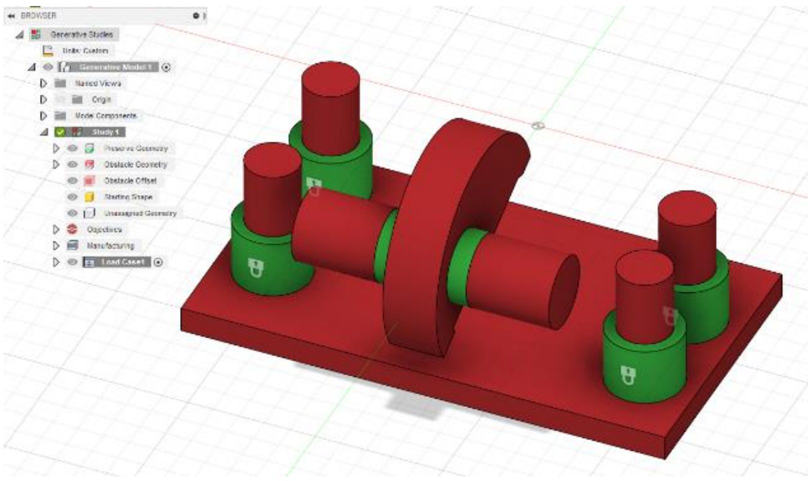


Figure 4 - Screenshot of interface for Generative Design tool in Fusion 360

This tool is well integrated into the Fusion 360 software package and is intuitive in a way that makes it seem integrated into the user interface. However, the workflow is still completely different from traditional parametric CAD, which means that if designers want to apply the process to their existing designs, they would have to evaluate the constraints and regions of their part from scratch. In addition, familiarising oneself with the steps for this procedure requires some training. These factors are enough to put off a designer from using this as an assistive tool, as the effort required outweighs the perceived benefit.

Experimental and academic solutions

Gunpinar et al (11) describes an experiment where neural networks and principal component analysis (PCA) machine learning models are used to generate a side silhouette of a car with an optimal drag coefficient. Finding the drag coefficient of a model requires either doing computational fluid dynamics (CFD) or wind tunnel testing, both of which can be expensive and time consuming. Performing these tests on thousands of different objects is impractical, especially for designs that have only small variations.

In their experiment, a few car silhouette designs are produced and parameterized using lines and Bezier curves. Each design's drag coefficient is computed using CFD and these results are used as training data for the machine learning models. With a sample of 1000 silhouettes, training took up to 12 days on a single workstation PC. The average error in drag coefficient obtained was 10%.

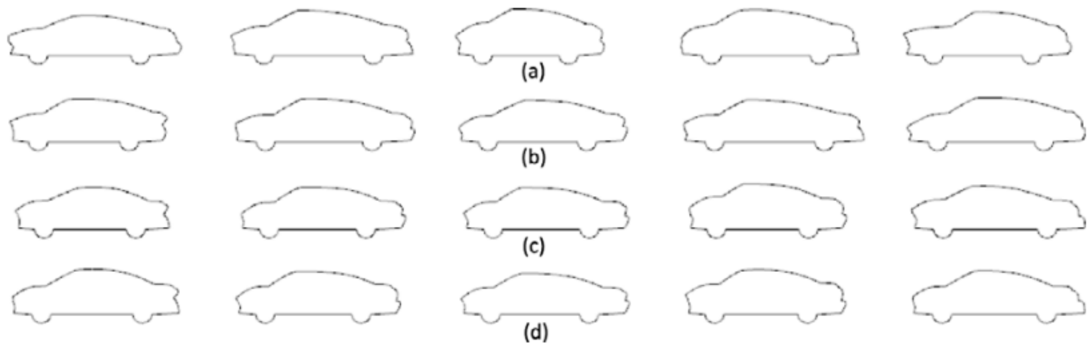


Figure 5 - Generated car silhouettes from Gunpinar et al.(11)

Singh et al (12) attempted to produce an integrated generative design framework for architectural design. A literature review was carried out on five generative different techniques. Namely these are: shape grammars, L-systems, cellular automata, swarm intelligence, and genetic algorithms. The overall consensus was that these techniques have very different strength and weaknesses, and thus there is no single technique that is optimal and general-purpose enough for all design cases. Because of this, it is important that designers are somewhat aware of innerworkings behind these generative methods, so that they know how best to leverage them in their design processes.

Krish (13) expands on the idea of designer-aware generative design by proposing a methodology that wors well with the way designers currently utilise CAD. He argues that analysis and optimisation works best when design intentions have been clarified, and basic shape and form has emerged. CAD doesn't provide the same creative stimulation as sketching, so it is inefficient at representing a very wide range of concepts simultaneously.

His proposed solution tries to encourage design emergence – reflecting on and drawing inspiration from successive iterations of design. The tool relies on the designer knowing how information is exchanged between stages of the generative process. The user can easily tweak almost any variable to get their desired result and can stop at any stage when they are satisfied with the level of variation.

Generative Design for Additive Manufacturing (G-DfAM)

There is not much literature on the combination of both these areas, given how novel they both are. The little research that was found was used to find gaps in the proposed solutions.

Briard et al (14) proposes a four stage methodology, which was determined from a mix of methods that six specialists used to redesign a specimen part. The methodology consisted of two stages of generation – one for initialisation where the solution space is explored completely unconstrained, and another for optimisation with respect to AM guidelines. The refinement stage is another iterative stage which tries to squeeze the best attributes from the base design.

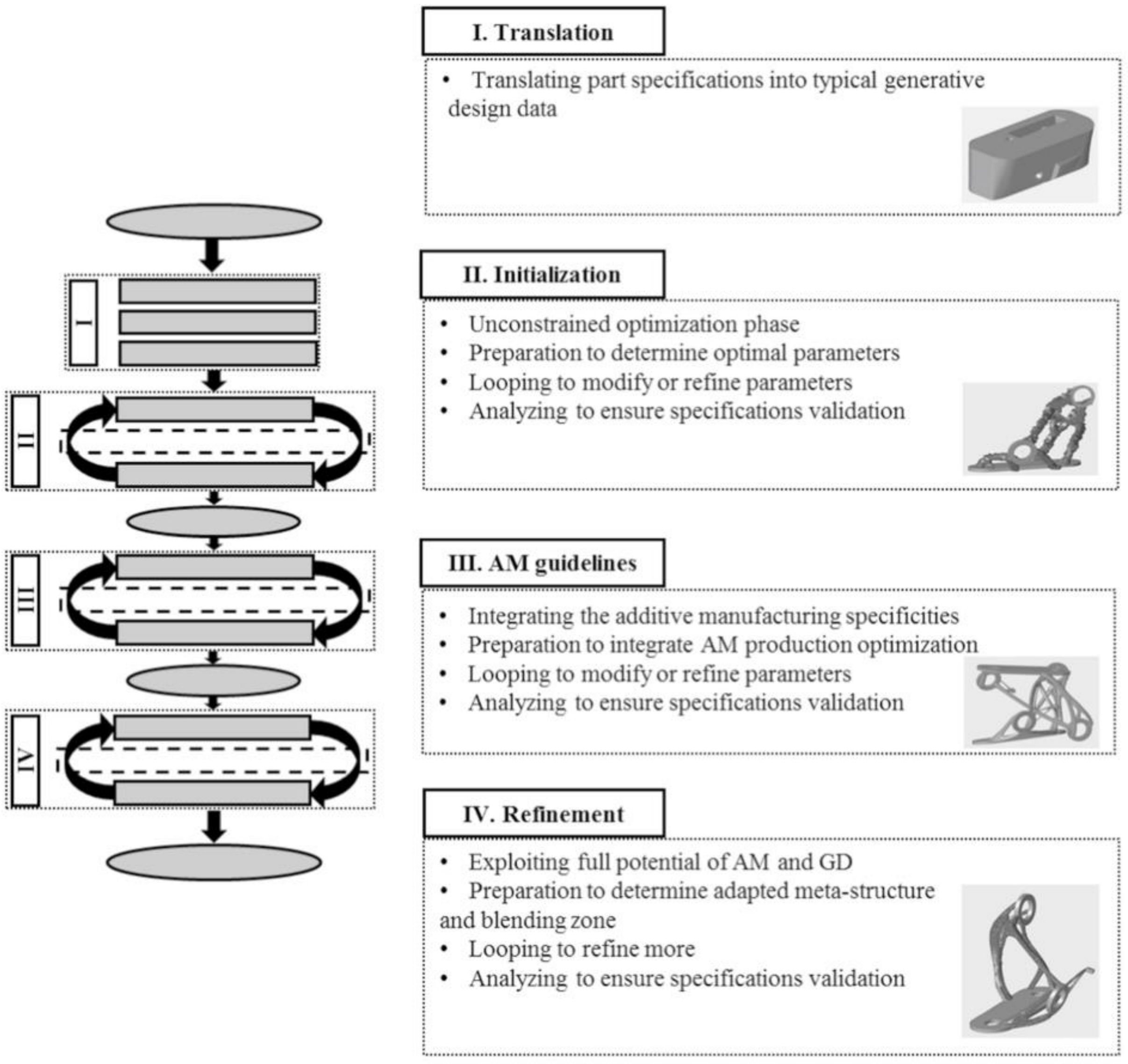

I. Translation
• Translating part specifications into typical generative design data
II. Initialization
• Unconstrained optimization phase
• Preparation to determine optimal parameters
• Looping to modify or refine parameters
• Analyzing to ensure specifications validation
III. AM guidelines
• Integrating the additive manufacturing specificities
• Preparation to integrate AM production optimization
• Looping to modify or refine parameters
• Analyzing to ensure specifications validation
IV. Refinement
• Exploiting full potential of AM and GD
• Preparation to determine adapted meta-structure and blending zone
• Looping to refine more
• Analyzing to ensure specifications validation
I
II
III
IV

Figure 6 - Proposed Generative Design for Additive Manufacturing (G-DfAM) methodology in Briard et al.(14)

Zimmerman et al (15) uses 3D spatial grammars with evaluation through finite element analysis (FEA) to generate an optimal part. Their method could produce a set of diverse yet structurally optimised designs which efficiently conformed to additive manufacturing constraints. It is proposed as a simulation-driven design alternative to topology optimisation.

Xue et al (16) uses machine learning models to produce optimal multi-material composites for 3D printing. Most other methods approach this by parameterising every voxel in the model, which leaves too many open variables. Instead, the lattice structure is constructed from gallery of cellular units called 'representative volume elements'. For each unit, a lattice is produced and the Young's modulus, shear modulus and Poisson's ratio are calculated. This data is used to train a PCA (principal component analysis) model.

Requirements and Process Scoping

For this tool, the methodology proposed by Briard et al (14) was compressed into three stages – Initiation, Generation and Refinement. Because this tool is designed to supplement a designer's workflow, it is envisioned that the designer will use this tool after the fundamental features have been defined, and particular details just need tweaking and optimisation. Thus, there is no need to have an unconstrained stage of generation for exploring the solution space. Unconstrained exploration in the beginning stages is also discouraged by Krish in his methodology (13).

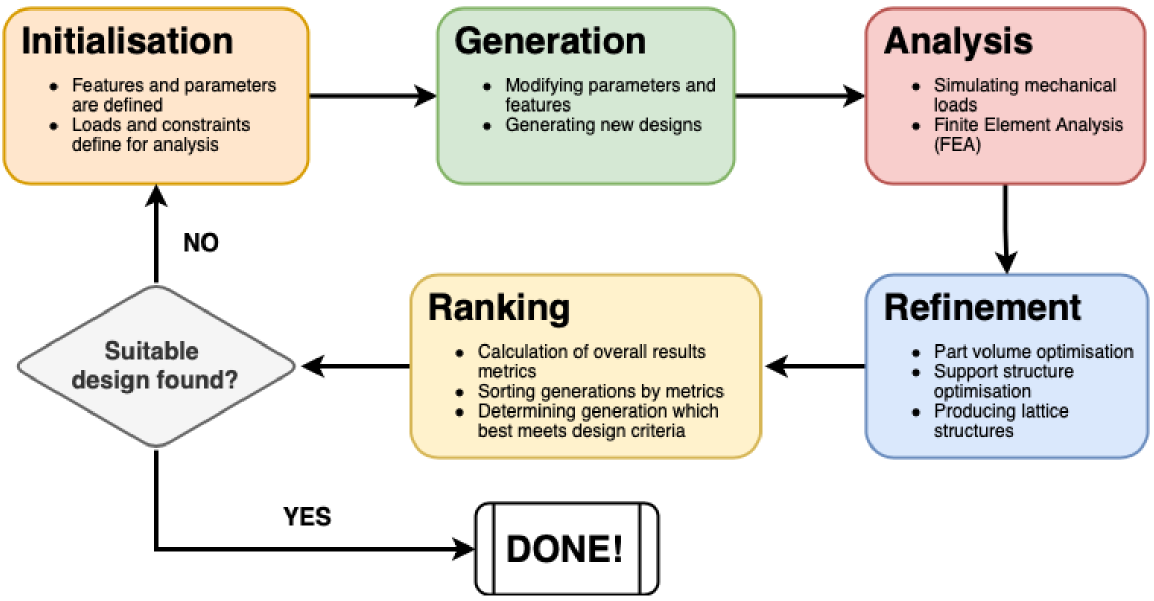

Initialisation
Features and parameters are defined
Loads and constraints define for analysis
Generation
Modifying parameters and features
Generating new designs
Analysis
Simulating mechanical loads
Finite Element Analysis (FEA)
Refinement
Part volume optimisation
Support structure optimisation
Producing lattice structures
Ranking
Calculation of overall results metrics
Sorting generations by metrics
Determining generation which best meets design criteria
Suitable design found?
NO
YES
DONE!

Figure 7 - Five stage methodology for the process scope of the tool

Initiation

To begin the process of generation and analysis, an interface needs to be defined between the design space and the tool so that it knows where and what the user would like to optimise. The most crucial piece of information for any generative process is knowing which features and parameters can be modified. These can include sketch dimensions, feature dimensions, pattern parameters, etc.. Then, the mechanical loads and constraints need to be defined so that there is sufficient information to perform a mechanical analysis later on.

One of the goals of this tool is to be open and transparent so that designers can clearly see how this tool fits into their design problem.

Krish (13) approached the initiation stage by having the user directly input the part parameters into an Excel sheet with value ranges, where these parameter values are regenerated before being fed back into the CAD program to produce the generations. This is a very open and transparent approach that naturally requires the user's attention through the whole process..

Conversely, the generative design tool in Fusion 360 is highly autonomous. All designers have to do is mark the preserved geometry and the mechanical loads and constraints. The tool then attempts to produce designs in what appears to be an iterative process until part volume is stochastically optimal while meeting the designer's specifications. While this is less tedious than the system proposed by Krish (13), it lacks the flexibility that would make it useful in many design optimisation scenarios. Many part designs have explicit relationships between parameters that can't be represented in the tool. Moreover, defining the required and blocking geometry is a non-trivial task that requires significant thought on the designer's part.

Generation

This is the stage where multiple part designs are generated using the conditions specified for parametrisation. The tool reads through all the parameters marked for modification and generates new values for them. These values are then assigned to the parameters and the part geometry is recalculated. This process should be autonomous, acting solely on the input parameters from the initiation stage.

The solution presented by Zimmerman et al (15) involves generating designs using grammars, evaluating their fitness using FEA, then using simulated annealing to stochastically find the optimal generation. This was tested on the example of spokes on a skate wheel. However, the solution is not well refined as this is presented more as a proof of concept. The grammars are defined and generated in a program called spapper (17), then passed into Siemens NX for finite element analysis before being optimised via simulated annealing through a Python script.

Refinement

This is the stage where optimisations that are specific to additive manufacturing are performed. For each generation produced, the part will be optimised for part volume and support structures. This will likely include functions such as optimising build orientation, calculating build time, determining manufacturability etc.

Zhang et al (18) uses a bio-inspired procedure to generate tree-like support structures that have less volume and are easier to print and remove. The process is parametric and thus highly controllable, but the generation has a high computational cost which makes it not very ideal for thousands of generations.

Michell structures are sometimes a better alternative to lattice volumes, as they produce structures that are most optimal in volume (9). These structures can generally achieve high strength and buckling resistance at the cost of stiffness (19), since they are designed such that the struts follow the path of maximum strain magnitude.

Voxelisation of the part makes it easier to do refinement operations for AM. This is largely because most AM processes work by depositing small elements in layers, which a voxel model represents very efficiently by its nature. If the resolution of the model equals the resolution of the process, then modifications to the model will naturally conform much better to the process' inherent constraints. Additionally, voxel models simplify the task of counting part volume, especially when there are composites of materials (20).

Vaidya and Anand (21) developed an approach to building support structures that uses cellular structures alongside Dijkstra's shortest path algorithm. This is made especially possible with a voxelised model space. Furthermore, their solution can accommodate constraints for support structure accessibility so that they can be easily removed after manufacturing.

Analysis

In this stage, the generations are analysed to check whether they meet their original functional criteria. In many cases, the criteria is ensuring that the part doesn't exceed a certain amount of stress, deflection, buckling etc. This will likely be done by performing Finite Element Analysis (FEA) on each individual part although solving analytically might be a possible option for certain cases.

Most of the research papers that were investigated use FEA for deciding part fitness. Because this is a computationally expensive task, any optimisations that may reduce the time spent on this stage are highly beneficial. Some groups implemented machine learning models to help classify a correlation between FEA results and part variations so that generations on successive iterations were more likely to have improved fitness.

Opgenoord and Willcox (22) used an analysis procedure where parts are first analysed using a coarse FEM analysis for speed. This is then used to adjust the meshing so that resolution is fine in high stress regions and low in low stress regions. The parts are then analysed properly using this mesh for high accuracy with minimal computational effort.

Ranking

This is the final stage where all of the results from the generations are consolidated and the most optimal generation is selected. If the criterion for generation fitness is simple, like minimise the peak stress or part volume, or keep the deflection with a certain range, then the tool can quite easily determine the best part automatically. For more complicated fitness criteria that involve a combination of several metrics, it should be possible to allow the user to define these metrics as a mathematical function that the tool can then use to rank the parts.

Another helpful feature would be to automatically revise and narrow the parameter space on successive process iterations, so that it better converges towards the desired fitness criteria. This would help to avoid wasting computation and thus time on generations that clearly do not converge towards the criteria. This would be implemented using a machine learning model, likely either a neural network or principal component analysis model as demonstrated in Gunpinar et al (11).

Software choices

Currently the most prominent program available which leverages generative design is Fusion 360 by Autodesk. The best open-source alternative found was FreeCAD, due to its relatively easy interface, extensibility, and its large support community and documentation.

Fusion 360

Fusion 360 is a popular CAD package from Autodesk. It is effectively a lightweight version of their flagship CAD package Inventor Professional. Because of its simplicity and ease of use, it is quite a popular program. Autodesk also usually implement their newest optimisation tools in Fusion 360 before any of their other platforms, so it is a fairly mature environment in terms of its generative design and topology optimisation tools.

Fusion 360 also has good support for scripting and plugins. Their API is well documented and all scripting can be done in either Python or C++. There is lots of information and tutorials within the developer community on getting started with scripting.

Also integrated into the program is an excellent simulation toolbox, with tools for static stress FEA, structural buckling analysis, and topology optimisation just to name a few.

The only major downside to this software is that it is closed-source. In particular, their generative design tool and simulation workspace tools aren’t well suited to being controlled via scripts. Rather, their functionality is only meant to be exposed through the user interface. Additionally, the processes and algorithms that these tools use aren't transparent to the user, which makes extending and customising their functionality difficult.

FreeCAD

FreeCAD is a popular free and open-source CAD package that began in 2002. Because of its large community, it has grown into a very sophisticated product that covers many bases. In addition to mechanical CAD, it also has workspace environments for architecture, civil engineering and electronic engineering.

The functionality and features bear a close resemblance to CATIA, from the keyboard and mouse gestures to the names of the part operations. This was beneficial for myself, as I am an experienced CATIA user so getting comfortable with the environment and producing designs was fairly quick and easy.

There is a large abundance of macros and plugins available for the program that have been developed by its community of users. This is in large part thanks to its open and robust foundation. Like Fusion 360, plugins and macros can be written in C++, Python, or a mixture of both. Unlike Fusion 360, every object, entity, and data structure in FreeCAD is well documented and easily accessible through the C++ and Python APIs, which makes it an excellent platform for experimentation.

FreeCAD is carefully designed from the ground up so that every single operation can be done programmatically through the APIs and without the graphical user interface (GUI). Every function and interaction in the GUI is invoked by calling some function or code snippet in Python, with the code listed inside the Python console window. This made it easy to make a script that would generate a part designed inside the GUI as the relevant commands could be copied from the console window.

Finally, FreeCAD comes packaged with an FEA solver called CalculiX, which is an open-source implementation of Abaqus (23). It is well integrated into the program, as FreeCAD has a workbench which makes it easy to define loads, constraints and conditions in a way that seems natural for a designer. FreeCAD takes care of converting the analysis conditions into an input file for CalculiX to run, and can translate the output file from Calculix into user-friendly results viewable as a colourmap of the part.

CalculiX by itself is unable to generate FEA meshes, so FreeCAD also comes with two mesh generators called GMsh and Netgen. Both are popular and equally capable open-source FEA meshing tools.

Decision Matrix

A decision matrix was constructed to help make a decision between which CAD package to use. The characteristics that are most important for this project are API support, scripting support, and community support as they play a key factor in the progress and success of development over the short available development period.

Table 1 - Weighted decision matrix for choice of CAD package

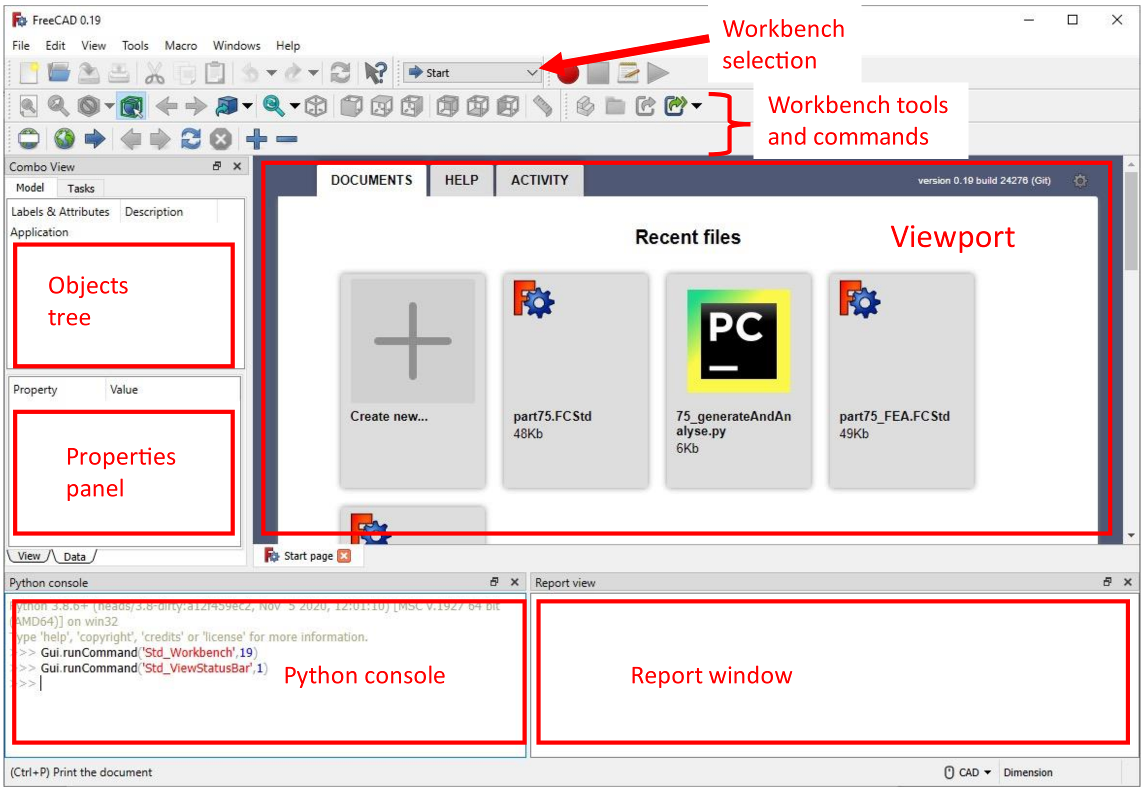
		CAD Package	
Characteristic	**Weighting**	**Fusion 360**	**FreeCAD**
User licence	1	2	5
Generative design tool	2	3	0
Simulation tools	3	5	3
API support	6	3	5
Scripting support	5	4	5
CAD modelling	3	5	3
Ease of use	3	5	4
Community support	4	3	5
Third party plugins	3	2	4
Documentation	2	4	3
	0	**117**	**128**

Because of these important factors, FreeCAD wins as the more suitable decision, despite the better support that Fusion 360 shows in other areas ease of use and simulation tools.

Implementation and Development

Software and Interface

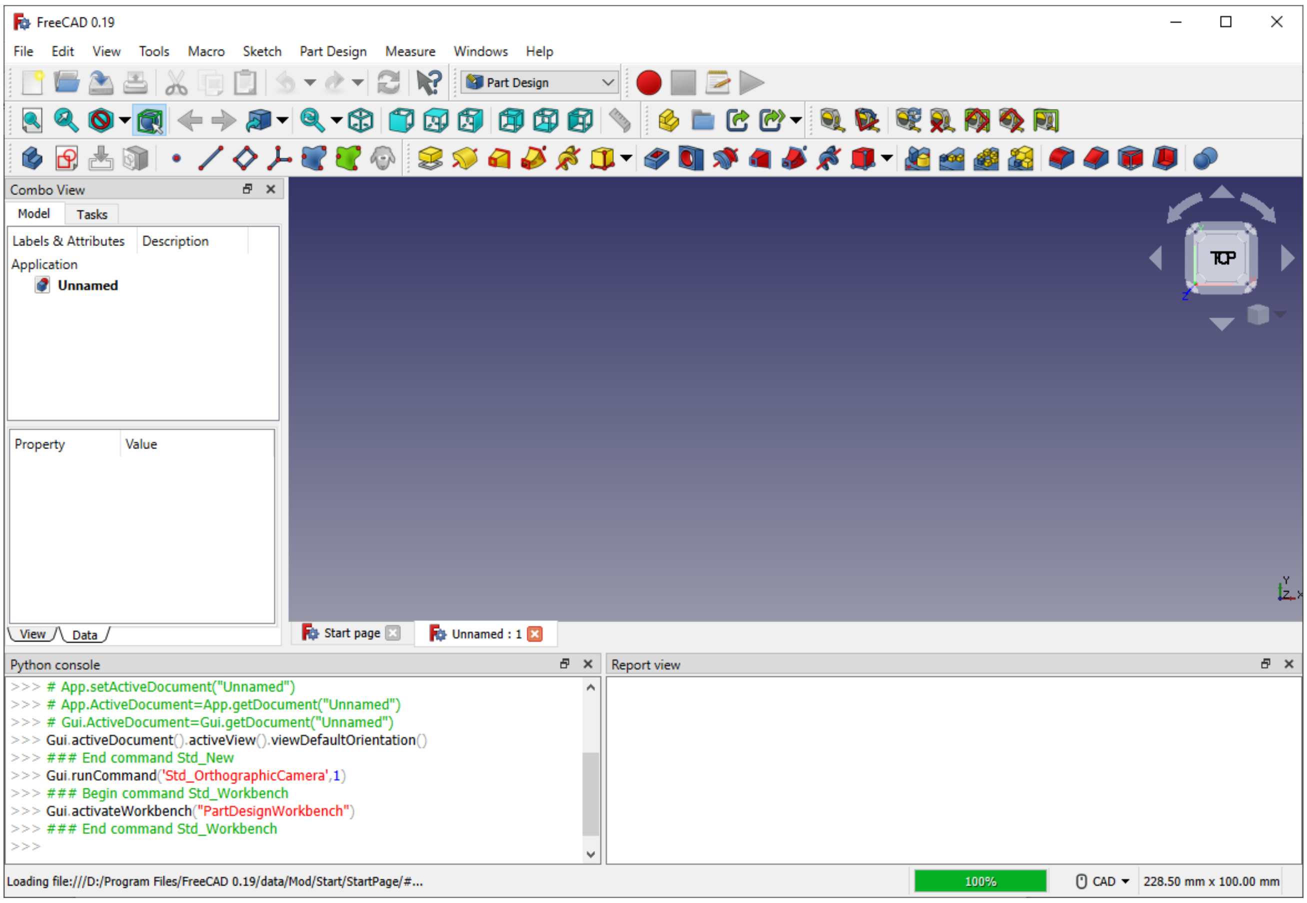
When the user starts up FreeCAD for the first time, they are greeted with the following user interface



```
Python 3.8.6+ (heads/3.8-dirty:a12f459ec2, Nov  5 2020, 12:01:10) [MSC v.1927 64 bit (AMD64)] on win32
Type 'help', 'copyright', 'credits' or 'license' for more information.
>>> Gui.runCommand('Std_Workbench',19)
>>> Gui.runCommand('Std_ViewStatusBar',1)
>>> 
```

Figure 8 - FreeCAD interface on startup

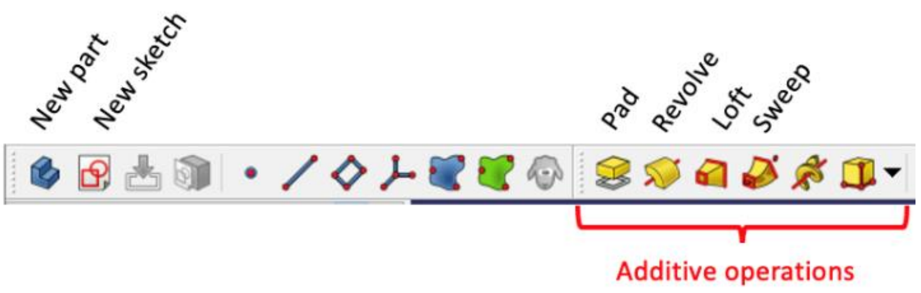
To begin designing a part, the user creates a new document by either going to **File > New**, or clicking on the icon of a paper in the top left corner. A blank canvas appears in the document viewport. The user then clicks on the Workbench drop-down menu and selects **Part Design**. This will expose to the user all of the functions that one would typically expect from a parametric CAD package.

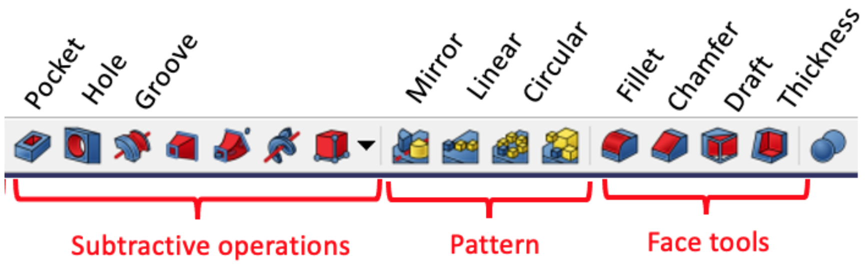


```
>>> # App.setActiveDocument("Unnamed")
>>> # App.ActiveDocument=App.getDocument("Unnamed")
>>> # Gui.ActiveDocument=Gui.getDocument("Unnamed")
>>> Gui.activeDocument().activeView().viewDefaultOrientation()
>>> ### End command Std_New
>>> Gui.runCommand('Std_OrthographicCamera',1)
>>> ### Begin command Std_Workbench
>>> Gui.activateWorkbench("PartDesignWorkbench")
>>> ### End command Std_Workbench
>>>
```

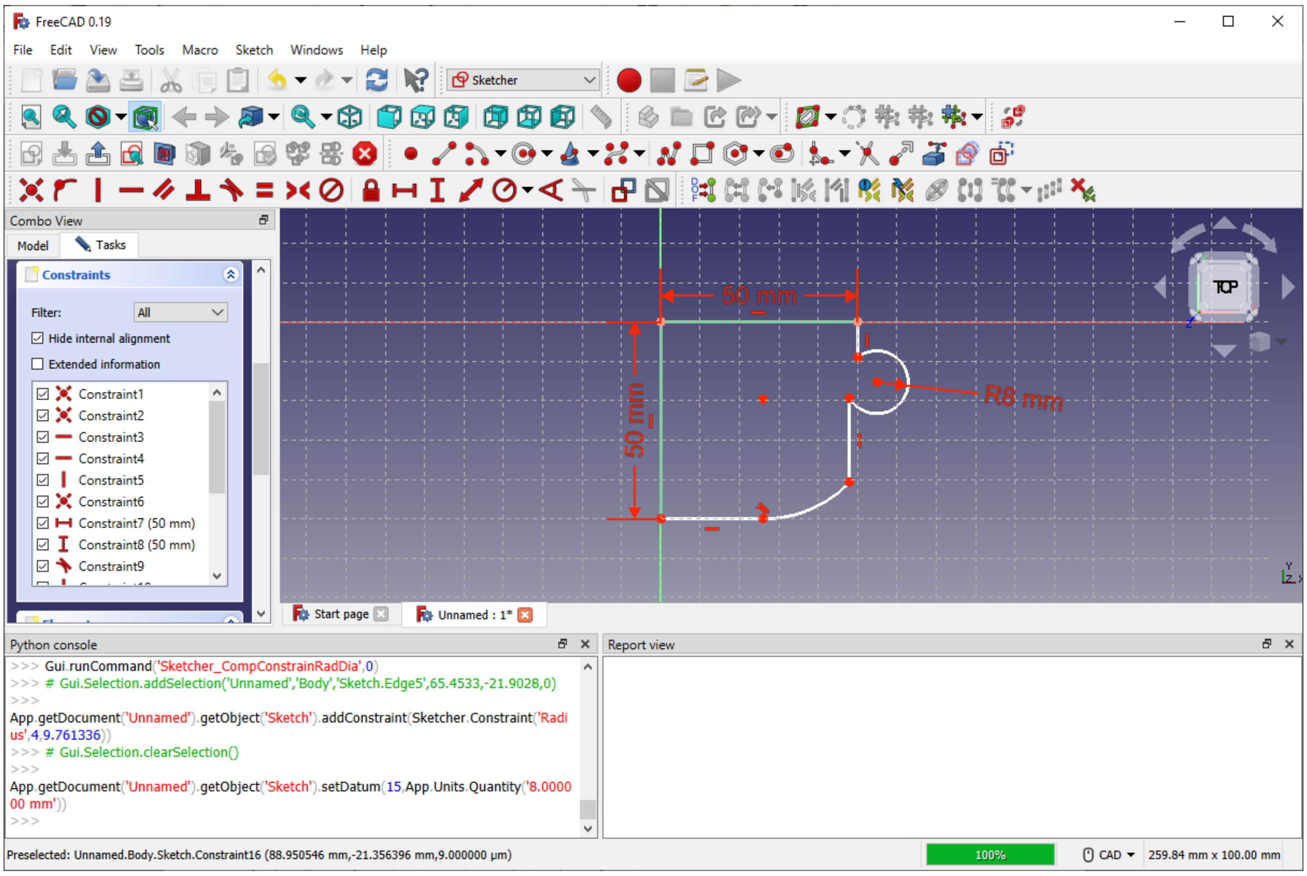
Loading file:///D:/Program Files/FreeCAD 0.19/data/Mod/Start/StartPage/#...

Figure 9 - Part design workbench interface





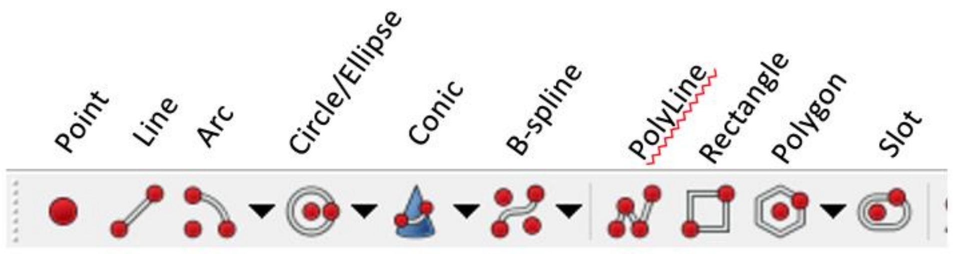
The user starts designing a part by creating a sketch by clicking on the **New sketch** button. The user then selects which one of the three base planes to sketch on, before they are brought into the ‘Sketcher’ workbench.

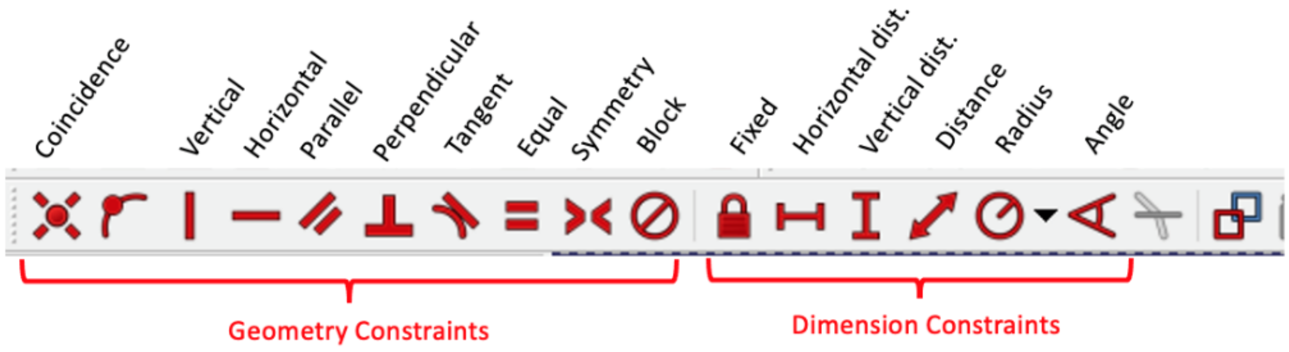


```
>>> Gui.runCommand('Sketcher_CompConstrainRadDia',0)
>>> # Gui.Selection.addSelection('Unnamed','Body','Sketch.Edge5',65.4533,-21.9028,0)
>>>
App.getDocument('Unnamed').getObject('Sketch').addConstraint(Sketcher.Constraint('Radi
us',4,9.761336))
>>> # Gui.Selection.clearSelection()
>>>
App.getDocument('Unnamed').getObject('Sketch').setDatum(15,App.Units.Quantity('8.0000
00 mm'))
>>>
```

Preselected: Unnamed.Body.Sketch.Constraint16 (88.950546 mm,-21.356396 mm,9.000000 µm)

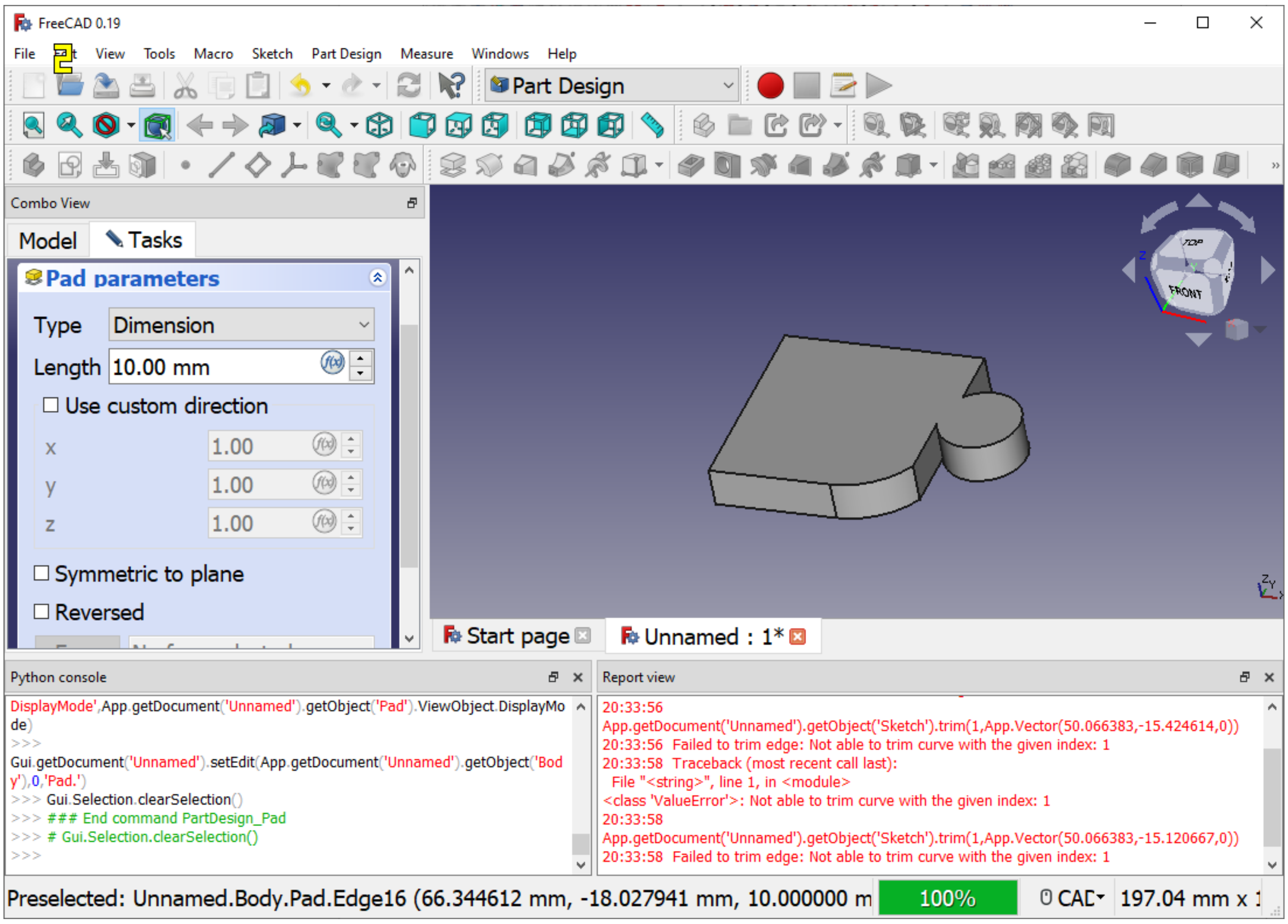
Figure 10 - Sketcher workbench interface





From here, the user constructs the sketch of their part feature in their usual way. Once finished, they can click the **Finish sketch** button which will bring back the **Part design** workbench shown in Figure 9. From here the designer can perform their typical parametric operations on their sketch.

Clicking on the **Pad** operation will extrude the sketch and open up the following interface



Pad parameters

Type: Dimension
Length: 10.00 mm
☐ Use custom direction
x: 1.00
y: 1.00
z: 1.00
☐ Symmetric to plane
☐ Reversed

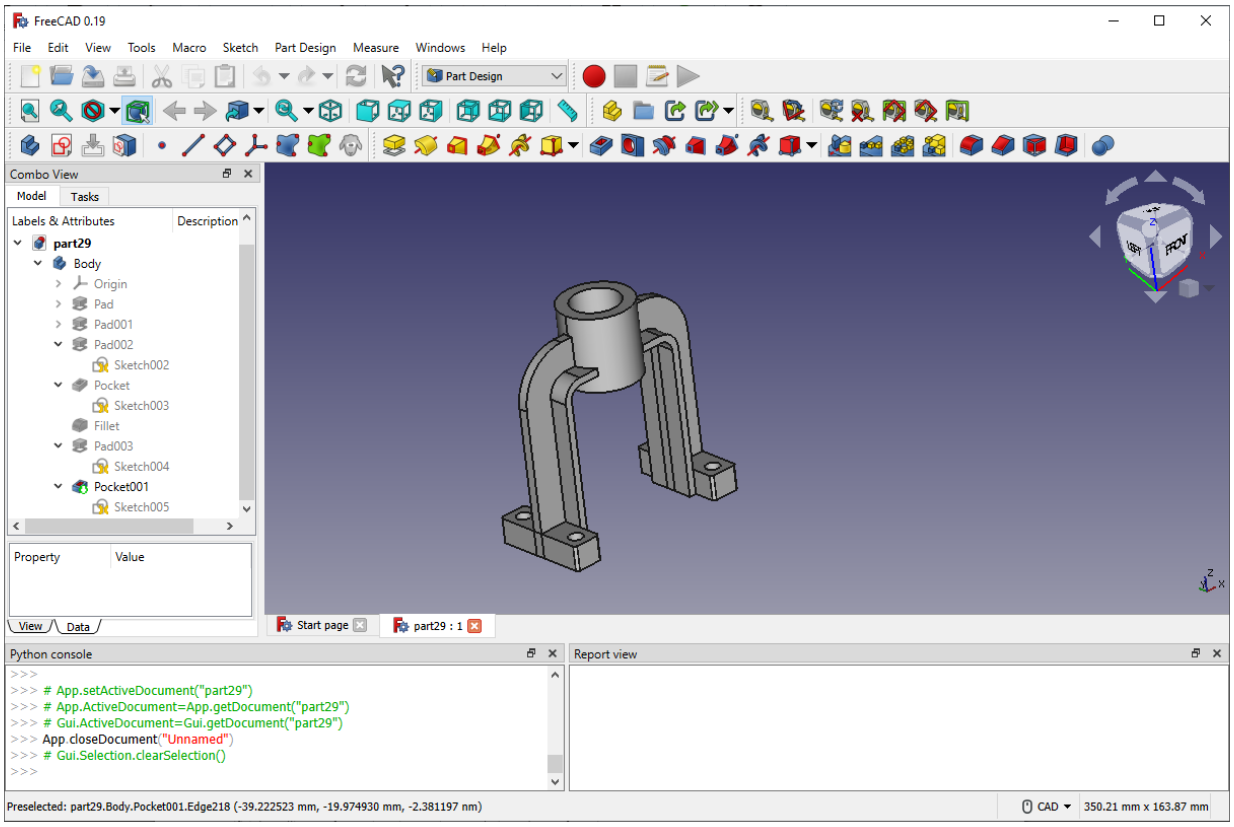
```
DisplayMode',App.getDocument('Unnamed').getObject('Pad').ViewObject.DisplayMo
de)
>>>
Gui.getDocument('Unnamed').setEdit(App.getDocument('Unnamed').getObject('Bod
y'),0,'Pad.')
>>> Gui.Selection.clearSelection()
>>> ### End command PartDesign_Pad
>>> # Gui.Selection.clearSelection()
>>>
```

```
20:33:56
App.getDocument('Unnamed').getObject('Sketch').trim(1,App.Vector(50.066383,-15.424614,0))
20:33:56  Failed to trim edge: Not able to trim curve with the given index: 1
20:33:58  Traceback (most recent call last):
  File "<string>", line 1, in <module>
<class 'ValueError'>: Not able to trim curve with the given index: 1
20:33:58
App.getDocument('Unnamed').getObject('Sketch').trim(1,App.Vector(50.066383,-15.120667,0))
20:33:58  Failed to trim edge: Not able to trim curve with the given index: 1
```

Preselected: Unnamed.Body.Pad.Edge16 (66.344612 mm, -18.027941 mm, 10.000000 m   100%   CAD   197.04 mm x 1

*Figure 11 - **Pad** command interface*

Figure 12 below shows a typical part and its constituent features in the ‘Model’ panel.



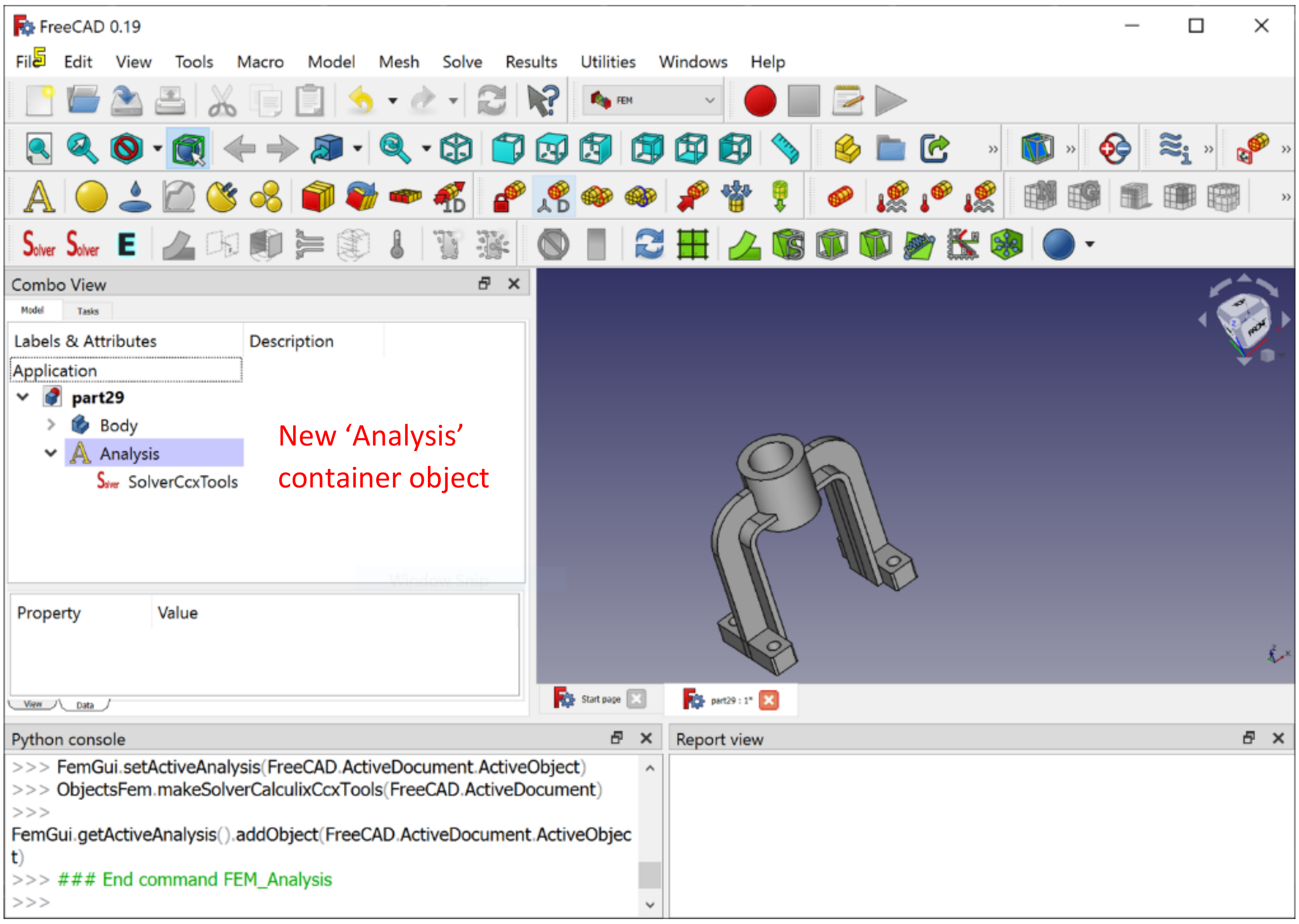
```
>>>
>>> # App.setActiveDocument("part29")
>>> # App.ActiveDocument=App.getDocument("part29")
>>> # Gui.ActiveDocument=Gui.getDocument("part29")
>>> App.closeDocument("Unnamed")
>>> # Gui.Selection.clearSelection()
>>>
```

```
Preselected: part29.Body.Pocket001.Edge218 (-39.222523 mm, -19.974930 mm, -2.381197 nm)
```

Figure 12 - Typical part model in the FreeCAD interfac

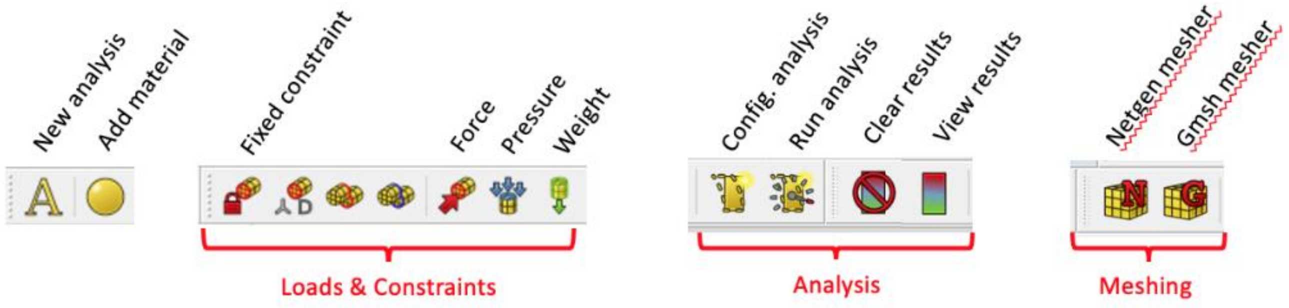
FEA Interface

To begin setting up a part for FEA analysis, the user changes the workbench to **FEM**, resulting in the interface shown in figure 13.



```
>>> FemGui.setActiveAnalysis(FreeCAD.ActiveDocument.ActiveObject)
>>> ObjectsFem.makeSolverCalculixCcxTools(FreeCAD.ActiveDocument)
>>>
FemGui.getActiveAnalysis().addObject(FreeCAD.ActiveDocument.ActiveObjec
t)
>>> ### End command FEM_Analysis
>>>
```

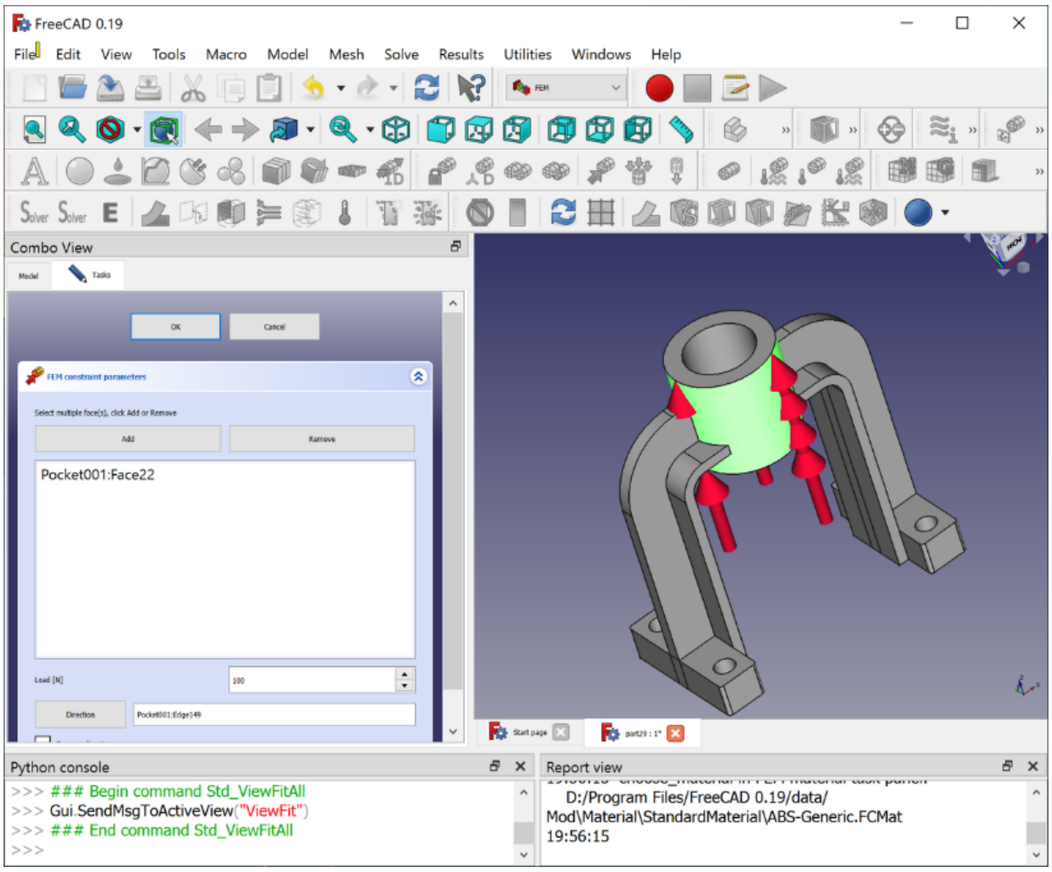
Figure 13 - Analysis workbench interface



Because the analysis workflow is separate from the design workflow, an 'Analysis' object container needs to be made to contain all of the necessary objects and information. This is done by clicking on the **New analysis** command.

Next, the user needs to define the material properties. The user clicks on the **Add material** button which looks like a yellow sphere. From here, the user can select from a database of materials included with FreeCAD to define the material properties.

The user then has to define the loads and constraints of the part to be used in the analysis. In nearly all cases, the three types mainly used are the **fixed constraint**, **static force** and **static pressure**. These are added to the appropriate parts by clicking on the faces that the constraints will act on, then clicking on the corresponding command. Figure 14 below shows an example where a static load of 100N is added to the curved outer face of the part.



Pocket001:Face22

Load [N] 100

Direction Pocket001:Edge149

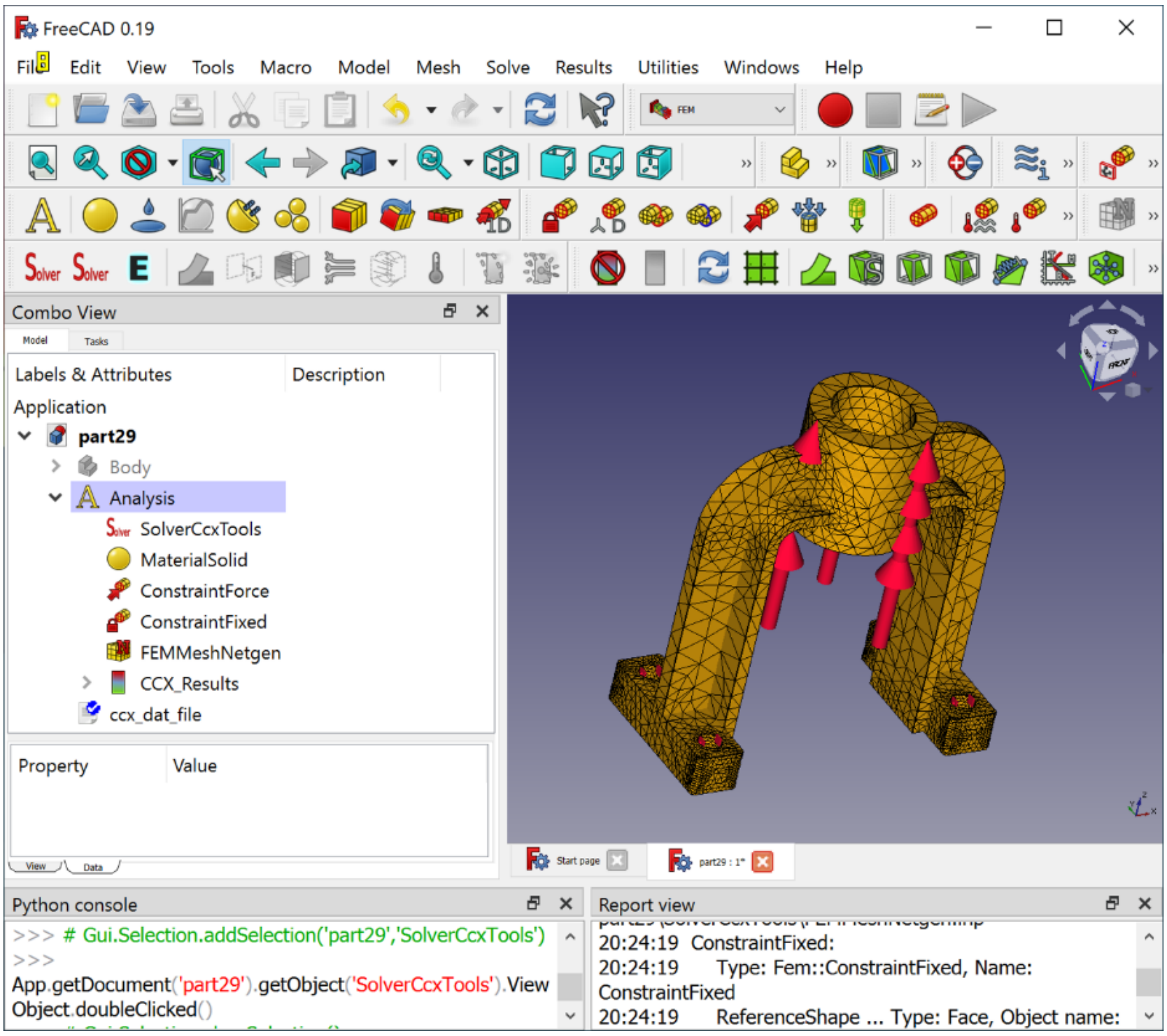
```
>>> ### Begin command Std_ViewFitAll
>>> Gui.SendMsgToActiveView("ViewFit")
>>> ### End command Std_ViewFitAll
>>>
```

```
    D:/Program Files/FreeCAD 0.19/data/
Mod\Material\StandardMaterial\ABS-Generic.FCMat
19:56:15
```

Figure 14 - Adding a force load condition to a part

After adding at least one load and one constraint, the part is ready for meshing in preparation for performing FEA. The user has the choice between using **Gmsh** and **Netgen** for producing the part mesh. Both are quite similar so it generally doesn’t matter which one is chosen. Finishing this step adds a mesh object to the analysis container.

The final step is to perform the actual analysis. This is done by clicking on the solver object that was created with the analysis container, then clicking on the **Start Calculations** command. Once all of these steps have been followed, the FreeCAD environment will look like figure 15.

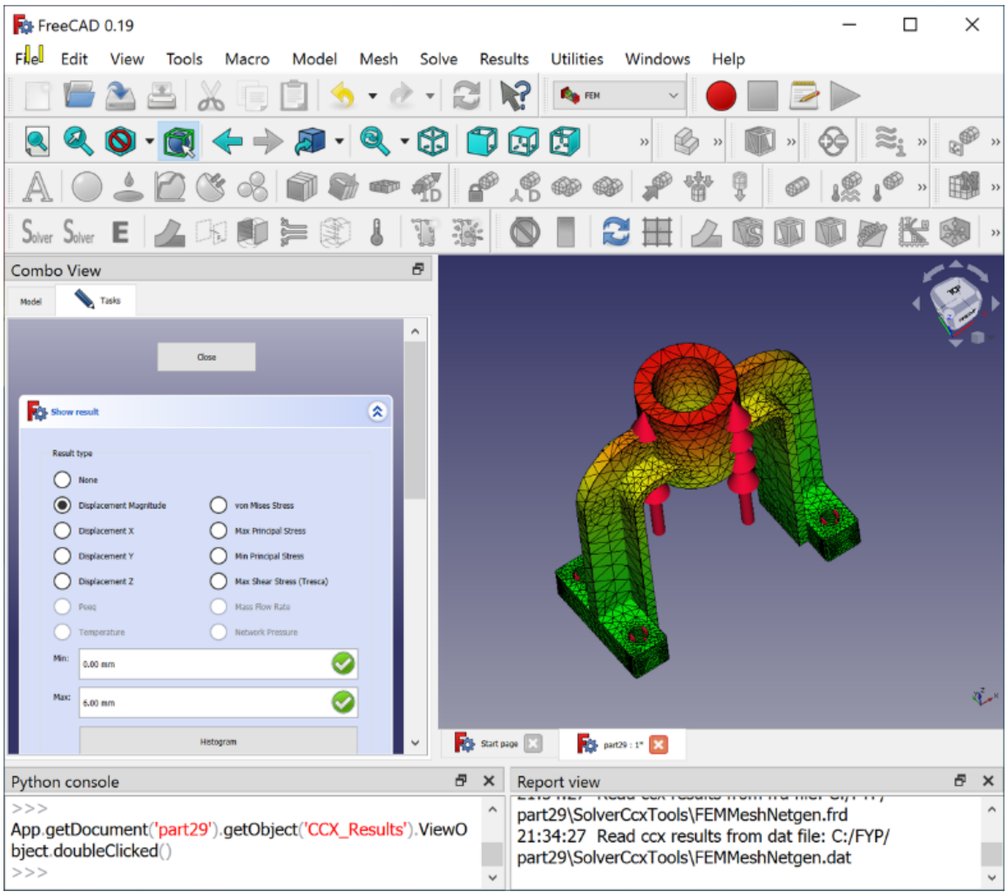


```
>>> # Gui.Selection.addSelection('part29','SolverCcxTools')
>>>
App.getDocument('part29').getObject('SolverCcxTools').View
Object.doubleClicked()
```

```
20:24:19  ConstraintFixed:
20:24:19      Type: Fem::ConstraintFixed, Name:
ConstraintFixed
20:24:19      ReferenceShape ... Type: Face, Object name:
```

Figure 15 - Interface after a completed FEA analysis

Double-clicking on the **CCX_Results** object will show a colourmap of the results inside the CAD viewport. By default, the solver will analyse for the static stress and displacement distribution through the part.



Result type

- None
- Displacement Magnitude [✓]
- Displacement X
- Displacement Y
- Displacement Z
- Peeq
- Temperature
- von Mises Stress
- Max Principal Stress
- Min Principal Stress
- Max Shear Stress (Tresca)
- Mass Flow Rate
- Network Pressure

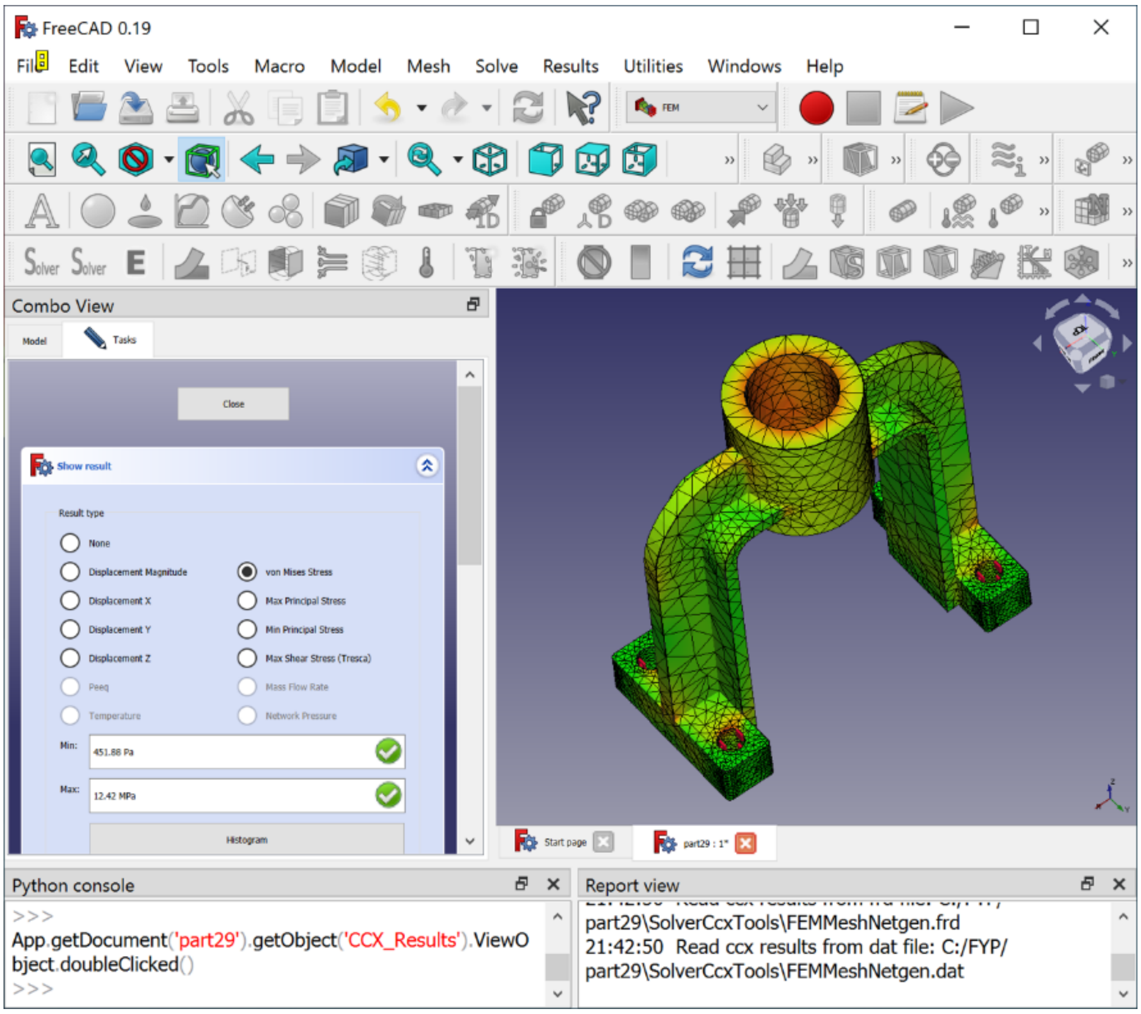
Min: 0.00 mm

Max: 6.00 mm

```
>>>
App.getDocument('part29').getObject('CCX_Results').ViewObject.doubleClicked()
>>>
```

```
part29\SolverCcxTools\FEMMeshNetgen.frd
21:34:27  Read ccx results from dat file: C:/FYP/
part29\SolverCcxTools\FEMMeshNetgen.dat
```

Figure 16 - Colourmap of displacement in part



Result type

None [ ] Displacement Magnitude [ ] Displacement X [ ] Displacement Y [ ] Displacement Z [ ] Peeq [ ] Temperature [ ]
[✓ von Mises Stress] Max Principal Stress [ ] Min Principal Stress [ ] Max Shear Stress (Tresca) [ ] Mass Flow Rate [ ] Network Pressure [ ]

Min: 451.88 Pa

Max: 12.42 MPa

```
>>>
App.getDocument('part29').getObject('CCX_Results').ViewO
bject.doubleClicked()
>>>
```

```
part29\SolverCcxTools\FEMMeshNetgen.frd
21:42:50  Read ccx results from dat file: C:/FYP/
part29\SolverCcxTools\FEMMeshNetgen.dat
```

Figure 17 - Colourmap of von-Mises stress in part

Scripting

Executing code in Python can be done in two ways. Firstly, lines of code can be executed directly in the Python console where code is interpreted on the fly. Code will execute every time the enter key is pressed and the results are immediately visible in the GUI. This is great for rapid prototyping and trying out new ideas.

Secondly, lines of code can be written in bulk into a .py script file then run directly from the interface. This is useful for running a whole process involving many lines of code. FreeCAD's macro system is also built on top of Python scripting. When the user starts recording a macro, the commands produced in the console from user's actions are copied into a text buffer until the user selects stop, then the buffer is saved to a Python script.

Python script files can be opened and edited within the program as code, with simple quality of life features like line numbering and syntax highlighting. However, it lacks some useful features like a monospaced font, syntax suggestions, and a debugger. For this reason, much of the heavy coding work was done in the PyCharm IDE.

Testing and Prototyping

To make development generally as iterative and rapid as possible, the process tasks that needed making were modularised into their own ‘test milestones’. Each milestone was developed in iterations in an exploratory nature without fixed goals to see how far they could be developed.

Test Milestone 1: Modifying Part Parameters

To test the functionality of the scripter, a part was made with a small variety of features to test the ability to modify parameters.

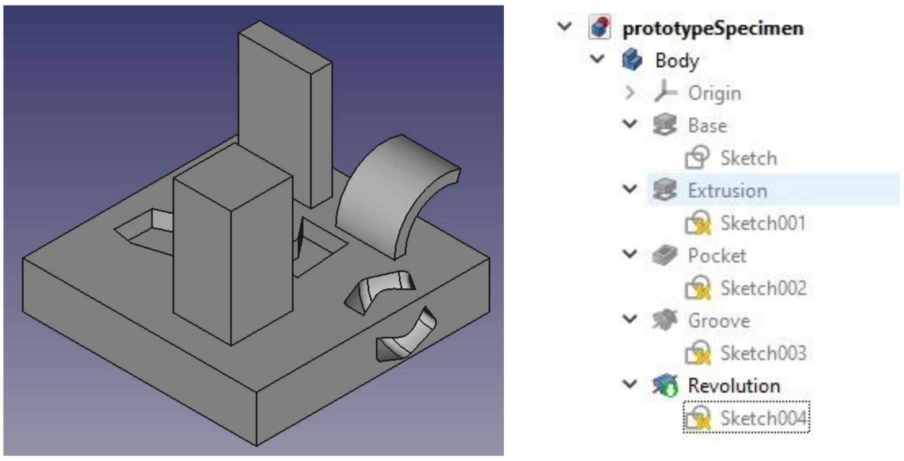


Figure 18 - Specimen part for testing with corresponding feature tree

The part has a number of Pad, Pocket, Revolve and Groove features. Constraints were assigned to features of the sketches that were to be parameterised. The code to edit these constraints was then copied into a script file. Appendix A shows the sketches and features of the part in detail with the parameters that were modified. Appendix B shows the code that was written to test the modifiability of these parameters.

The script was able to successfully modify the parameters that were marked and the part changed accordingly. By passing this test, two things are confirmed:

- Constraint and feature dimensions can be modified easily from Python
- Values with standard Python data types (int, float) can be assigned to constraints

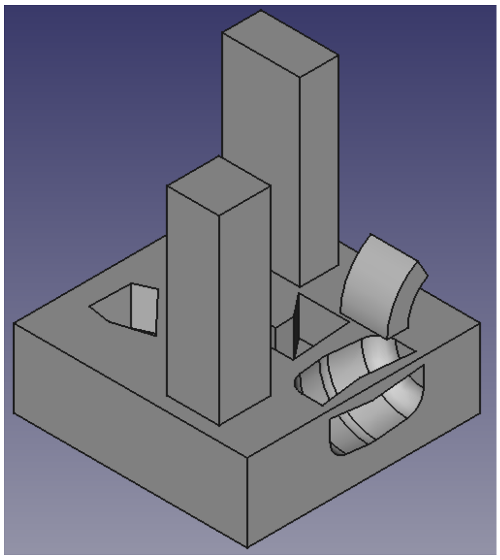


Figure 19 - Regenerated specimen part

While playing around with the parameter assignment code, a few issues were run into. If values were assigned that lead to invalid geometry (e.g sketch lines that were outside the sketch region) then FreeCAD would assign these values, but the part geometry would not be updated. Worse yet, trying to assign the parameters back to their values often wouldn't fix the problem because they would now interpret them differently based on the warped geometry. This would be a major problem for generation, as it means that part generations produced after one that produced invalid geometry would appear to be the same.

A reasonable fix to this problem was to save the original part as a master design. Then on each generation, this master design would be opened again so that it wasn't operating on invalid geometry. Although some generations still produced invalid geometry, they at least wouldn't affect successive generations.

Test Milestone 2: Generating Multiple Parts

The next task was to see if this procedure could be iterated in a loop. The goal was to export each generation as its own file, and generate random parameter values within specified ranges.

Appendix C lists the code which accomplishes this task. The code exports each generation as '.FCStd' (FreeCAD document) and '.stl' files. The first is important so that the designer can modify that generation in FreeCAD and the STL file is more suitable for voxelisation and 3D printing.

The script was successfully able to generate several iterations of the part with randomised parameters. Each generation was successfully exported as a FreeCAD document and STL file.

Test Milestone 3: Analysing Parts

A script was developed that could automate the process of performing FEA analyses on each generation and output the results to a file. Appendix D lists this code.

The script takes care of producing a mesh for a part then running the CalculiX solver. As mentioned above, there was a choice between Gmsh and Netgen as the meshing tool. After experimenting with both meshing tools on many part designs beforehand, it was determined that Netgen was generally more robust and successful at producing valid meshes.

When the script was run, nearly every generation was successfully analysed. A few did not successfully pass analysis because of invalid meshing, but in most cases this was easily corrected by adjusting to finer meshing parameters. In extremely rare cases, CalculiX would catastrophically fail but this wasn't frequent enough to write off any significant number of generations.

Now that it was determined that FreeCAD could do a sufficient job of generating and analysing parts automatedly, the next step was to start working on the test milestones that didn't rely on the functionality of FreeCAD. Specifically, these test milestones focus on the Refinement and Ranking stages of the process.

Test Milestone 4: Voxelising Parts

Several voxelisation algorithms and libraries were tried and tested before one was decided on that was suitable for this tool. The one that was used in the end was a simple library written by a developer under the MIT license (24).

By default, the library takes and STL model file as an input parameter, then outputs a series of monochrome PNG images that represent the voxels at each slice in the vertical z axis. To make the library more suitable for for the automation process, it was modified to instead return a 3D array data structure of voxels.

One problem that was encountered was that the base aspect ratio would not be proportional to the actual part dimensions if the base was non-square. For example, if the base had a width and depth of 60mm by 80mm, the output voxel model would have a resolution of 100 by 100 pixels in the x and y axes. This means the width would have more resolution than the depth, so the part would appear stretched when rendered with cube voxels. This only required a small library modification, since the actual part dimensions were read in normally but instead square dimensions were enforced.

Another problem encountered was that the resolution of voxels wouldn't be proportional to the part dimensions. The base dimensions on the x and y axes would be 100 by 100 pixels. For this purpose, it is desirable for a voxel to represent a constant element size. This was easily fixed as the actual part dimensions were read in by design, so the code was modified to have the scanning resolution proportional to the dimensions.

One unsolved problem with this algorithm was that voxelised parts would exhibit some incorrect artifacts. The figures below show some examples.

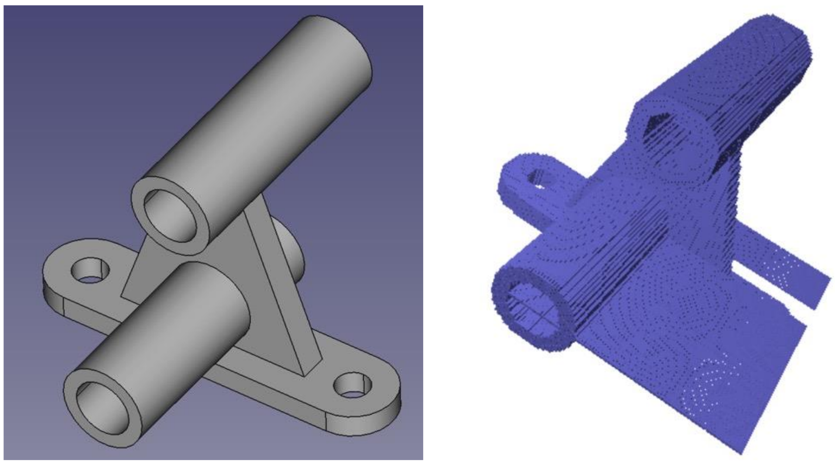


Figure 20 - Artifact planes on voxelised part

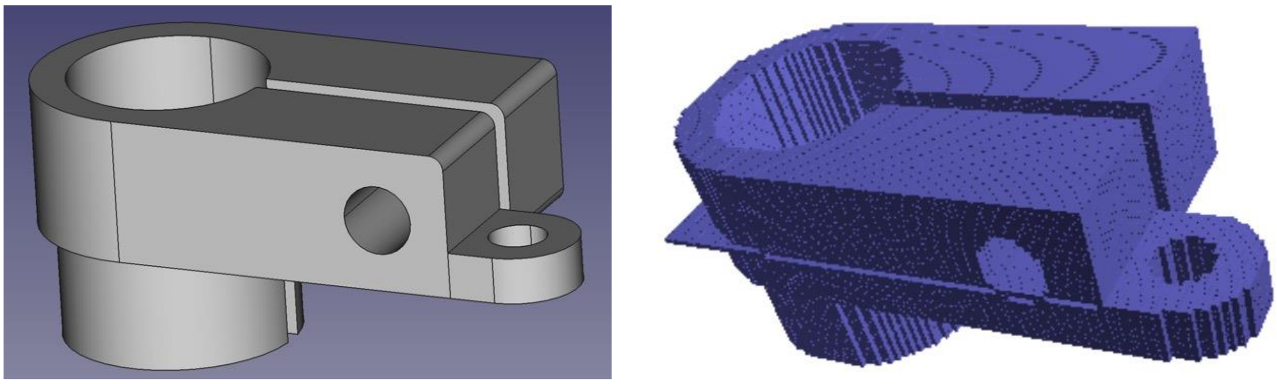


Figure 21 - Artifact planes and missing edge fillets

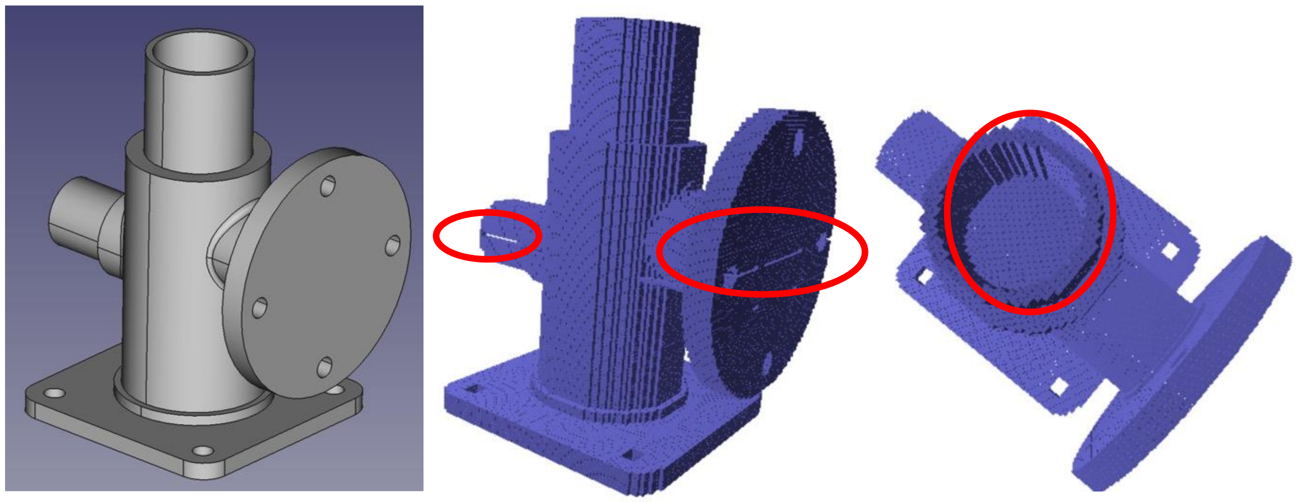


Figure 22 - Slots of empty materials and closed up features

Most of the time, this problem would disappear when the scanning resolution of the voxelisation procedure was adjusted. However, this behaviour was unpredictable as there was no obvious link between scanning resolution and the chance of artifacting.

For visualisation, the simple-3dviz library was used which is also under the MIT license (25). The library is quite flexible in the data that it can accept for visualisation. The voxels were passed in as a numpy array, and simple information like voxel colours and model dimensions were passed in. A simple function call would render the voxel array in a window that could be panned, rotated, and zoomed with the mouse.

The full code listing for this procedure can be found in Appendix E

Test Milestone 5: Part volume analysis and support structure

Once the part had been voxelised, calculating the part volumes for each generation was easy, as this simply required simply counting the number of voxels in the part, then dividing by the scanning resolution. This was simply done with the statement *numpy.sum()* which calculates the sum of elements in an array.

Generating support structures was also a simple task to carry out. Generating support structures on a voxelised model was much simpler than on a polygonal or parametric part. Regions of material are considered overhanging when they are inclined from the horizontal at an angle greater than the critical value of 45 degrees. In a voxel model, a flat face inclined at 45 degrees appears like a staircase. When a feature is at an angle lower than this, it will have some voxels immediately adjacent to it

A simple ad-hoc routine was developed which iterates through each voxel in the model and checks whether there is a voxel immediately below and adjacent. If there isn't, that voxel is marked as needing support material, and additional voxels are added underneath successively until some material is touched immediately below it. Appendix F lists this code.

Test Milestone 6: Reading results from analysis

After the FEA solver has been run on a generation, the results needed to be read and interpreted externally from FreeCAD so that results metrics can be calculated. CalculiX outputs an .frd file which contains the stress, strain, displacement etc. at each nodal point in the generated mesh. Fortunately, the data in this file was neatly formatted and human readable in an ASCII table format, which meant that parsing the data wouldn't be too difficult.

A third-party library module was found which does the job of parsing this data quite efficiently (26). The module allows for easy extraction of calculated stress, strain, or displacement in the mesh at any node or position. It worked well for reading data when the FEA analysis successfully produced results. However, when there weren't any results in the FRD, an exception would be encountered by the parser and the script would crash. An easy fix was to skip analysis for that generation and return that the metrics were invalid, which was acceptable because this problem happened with few generations.

For each generation, all of the mesh nodal values for stress and displacement are read into array data structures. This information is then used to calculate the means and maximums for both stress and displacement. Appendix G lists the code that carries out this routine.

Final implementation with GUI

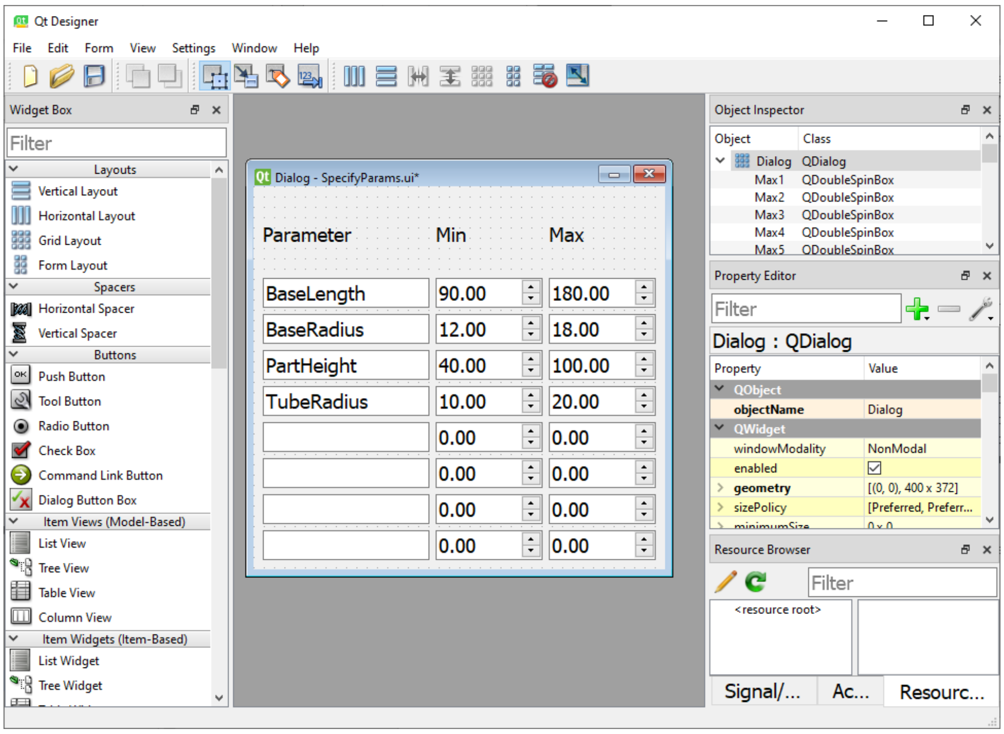
Now that all of the necessary logical components were tested and working, the last stage of development was to refine the code from these test milestones so that they would cooperate, then wrap up all of the operations into a graphical user interface (GUI) that is user-friendly.

Background

Custom workbenches and commands can easily be added to FreeCAD by writing new classes in C++ or Python. To add a workbench, the user creates a folder for the workbench in their ‘Mod’ folder then creates an ‘InitGui.py’ script. Inside the script is a class definition of the user’s workbench which extends FreeCAD’s base workbench class. The class contains definitions for the toolbar and menu commands, as well as the logic when the user activates the workbench. Appendix H lists this code.

Each toolbar item in a workbench is called a **Command**, which are defined as Python class objects and contain the logic to be executed. When the user invokes a command, FreeCAD instantiates that class object and runs through its class methods. These classes can be also defined inside the ‘InitGui.py’ file but it is easier to define each command inside its own .py script file. Also inside the class definition is the information needed to display its icon.

Workbench commands can also create their own tasks panel in the left of the FreeCAD window to allow users to control the command and view results. The tasks pane is programmed in the form of a Qt dialog box. Qt is an open-source platform for developing rich GUIs that are cross-platform. Qt dialogs can be built by instantiating the correct classes for controls like buttons, labels, text boxes etc. in code, but it is also possible to build them graphically using a program called Qt Designer.



Parameter	Min	Max
BaseLength	90.00	180.00
BaseRadius	12.00	18.00
PartHeight	40.00	100.00
TubeRadius	10.00	20.00
	0.00	0.00
	0.00	0.00
	0.00	0.00
	0.00	0.00

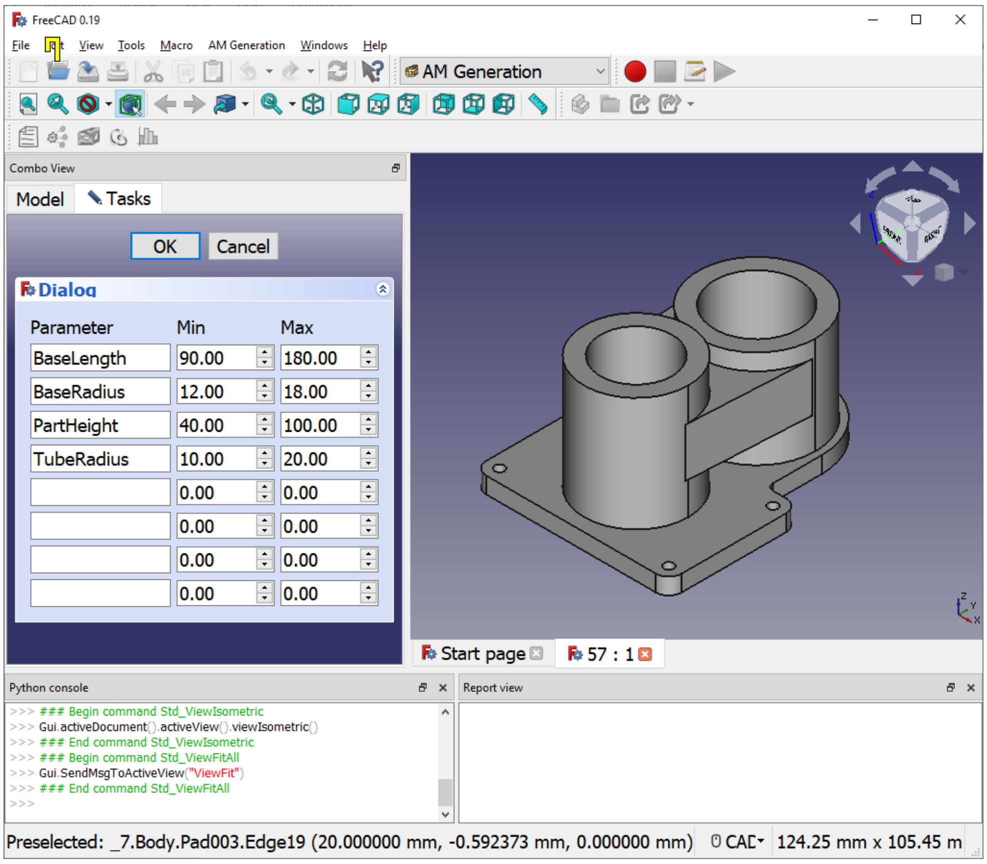
Object	Class
Dialog	QDialog
Max1	QDoubleSpinBox
Max2	QDoubleSpinBox
Max3	QDoubleSpinBox
Max4	QDoubleSpinBox
Max5	QDoubleSpinBox

Property	Value
QObject	
objectName	Dialog
QWidget	
windowModality	NonModal
enabled	☑
geometry	[(0, 0), 400 x 372]
sizePolicy	[Preferred, Preferr...
minimumSize	0 x 0

Figure 23 - Qt Designer program with Qt dialog loaded

Dialog GUIs can be easily constructed by dragging and dropping controls from the widget box on the left. Then the dialog is saved as a ‘.ui’ file which contains plaintext XML code that defines the layout of the controls. The Python code that is responsible for instantiating the GUI panel in FreeCAD can then simply load the ‘.ui’ file

Qt Designer greatly helped to accelerate development of the GUI for several reasons. Firstly, it was easy to rapidly prototype different ideas for each tool as controls could be configured much more quickly and easily than in code. Secondly, having the GUI displayed as it was being worked on helped to stimulate creativity.



Parameter	Min	Max
BaseLength	90.00	180.00
BaseRadius	12.00	18.00
PartHeight	40.00	100.00
TubeRadius	10.00	20.00
	0.00	0.00
	0.00	0.00
	0.00	0.00
	0.00	0.00

```
>>> ### Begin command Std_ViewIsometric
>>> Gui.activeDocument().activeView().viewIsometric()
>>> ### End command Std_ViewIsometric
>>> ### Begin command Std_ViewFitAll
>>> Gui.SendMsgToActiveView("ViewFit")
>>> ### End command Std_ViewFitAll
>>>
```

```
Preselected: _7.Body.Pad003.Edge19 (20.000000 mm, -0.592373 mm, 0.000000 mm)   CAD   124.25 mm x 105.45 m
```

Figure 24 - Qt Designer dialog being prototyped in FreeCAD

The process of designing the GUIs began with sketching some mock-ups (Appendix U). One of the goals that was kept in mind when designing these interfaces was to keep them as simple as possible, and avoid exposing unnecessary details to the user where possible.

In the final plugin, a command icon is dedicated to each stage of the process.

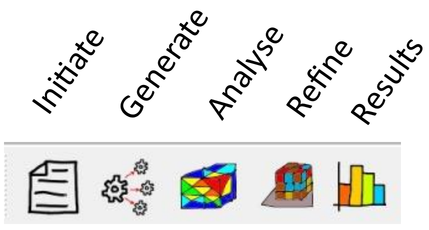
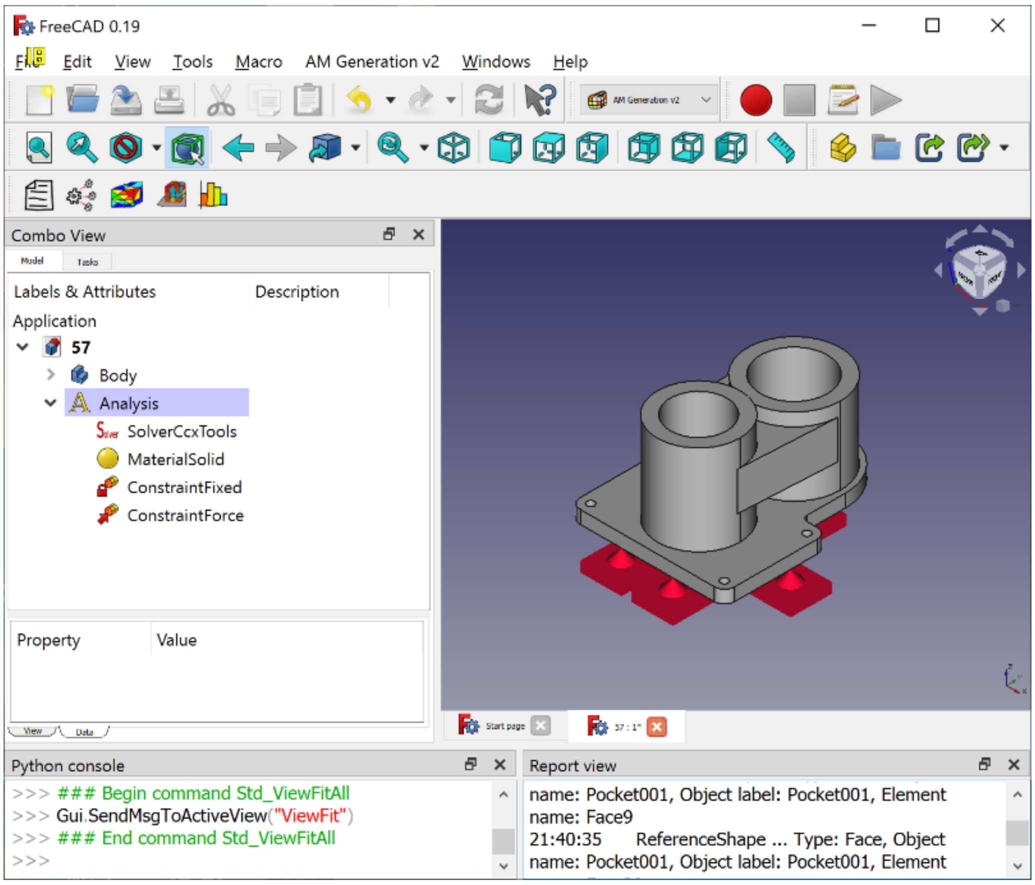


Figure 25 - Commands implemented in the plugin workbench

Appendix I lists all of the supporting code used in this workbench.

Initiation

Before the process can begin, the user must have assigned labels to all the constraints that will be parameterised. Additionally, if the user decides to perform FEA analysis for this procedure run, then an Analysis container needs to be made and the part material, constraints and loads need to be defined before initiation can commence. Figure 26 below shows an example of this.

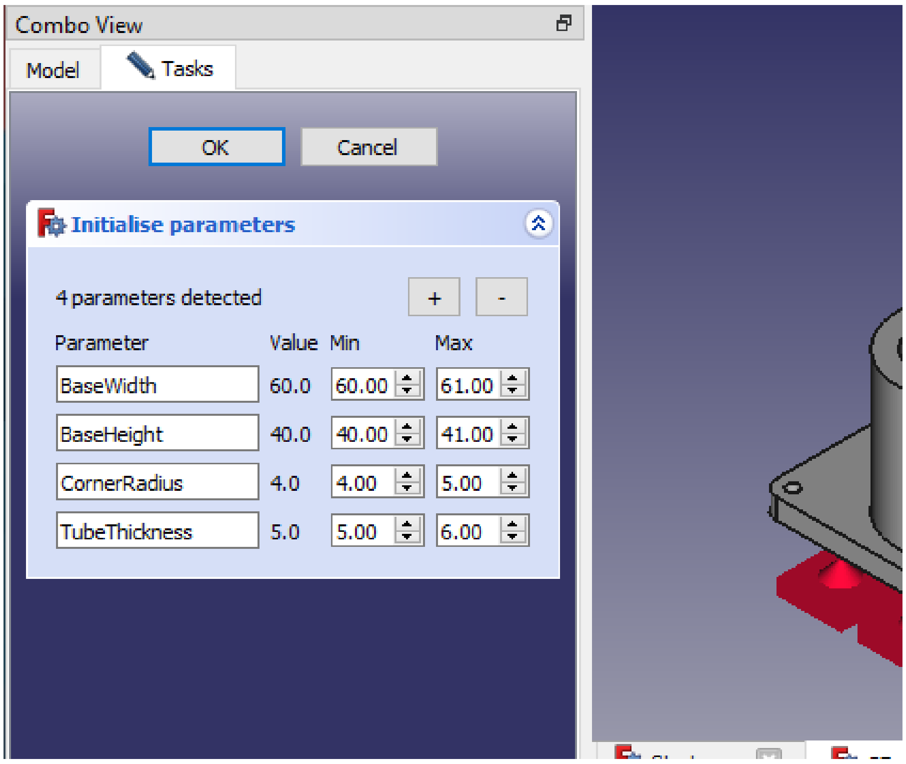


```
>>> ### Begin command Std_ViewFitAll
>>> Gui.SendMsgToActiveView("ViewFit")
>>> ### End command Std_ViewFitAll
>>>
```

```
name: Pocket001, Object label: Pocket001, Element
name: Face9
21:40:35      ReferenceShape ... Type: Face, Object
name: Pocket001, Object label: Pocket001, Element
```

Figure 26 - Prerequisite example environment for starting the ***Initiate*** *procedure*

The user then begins initiation by clicking on the **Initiate** command which brings up the following interface.



4 parameters detected

Parameter	Value	Min	Max
BaseWidth	60.0	60.00	61.00
BaseHeight	40.0	40.00	41.00
CornerRadius	4.0	4.00	5.00
TubeThickness	5.0	5.00	6.00

*Figure 27 - **Initiate** command GUI*

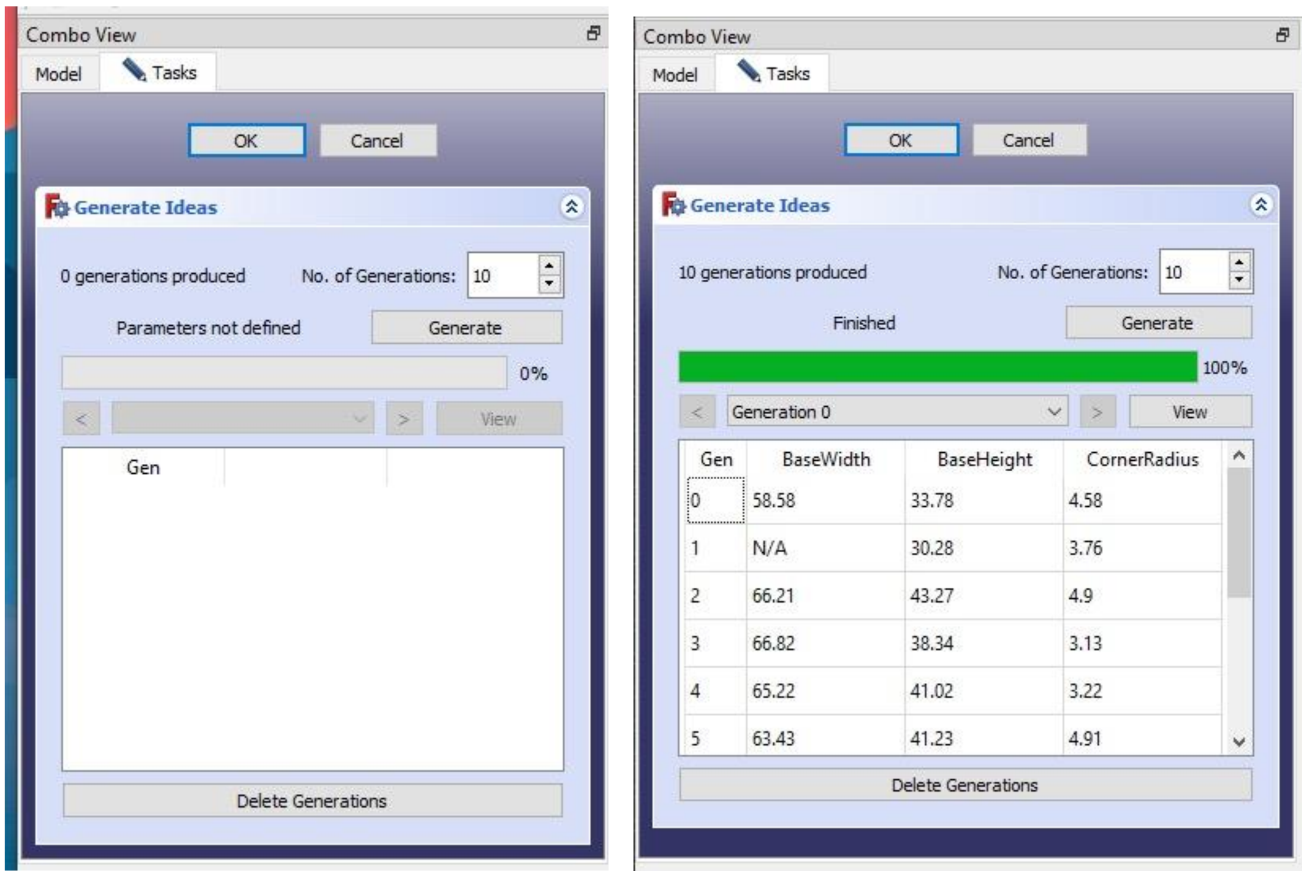
When the command is called, it automatically scans through all of the objects in the part for named parameters, then adds them to the list. The user can remove parameter names that they wouldn't like to be regenerated, and they can add additional parameters that may not have been detected or that might be added later.

Then, the value ranges for parameter randomisation are given by the user into the **min** and **max** input boxes. Once this is done, the user clicks **OK**, which saves the parameter ranges to a file called "Parameters.txt" to be used by the next stage in the process.

Appendix J lists the relevant code for the **Initialise** command.

Generation

Clicking on the **Generate** command brings up the interface shown in the left panel of Figure 28.



0 generations produced

No. of Generations: 10

Parameters not defined

Generate

0%

Gen		

10 generations produced

No. of Generations: 10

Finished

Generate

100%

Generation 0

Gen	BaseWidth	BaseHeight	CornerRadius
0	58.58	33.78	4.58
1	N/A	30.28	3.76
2	66.21	43.27	4.9
3	66.82	38.34	3.13
4	65.22	41.02	3.22
5	63.43	41.23	4.91

*Figure 28 - **Generate** command GUI, before and after the generation procedure*

In the background, the command has just read the parameter ranges from the "Parameters.txt" file and has located the constraints in the part that they correspond with.

To begin generation, the user simply enters the number of generations to produce and clicks on **Generate**. The tool then carries out the process of producing new part generations with adjusted parameters using the same procedure described in Test Milestone 2.

A progress bar gives the user feedback on how quickly the procedure is taking. Once generation has completed, the interface looks the right panel of Figure 28. The table lists the parameter values that were assigned for each generation. Individual generations can be viewed in FreeCAD by selecting from the drop-down box and clicking **View**.

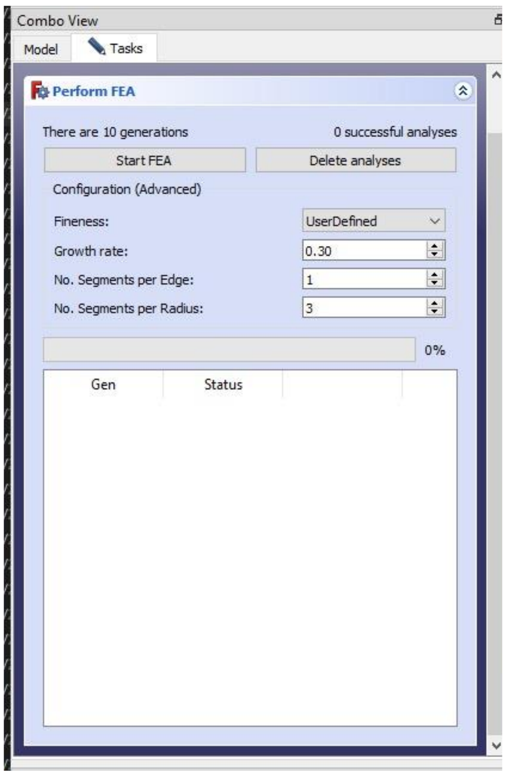
Appendix K lists the relevant code for the **Generate** command.

Analysis

Clicking on the **Analyse** command will bring up the analysis interface shown in the left panel of Figure 29. To start performing analysis on the generations that were produced, the user simply has to click on **Start FEA**. This follows the procedure that was outlined in Test Milestone 3.

There are also some advanced controls for configuring the meshing tool. In most cases it is not necessary at all to use these controls. They have been added because it was found that many parts that were tested on wouldn't be meshed properly under the default parameters.

Once all generations have been analysed, the interface will look something like the right panel of Figure 29. The table shows whether each generation was analysed successfully or there was a problem. In this case, most generations seemed to have been analysed successfully but a few failed analysis, or were not analysed due to an issue wish meshing.



There are 10 generations 0 successful analyses

Configuration (Advanced)

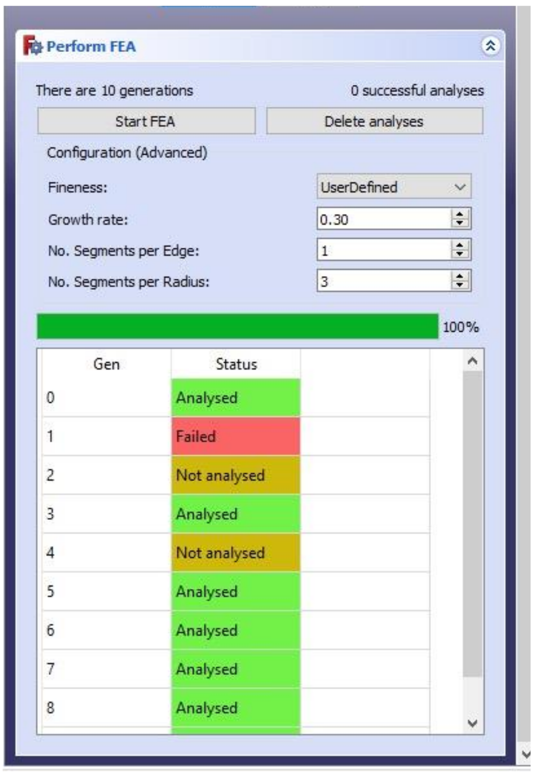
Fineness: UserDefined

Growth rate: 0.30

No. Segments per Edge: 1

No. Segments per Radius: 3

0%



There are 10 generations 0 successful analyses

Configuration (Advanced)

Fineness: UserDefined

Growth rate: 0.30

No. Segments per Edge: 1

No. Segments per Radius: 3

100%

Gen	Status
0	Analysed
1	Failed
2	Not analysed
3	Analysed
4	Not analysed
5	Analysed
6	Analysed
7	Analysed
8	Analysed

*Figure 29 - **Analysis** command GUI, before and after analysis has occured*

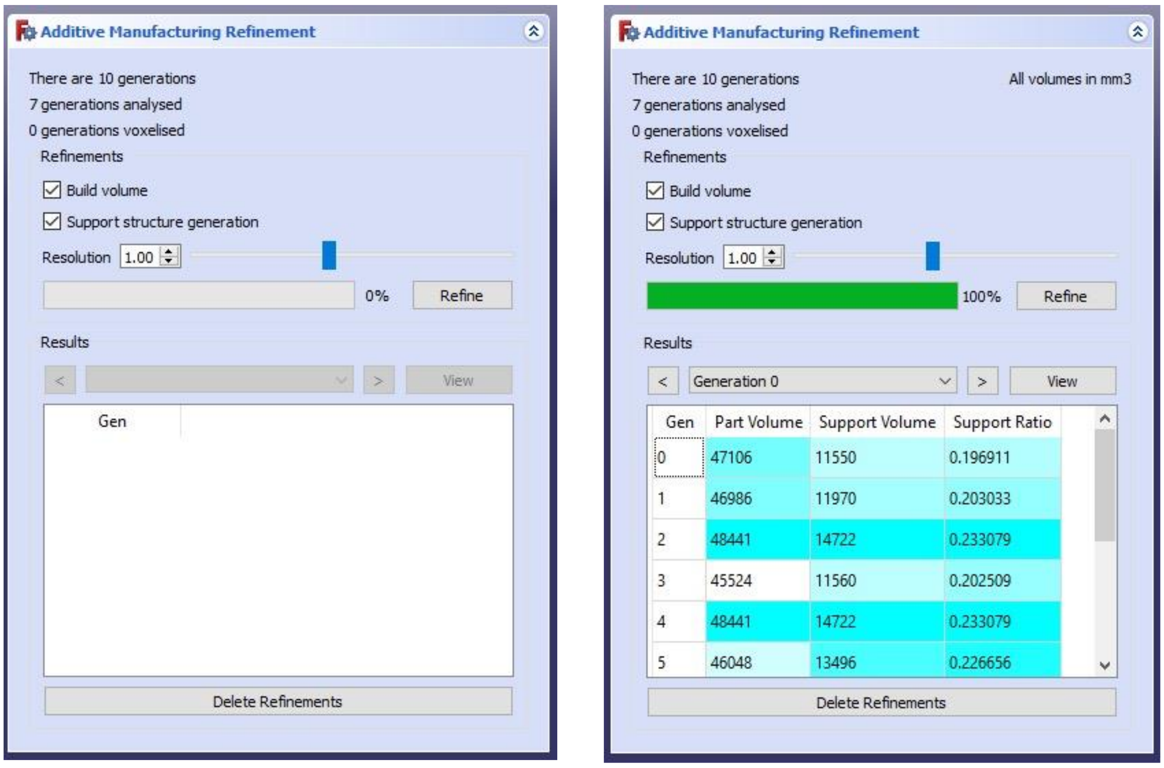
Appendix L lists the relevant code for the **Analyse** command.

Refinement

Activating the **Refine** command brings up the refinement interface shown in the left panel of Figure 30. This stage is optional, as the user might decide that they do not want to optimise part volume or support structures. The user is given a choice between optimising only build volume, only optimising support structures, or both.

The only configuration detail exposed to the user was the voxelisation resolution, which can be varied between 0.25 – 2 voxels/mm. This was necessary, because increasing the voxel resolution might be desired for parts that have lots of small and fine detail that needs to be represented accurately. However, increasing voxel resolution rapidly makes this task more computationally expensive, so finding a trade-off that suits the optimisation case is important.

The refinement process is started by clicking the **Refine** button. The procedure followed is derived from the logic built in Test Milestones 5 and 6. Once the process has completed, the results table is filled with the part volume, support structure volume, and support volume ratio for each generation. Each column is also applied with colour scaling for easy visual comparison of metrics between generations.


Additive Manufacturing Refinement

There are 10 generations
7 generations analysed
0 generations voxelised

Refinements

☑ Build volume
☑ Support structure generation

Resolution 1.00

0% Refine

Results

< > View

Gen

Delete Refinements

Additive Manufacturing Refinement

There are 10 generations
7 generations analysed
0 generations voxelised

All volumes in mm3

Refinements

☑ Build volume
☑ Support structure generation

Resolution 1.00

100% Refine

Results

< Generation 0 > View

Gen	Part Volume	Support Volume	Support Ratio
0	47106	11550	0.196911
1	46986	11970	0.203033
2	48441	14722	0.233079
3	45524	11560	0.202509
4	48441	14722	0.233079
5	46048	13496	0.226656

Delete Refinements

*Figure 30 - **Refinement** command GUI, before and after the refinement procedure*

The user can view what each voxelised model looks like by selecting it from the drop-down box above the results table and clicking **View**. This opens up another window which shows the voxelised part in orange voxels, with support structure voxels in red.

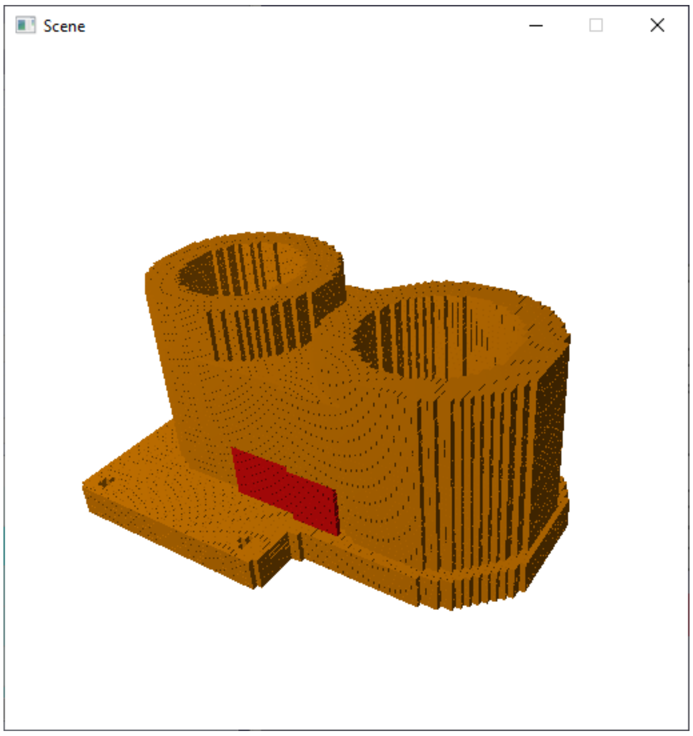
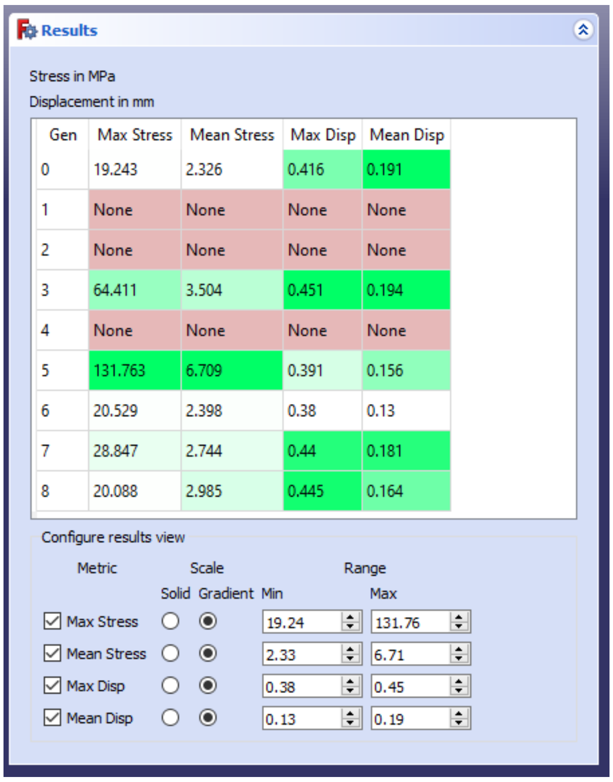


Figure 31 - Rendered view of voxelised generation

Appendix M lists the relevant code for the **Refine** command.

Ranking

The last stage involves calculating the overall mechanical part metrics for each generation, and displaying them to the user in a fashion that makes them easy to compare. When the user first invokes this process by clicking on the **Results** command, the program pauses for a minute to calculate all of the necessary metrics as outlined in Test Milestone 7. Once this process is complete, the following interface is produced.



Stress in MPa
Displacement in mm

Gen	Max Stress	Mean Stress	Max Disp	Mean Disp
0	19.243	2.326	0.416	0.191
1	None	None	None	None
2	None	None	None	None
3	64.411	3.504	0.451	0.194
4	None	None	None	None
5	131.763	6.709	0.391	0.156
6	20.529	2.398	0.38	0.13
7	28.847	2.744	0.44	0.181
8	20.088	2.985	0.445	0.164

Configure results view

Metric	Scale		Range	
	Solid	Gradient	Min	Max
✓ Max Stress	○	●	19.24	131.76
✓ Mean Stress	○	●	2.33	6.71
✓ Max Disp	○	●	0.38	0.45
✓ Mean Disp	○	●	0.13	0.19

*Figure 32 - **Results** command GUI*

The maximum and mean for both stress and displacement are displayed for each generation, and each column is colour scaled for easy comparison in a similar fashion to the refinement table. Additionally, the user can configure how the table is coloured. The upper and lower limit of the colour gradient can be adjusted to span a narrower range to highlight more relevant results.

Using this information alongside the refinement results and the generation parameters, the user can then make an informed decision on which generation was the most optimal and proceed with the design stage using that generation if it is satisfactory.

Generations that failed in the analysis stage do not have calculated metrics, and thus are highlighted red in the results table.

Appendix N lists the relevant code for the **Results** command.

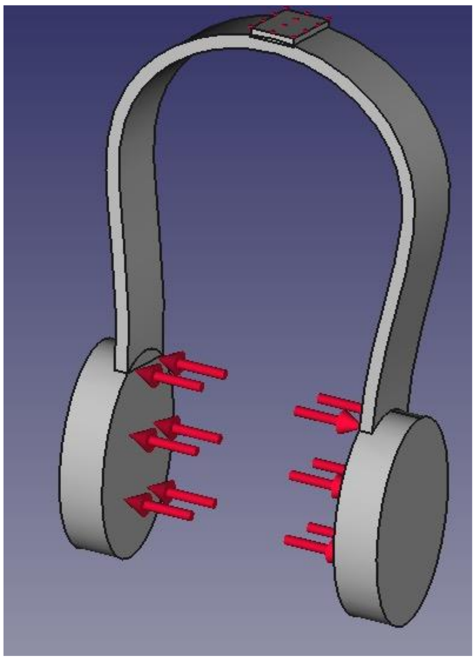
Experimentation and Results

To explore and test the usefulness of the tool, a number of design case studies were devised for some parts that could be additively manufactured but are mostly made by traditional processes.

These designs were chosen such that they were highly parameterizable and could be optimised in terms of mechanical performance or printability, but preferably both.

Case Study 1 – Headphones Frame

The arched frame that join the two sides of a pair of headphones are a crucial component. They need to be strong enough to withstand being dropped or mishandled, but flexible enough to fit typical user's head. This is a part that is advantageous to 3D print because this makes it efficient to produce bespoke to an individual. Additionally, manufacturing it as a single-part compliant mechanism is more efficient and less prone to error.



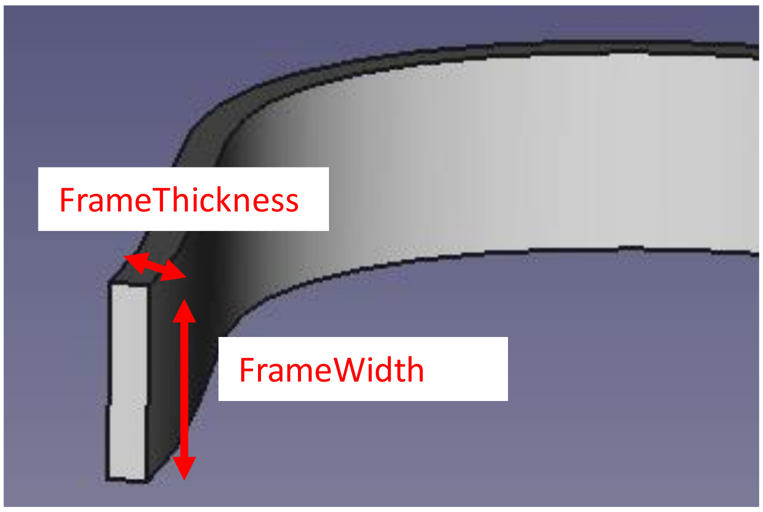


Figure 33 - Depiction of load case, and parameters to be randomised for Case Study 1

When not in use, the headphones will be closed together. It is expected that the user will exert a reasonable amount of force to open the headphones to the width of their head. In this case, a typical person's head is 20cm (27) so they must be able to stretch to this width comfortably. In their resting position, the 'muffs' of the headphones are 15cm apart so the headphones must stretch by approximately 40-60 mm when pryed open.

A comfortable prying force is 20N or the equivalent of lifting approximately 2kg. Thus, equal and opposite forces of 10N each will be applied to the end where the ear muffs are.

Finally, the part must bend without exceeding the tensile strength in any region and thus breaking. The part will be printed using ABS. Assuming that the material has a tensile strength of 40MPa, the upper limit of stress allowed in the part is 20MPa with a safety factor of 2(28).

The parameters to be varied are the width and thickness of the frame. Some experimentation revealed that the stress and displacement varied widely with particular ranges for **FrameThickness** and **FrameWidth**. These value ranges were used in the generation stage.

Table 2 - Parameter randomisation ranges for Case Study 1

Parameter	Min	Max

FrameThickness	4 mm	6 mm
FrameWidth	15 mm	40 mm

Support structure refinement was not performed on the generations as they could all be printed on one side without needing any support. The procedure was run for 50 generations with analysis. The full list of results for each generation can be found in Appendix O.

The generation that produced the best results was generation 45 with the following attributes:

Table 3 - Part attributes for generation 45 of Case Study 1

Attribute	Value
FrameThickness	4.37 mm
FrameWidth	27.62 mm
Peak stress	20.2 MPa
Deflection	43 mm
Part volume	46320 mm^3

The generation produced had sensible parameters for its results, which were verified by viewing the generation in FreeCAD's 'FEA' workbench. Thus, this case study has demonstrated that the tool is suitable and convenient for optimising designs that are easily parameterisable, and where the criteria is simple to calculate.

Case Study 2 – Bicycle Seat

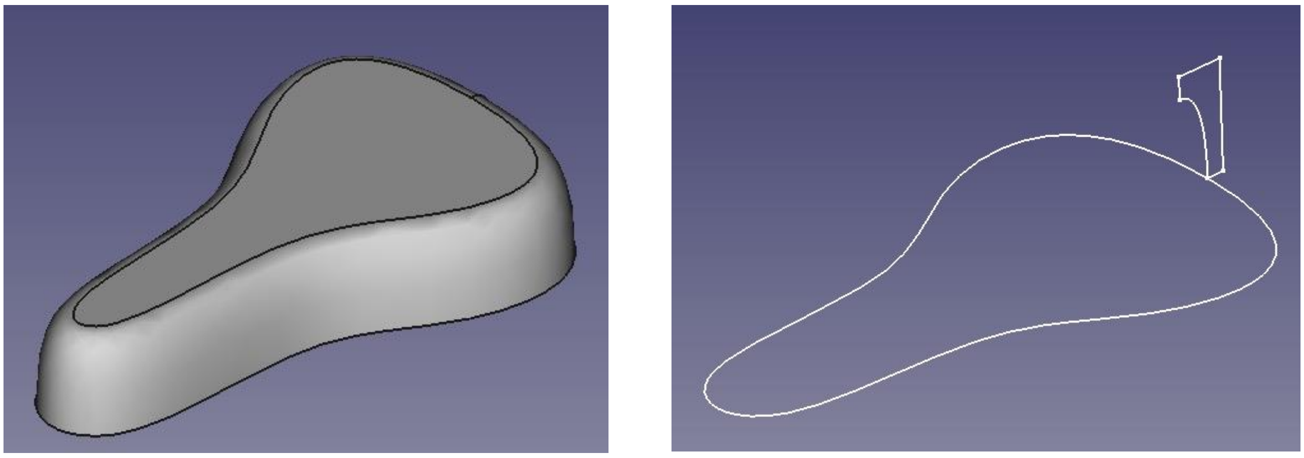


Figure 34 - Shell of bicycle seat, and extrustion and sweep sketches which produced the shell

A bicycle seat shell was designed using a set of B-splines. The ratios and distances between the spline points are parameterised so that the generative tool can easily produce different design variations.

The goal of this case study is to visually explore the design space in terms of shape and aesthetics, and to keep part volume at a minimum where possible. Thus, there is no definitively optimal generation that will come out of this procedure. The objective is to produce a wide variety of designs that the user can pick which suits their own preferences.

After some experimentation, the following variables were parameterised in the given ranges.

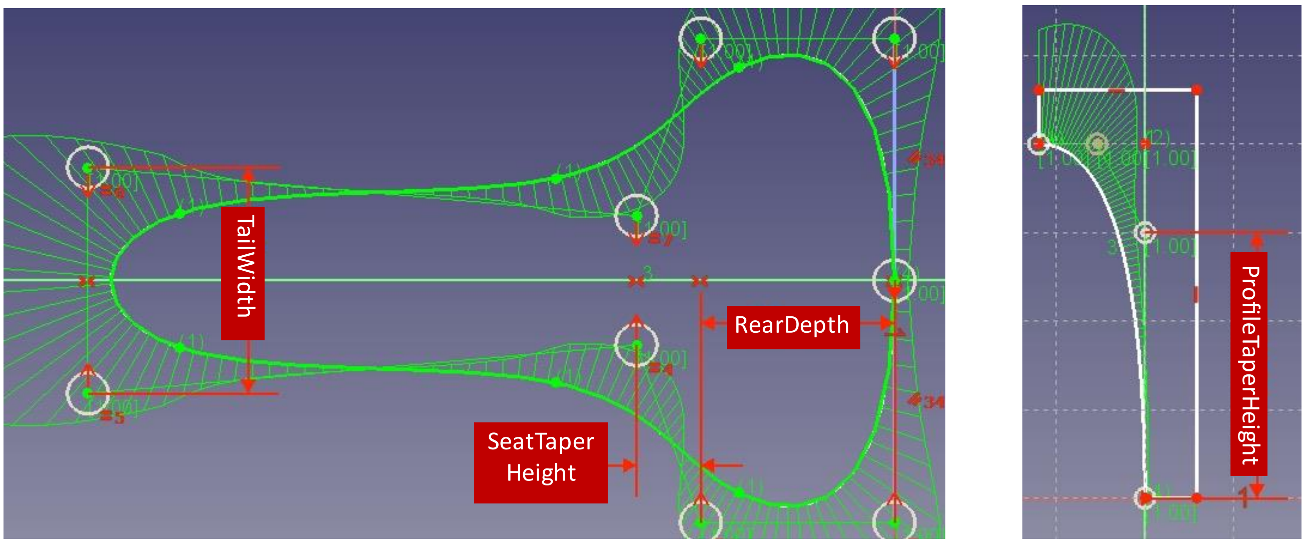


Figure 35 - Parameters to be randomised for Case Study 2

The outline of the seat is made from nine points compromising a B-spline shown in the figure above. This is extruded as a solid block. Then, the curved profile of the seat is produced using the ‘Groove’ operation which cuts out the profile drawn in the figure above. Finally, the part was shelled using the ‘Thickness’ operation.

After some experimentation, a good variety of design situations were obtained by varying four parameters with the following ranges:

Table 4 - Parameter randomisation ranges for Case Study 2

Parameter	Min	Max
TailWidth	60 mm	90 mm
RearDepth	60 mm	100 mm
SeatTaperHeight	0 mm	50 mm
ProfileTaperWidth	12 mm	20 mm
ProfileTaperHeight	20 mm	40 mm

Because of the relatively large range of values to explore, the procedure was run for 100 generations. This was decided as a compromise between maximizing the potential to explore the whole range space, and keeping the number limited so that the user would not be overwhelmed by the number of choices.

After this, the generations were run through the refinement stage to calculate their volume. Support structure generation was disabled, as every variation uses almost no support material when they are printed upside down.

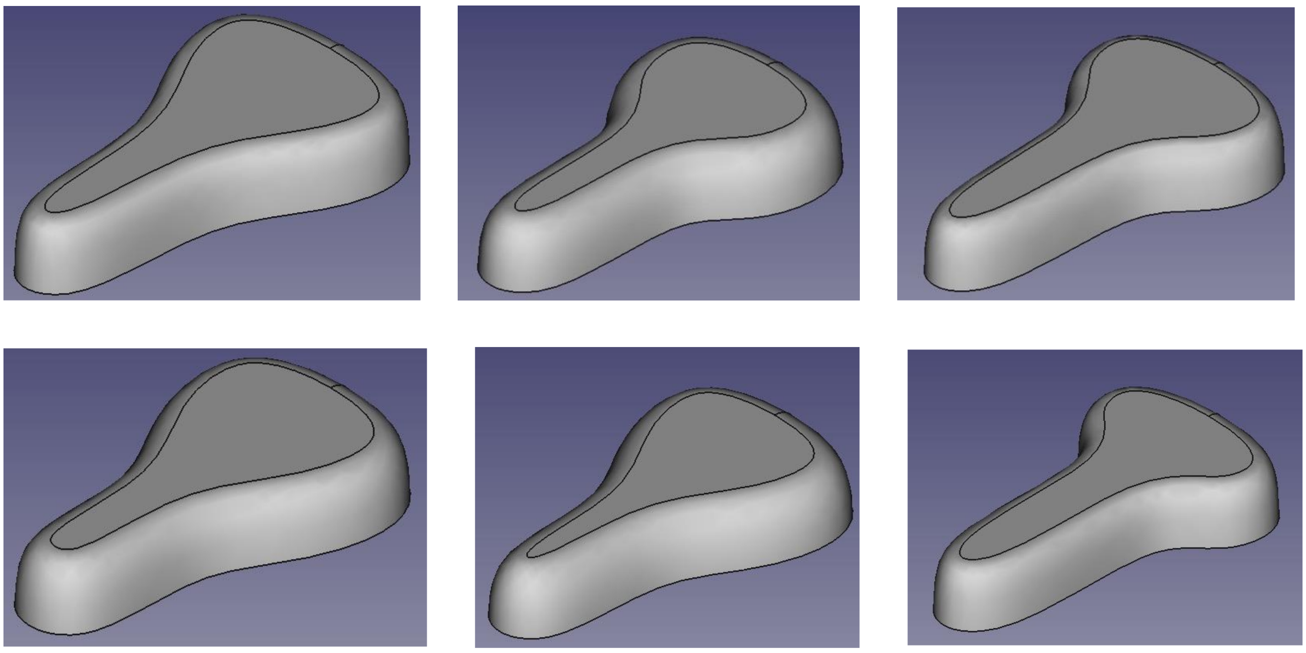


Figure 36 - Sample of generations for Case Study 2

The procedure produced a reasonable amount of variety in the generated designs. Shown above are some of the most different design variations produced. The range in part volume across all generations was also quite modest, ranging between 119 – 129 cm3 or 7.7% of the maximum volume. The original part volume was 123.3 cm3, so this procedure was relatively ineffective at part volume reduction. However, it can be clearly seen that the procedure successfully produced a good variety of designs which was the main goal of this case study.

Case Study 3 – Bicycle Pedal

This study involves designing a bicycle pedal using printed PLA that is more material efficient. The frame has a skeletal topology as it is made from thin struts.

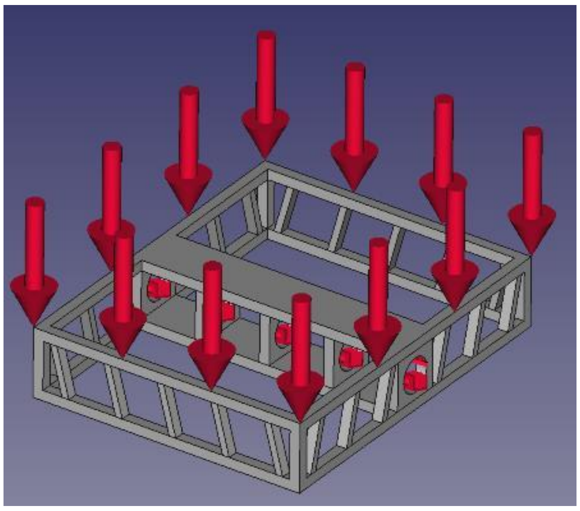


Figure 37 - Bicycle pedal loading conditions for Case Study 3

In this model, a fixed support is attached on the circular faces in the centre as this is where the pedal shaft will be. A force of 981N is applied to the top face which is equivalent to the weight of 100kg, as this must be able to support the weight of the vast majority of people.

The objective of this case is to obtain a design variation where maximum stress doesn't exceed the tensile strength of PLA which is 37MPa (28). Additionally, we want to obtain a design that minimises material volume where possible. In particular, a lot of material could be wasted on support structures if designed badly, as there are a lot of regions with empty space underneath the overhanging top struts.

The frame and struts contribute the most towards the material usage and mechanical performance, so these features were parameterised. The exact parameters that were varied were **StrutAngle**, **StrutThickness** and **FrameThickness**.

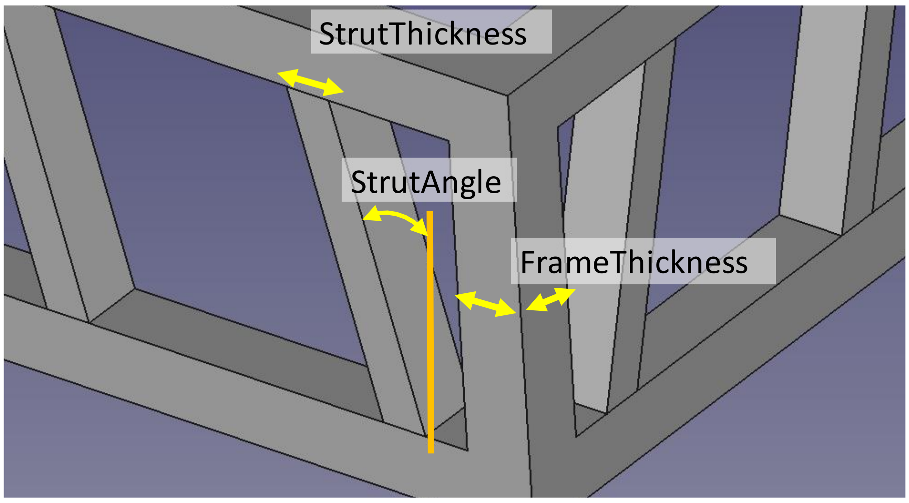


Figure 38 - Parameters to be randomised for Case Study 3

Table 5 - Parameter randomisation ranges for Case Study 3, Process Iteration 1

Parameter	Min	Max
StrutThickness	2 mm	4 mm
StrutAngle	5 deg	25 deg
FrameThicknes	2 mm	4 mm

These parameters were varied with the following ranges. 30 generations were produced. The generation, analysis, and refinement stages were all completed quite quickly because of how simple the geometry was.

Part volume ranged between approximately 16000 - 24000 mm3 and support volume ranged between 17300 – 21300 mm3. Interestingly, there was a strong negative correlation between the two variables – generations with more support volume tended to have less part volume and vice versa. Further investigation revealed a strong positive correlation between **FrameThickness** and part volume. **StrutAngle** and **StrutThickness** comparatively didn't affect part volume as much, but did affect the support volume.

Maximum stress also varied widely between generations with a range of 25 – 137MPa. Out of the 30 generations, only 7 had a stress below the threshold of 37MPa, and just 2 were below the 1.5x safety factor threshold at 25MPa. On closer inspection, there was generally a positive correlation between *lower stress* and higher **FrameThickness**. **StrutThickness** and **StrutAngle** also showed some correlation but affected the stress distribution much less.

The full table of metrics for these 30 generations can be found in Appendix P.

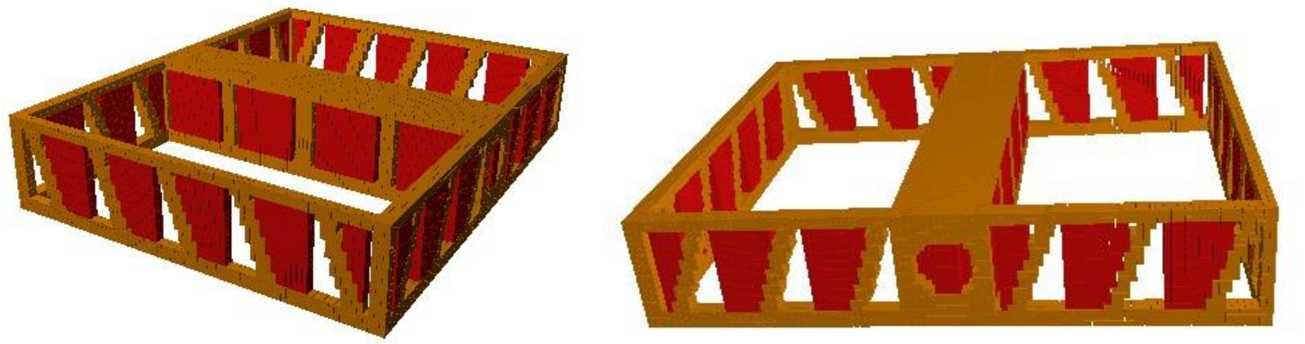


Figure 39 - Voxel models of generation sample for Case Study 3, Process Iteration 1

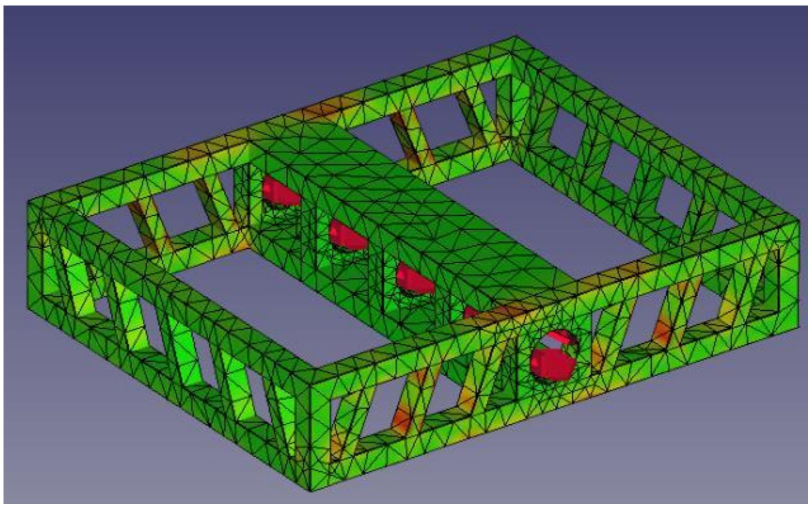


Figure 40 - Stress colourmap for generated part of Case Study 3, Process Iteration 1

On closer inspection of the voxel and analysis models, it is apparent that support structures are being used disproportionately in areas that do not carry as much of the mechanical load. Particularly, the solid face above the shaft opening has little stress exerted yet has a large volume of red support voxels underneath. Additionally, the top frame is mostly green yet there is a lot of support material underneath it.

Using this information, another design iteration was devised to minimise these overhangs. Diagonal struts surrounding the frame were alternated in both directions, and the upper shaft face was converted into a frame.

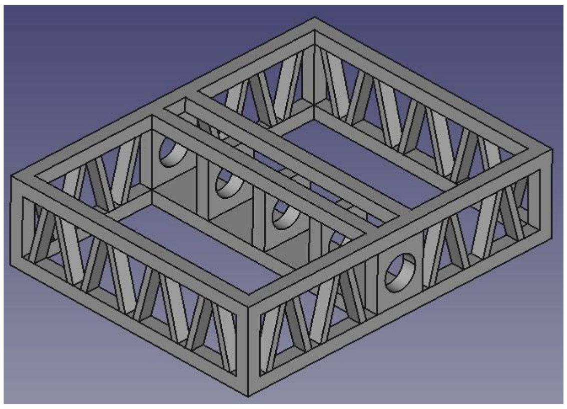
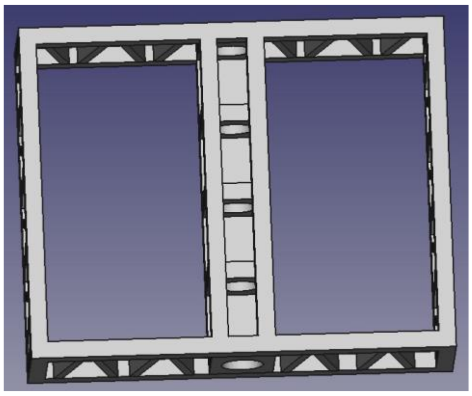



Figure 41 - Revised design for bicycle pedal in Case Study 3

In addition, the parameter ranges were revised. **FrameThickness** range was increased to reduce the peak stresses. **StrutAngle** limit was increased up to 45 degrees to cover a larger width of the side frames, so that support material could be minimised.

Table 6 - Parameter randomisation ranges for Case Study 3, Process Iteration 2

Parameter	Min	Max
StrutThickness	2 mm	4 mm
StrutAngle	17.2 deg	45 deg
FrameThickness	4.5 mm	7.5 mm

This second process iteration was run for just 10 generations, but this was enough as the ranges for peak stress and part volume were much smaller indicating better convergence. Of the 8 generations that were successfully analysed, 6 were well converged to around 20-21MPa for peak stress. Part volume was a bit wider in the range of 45000-58000 mm3 indicating that there was a bit more room for convergence. The full results for all generations as well as their models can be found in Appendix Q.

The process was run for a third and last iteration with a more shifted range of parameters.

Table 7 - Parameter randomisation ranges for Case Study 3, Process Iteration 3

Parameter	Min	Max
StrutThickness	3 mm	4.5 mm
StrutAngle	30 deg	45 deg
FrameThickness	5 mm	6 mm

50 generations were produced. This time, good convergence was achieved in both stress and part volume. All generations were in the range of 44000-49000 mm3 for part volume. However, many generations had produced parts that were difficult to mesh, resulting in many generations that failed analysis or gave anomalously incorrect stresses.

The most optimal generation chosen was Generation 24, with peak stress of 24.4MPa and the lowest part volume of 44128 mm3. The full list of results for all generations can be found in Appendix R.

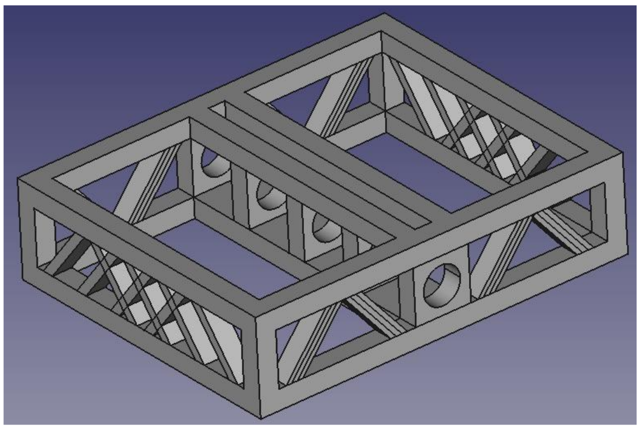
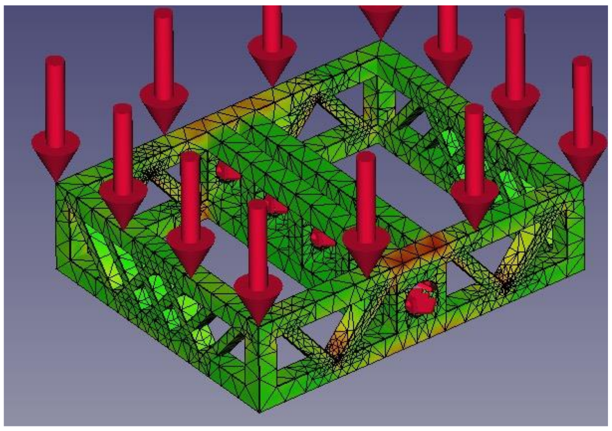
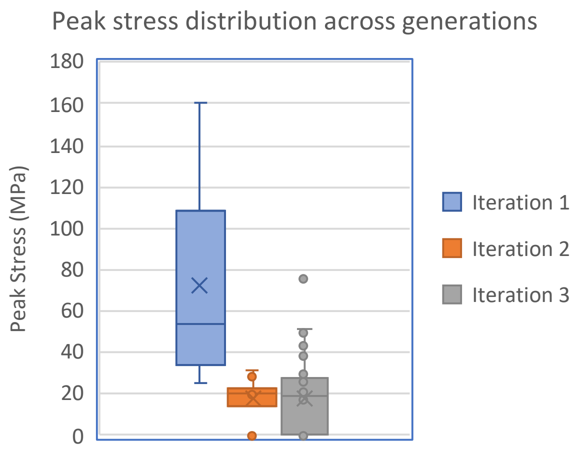



Figure 42 - CAD model and stress colourmap of most optimal generation

The box plots of peak stresses and part volumes across the three process iterations in figure 43 help to illustrate the convergence of this case study. In the first iteration, there is a wide range in peak stress across all the generations. The part volume is initially quite low.

In the second iteration, the peak stress range has narrowed and decreased significantly towards the more desired region as a result of the redesign, despite the only modest change in parameter ranges. This indicates that the redesign was successful in guiding the process towards more optimal performance. The part volume range also increased substantially in both range and absolute height, which is expected due to the added struts and the higher range for **FrameThickness**.

The third process iteration widens the peak stress range, due to the shift in parameter ranges that cover the tight region where small variations result in large variations for mechanical performance. Conversely, the range in part volume has decreased in range and absolute height, indicating that this iteration was in the most optimal parameter range where small changes in volume effect large changes in mechanical performance.



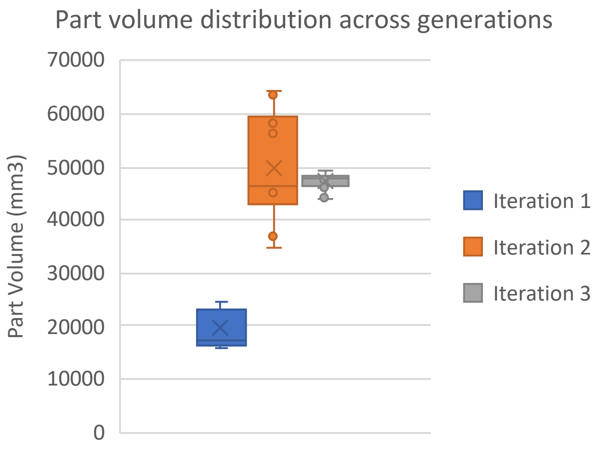


Figure 43 - Box plots of peak stress and part volume across generations for all three process iterations

The total part volume for the bike pedal if there were no struts would be 67800 mm3, so the part produced from the optimal generation represents a substantial 34% decrease in material. Thus, this case study shows that this tool can be used to stimulate the designer into improving a part design towards their desired goals.

Case Study 4 – Connecting rod with lattice structure

Although this tool doesn't support the automatic generation of optimal lattices in solid volumes, it can still be used to optimise parts that already have lattice structures constructed inside them if they are sufficiently parameterised. Below is a simple rectangular connecting rod that is used to constrain two shafts.

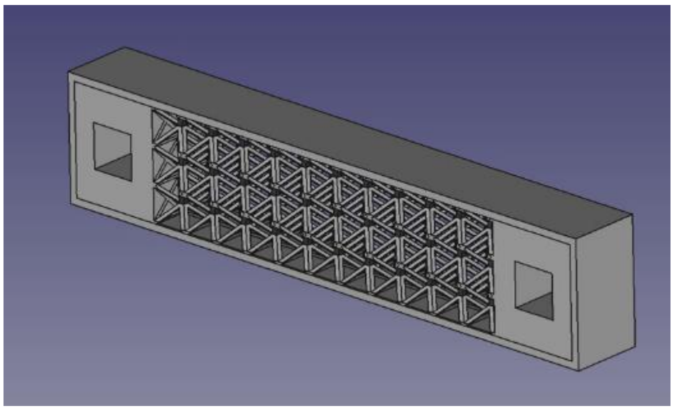


Figure 44 - Specimen part for Case Study 4

The lattice structure was constructed in FreeCAD using a pattern of subtractive features that cut material in all three axes to leave behind struts.

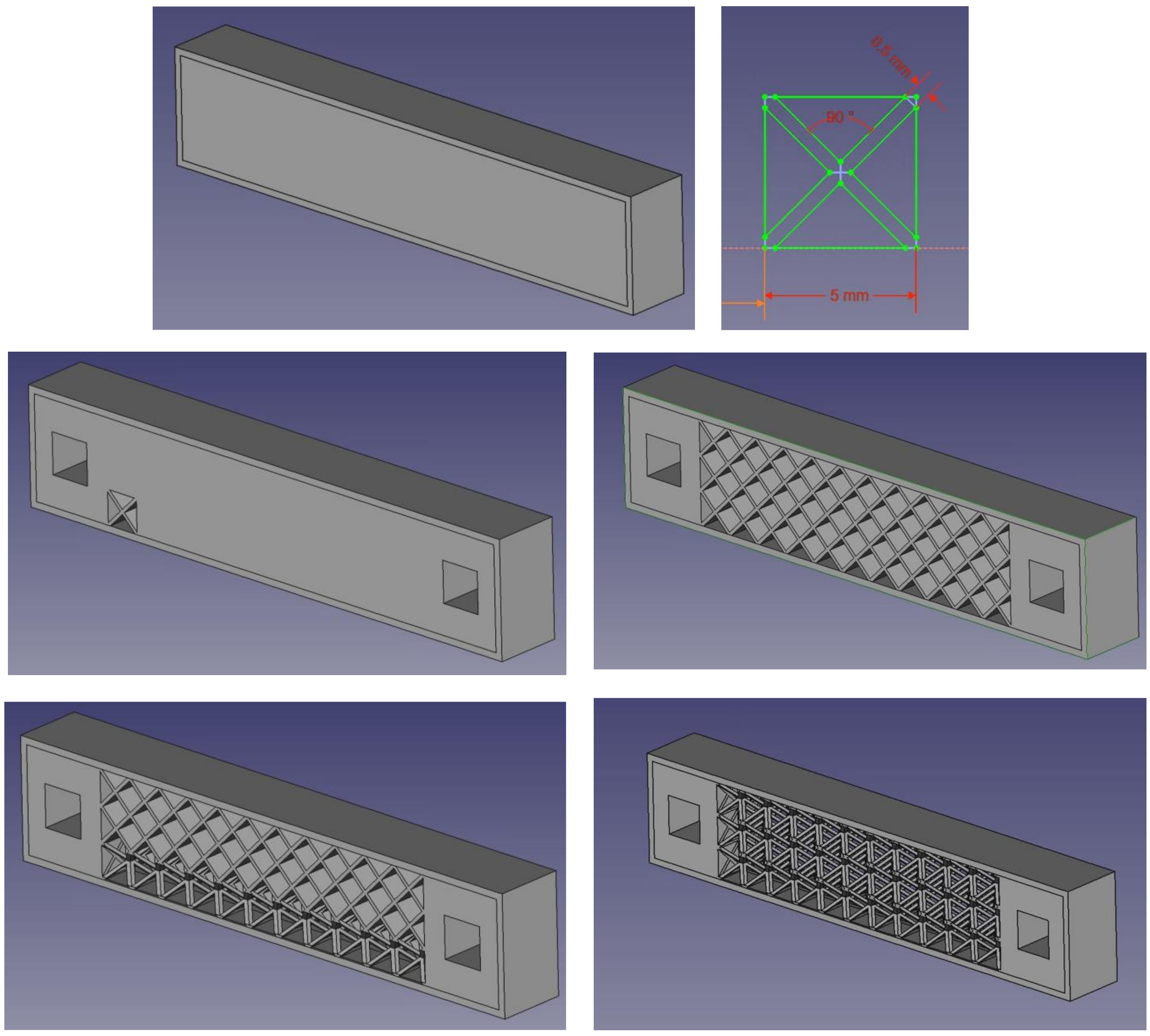


Figure 45 - Timeline of operations illustrating how a lattice volume of vertical struts can be produced by a three-dimensional pattern of Pocket features

The frame thickness and strut thickness were each parameterised as **FrameThickness** and **StrutThickness**, respectively. In this particular case, a fixed constraint was applied to the left shaft hole, and a pair of 100N and 70N forces applied to the right shaft.

It is envisioned that the connecting rod will be used as a disposable temporary support in an assembly for flatpack furniture. For this reason, it will be printed using inexpensive PLA with a tensile strength of 27MPa, so the maximum allowable stress is 18MPa with a safety factor of 1.5. The purpose of the part is to constrain two assembly shafts in place as rigidly as possible.

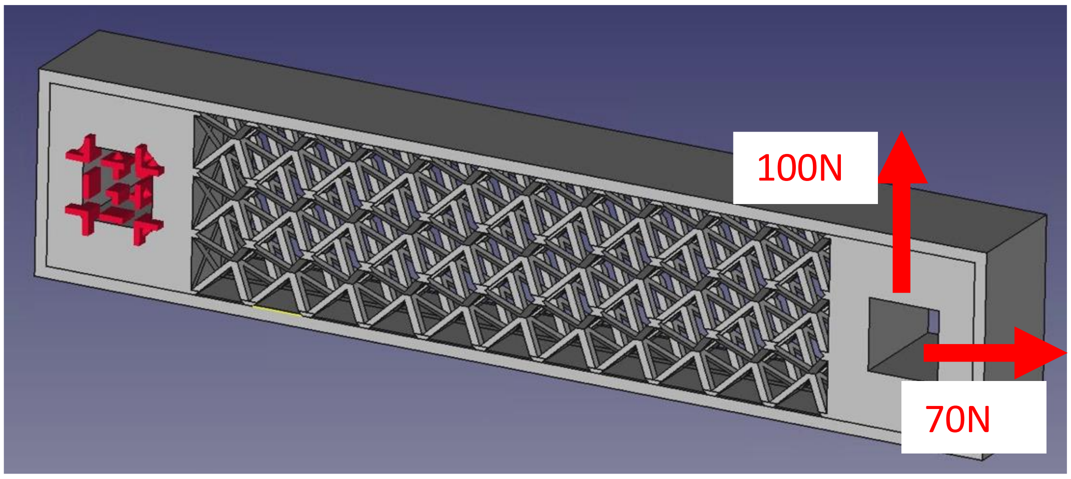


Figure 46 - Loading conditions for Case Study 4

After a little experimentation, a good performance envelope was found with the following value ranges.

Table 8 - Parameter randomisation ranges for Case Study 4

Parameter	Min	Max
StrutThickness	0.5 mm	2 mm
FrameThickness	0.5 mm	1.5 mm

Twenty generations were produced, analysed, and refined. Only part volume was checked in the Refinement panel, and the voxelisation resolution was increased to 2 voxels/mm because the typical resolution of 1 voxel/mm wouldn't be enough to accurately model the thin struts which could be less than a millimetre thick.

Six out of the twenty generations that were produced were comfortably within the 18 MPa peak stress limit. There was a good variety of parts generated that sufficiently encapsulated the desired range of allowable maximum stress and deflection was between. Peak stress ranged between 16.8 – 52 MPa and displacement was between 1.5 – 4 mm. Appendix S shows the full table of results for all the generations.

Generation 14 was chosen as the most optimal, with a **FrameThickness** of 1.38mm and **StrutThickness** of 0.87mm. Peak stress was 19.63 MPa, maximum deflection was 1.59mm, and part volume was 6917 mm^3. *This is a 38.7% decrease in volume of 11280 mm^3 compared to if the part was solid material instead.* Interestingly, this generation was among the lower quartile of generations in terms of part volume despite also being amongst the lowest in peak stress, where usually the contrary is true.

Figures 47 and 48 show the distributions of stress and displacement in the part, respectively.

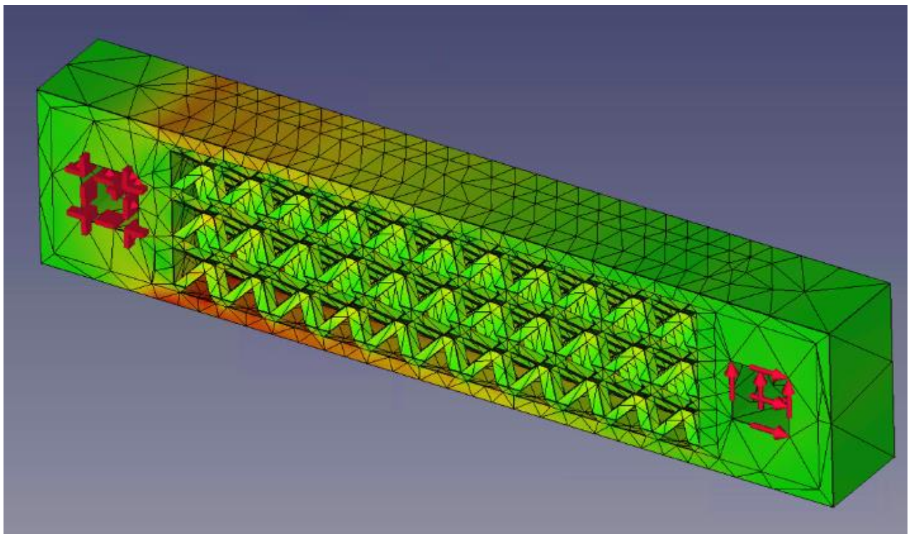


Figure 47 - Stress colourmap in generation 14

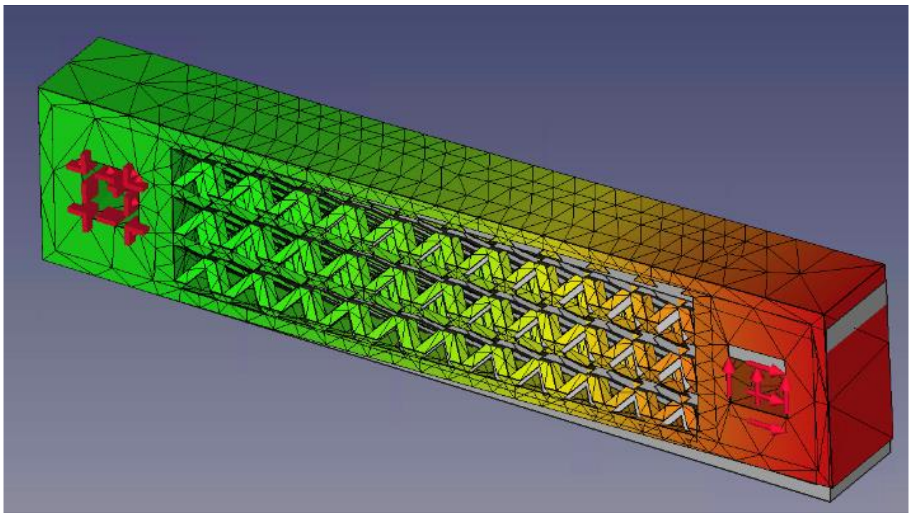


Figure 48 - Displacement colourmap with deflection for generation 14

Case Study 5 – GE Jet Engine Bracket

A popular design scenario that is used for testing the effectiveness of new generative design and additive manufacturing techniques is the GE jet engine bracket challenge. This was run as a competition in 2013, where contestants were given the objective of redesigning a mounting bracket that was to be additively manufactured and could tolerate certain loads with minimal part volume.

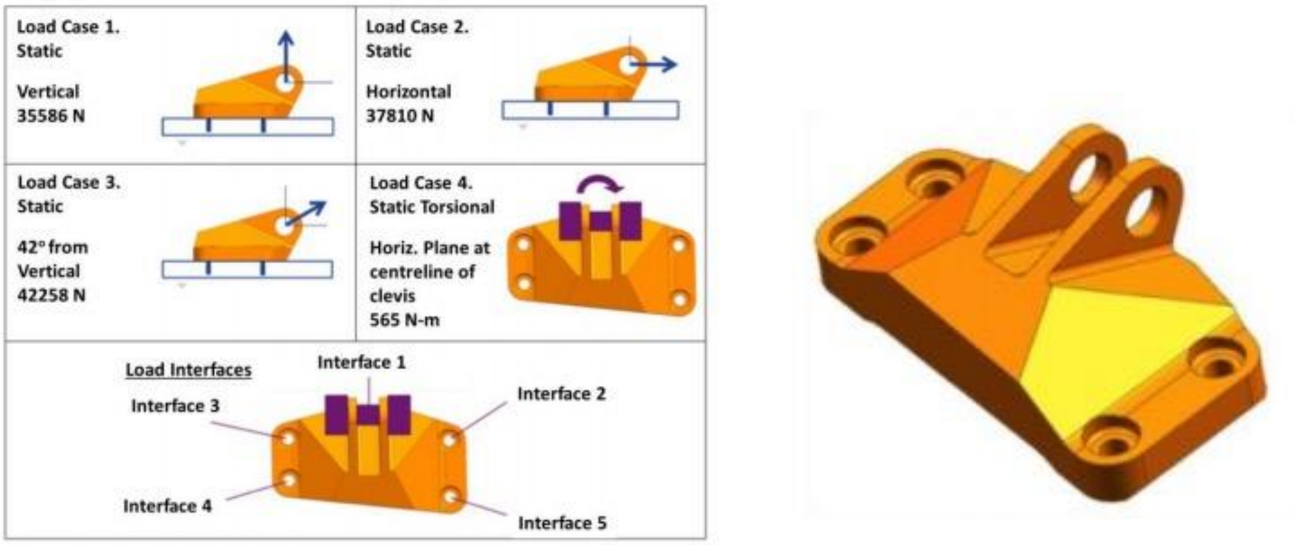


Figure 49 - Load cases and allowed material envelope for competition

The top 10 submissions were printed and tested by GE. A variety of approaches and design techniques were seen in all the submissions. In the end, the designs could be classified into four main categories of design (29)

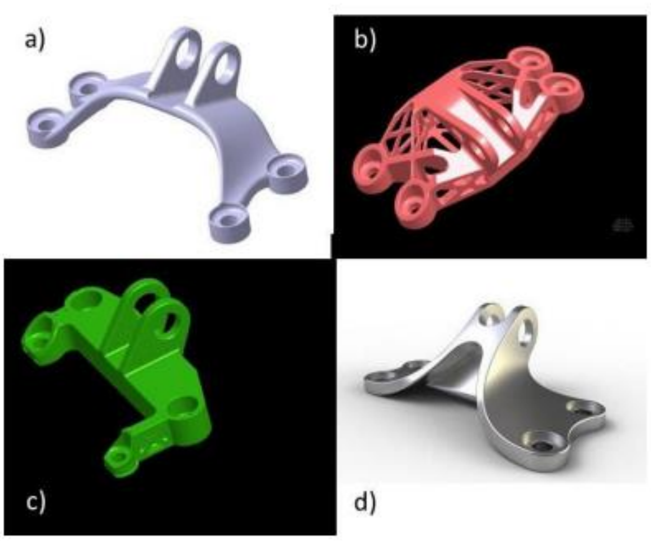


Figure 50 - The four main categories of design submitted

Categories a and d consists of features made out of swept profiles, category b is made from a skeleton of struts, and category c uses flat features. Many of these designs were informed of their topology using topology optimisation, then fine-tuned further manually.

A design inspired by category d was constructed using B-spline curves in FreeCAD. After a quick mechanical analysis of the part, it was discovered that most of the stress concentration was around bend in the sweep and the bolt holes. The flat parts comparatively had much less stress concentration in them, thus leaving room to optimise on the design by removing material.

The material of choice is Ti6Al4V with a tensile stress of 900MPa, which is a popular material in aerospace applications for its strength but low density. The competition specifies that the only criteria that must be satisfied is that the load doesn't fail via plastic deformation. With a safety factor of 1.5, this sets the *maximum allowable stress in the part as 600MPa*. Load case 1 from figure 49 was applied to the analysis portion of the part, with fixed supports on the countersunk holes.

A single strut element like the one used to make the lattices in Case Study 5 was constructed as a subtractive element then parameterised. Additionally, the thickness of the swept feature was parameterised.

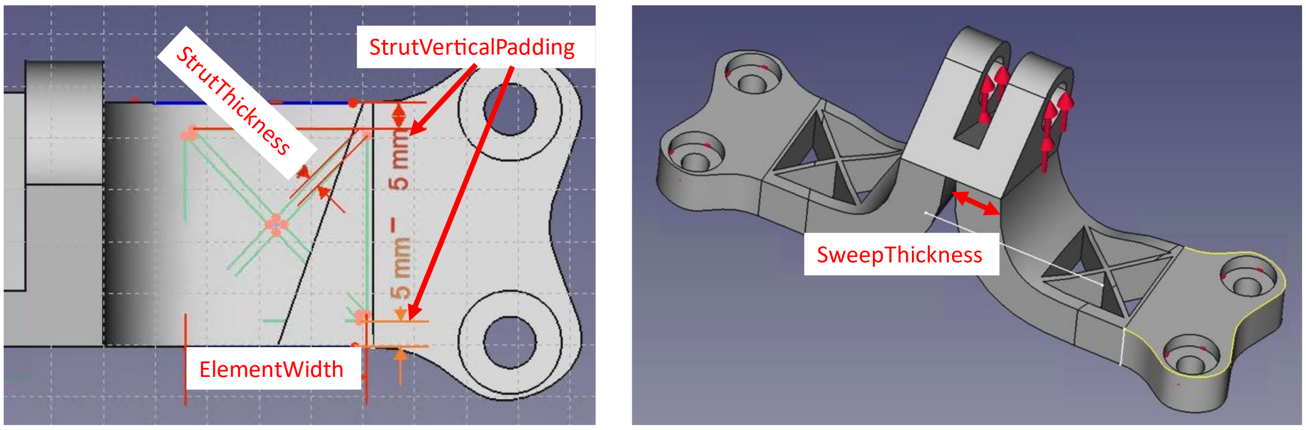


Figure 51 - Parameters to be randomised for Case Study 5

Table 9 - Parameter randomisation ranges for Case Study 5

Parameter	Min	Max
SweepThickness	15 mm	20 mm
StrutVerticalPadding	5 mm	10 mm
StrutThickness	5 mm	10 mm
ElementWidth	25 mm	35 mm

20 generations were produced with a good amount of variation in maximum stress. Eleven generations were within the 900MPa tensile strength limit of the material, and four generations were comfortably within the safety factor limit of 600MPa. The full list of results for all generations can be found in Appendix T

Interestingly, the generations that ended up with a higher part volume due to having a thicker **SweepThickness** also had less support volume. The generation chosen in the end was one of these parts, with a maximum stress of 567MPa, part volume of 241208 mm^3, and a support volume of 32533 mm^3.

Table 10 - Parameters for generation 16

Parameters	
SweepThickness	17.64 mm
StrutVerticalPadding	8.64 mm
StrutThickness	6.34 mm

ElementWidth	26.58 mm

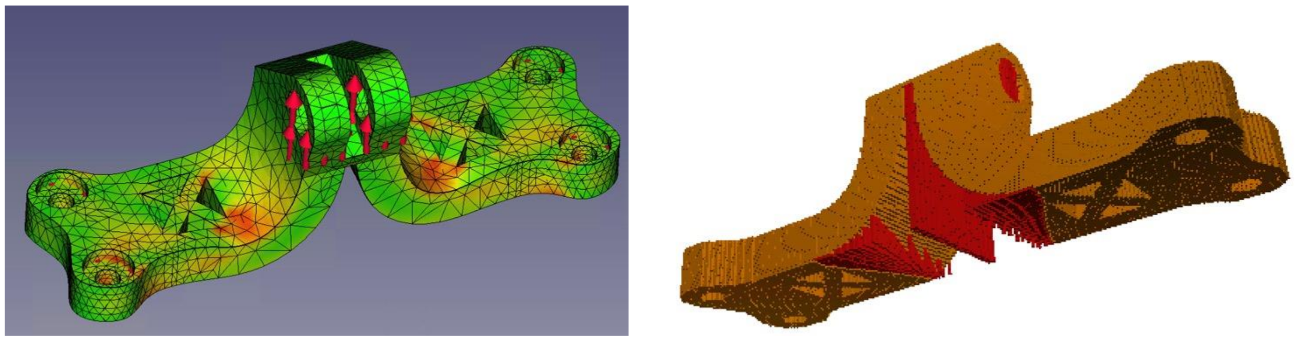


Figure 52 - Stress colourmap and voxelised part with support material for generation 16

Savings in material due to the lattice element are quite modest, with a reduction of about 10800 mm^3 in volume, or 4.3% of the part volume without the strut element. The ratio of material which comprised support structures was quite low at 0.1, which is thanks to the sharp sweep which minimises overhang at low horizontal angles.

The average stress in the generation was 88 MPa, which is quite a high ratio of the peak stress thus indicating that the part is topologically quite optimal. The histogram of stress confirms this, as there is a substantial number of nodes that are in the 100 – 300MPa stress range which indicates a large proprotion of material being utilised to its full load potential.

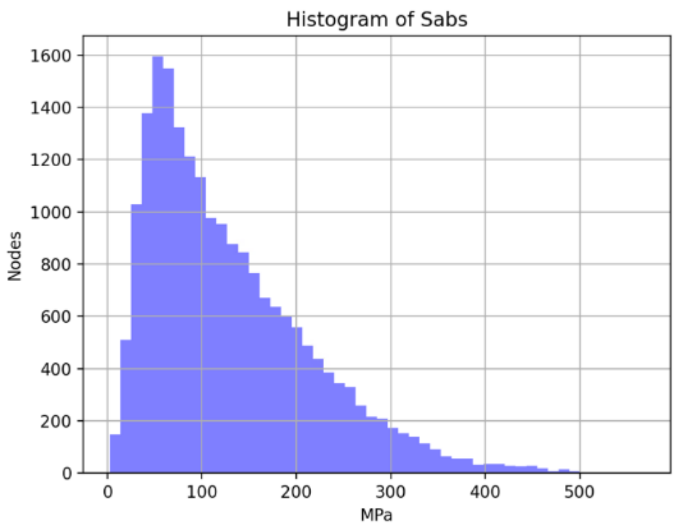


Figure 53 - Histogram of stress distribution across generation 16

Discussion

Across all of these case studies, the tool proved useful in optimising a design for a combination of criteria that involved mechanical performance, part volume, and support structure volume. Most part optimisations will be of the kind done in Case Study 1, where part volume is minimised while keeping within tensile stress limits for some specific loading conditions.

Despite the simplicity of the tool and its generation routine, the case studies show that it is very versatile and can be applied to benefit a variety of use cases if used properly. The simplicity of just inputting parameter ranges for setting up the process makes the process transparent and easily understood by the designer.

This is especially advantageous in the case of Case Study 3, where the process was applied iteratively to converge on a narrower design envelope. Case Study 3 also demonstrates that the tool is able to satisfy the criteria of supporting the designer's thinking process, as the first process iteration was used to inform the redesign of the part in the subsequent iteration.

One major downside that could be seen in all cases is the inability of the tool to self-improve and converge by itself. Because parameters are generated randomly, many generations are wasted on configurations that are clearly non-optimal. This might especially become a problem for more complex case studies, where the relationship between parameters and fitness is much less predictable and the optimal parameters lie in a very narrow region. This could be addressed by implementing an optimisation technique in the generation stage such as genetic algorithms or simulated annealing as explored in the literature review.

Another advantage is that the tool's functional transparency makes it easy for the designer to optimise a part for a particular feature, like with the lattice structure in Case Study 5.

The tool also meets the criteria of promoting flexibility in the designer's workflow. This made it possible to combine part features as in Case Study 4 which were very different and produced using different methods and optimise for them. It is also proof that the tool is capable of improving on designs that have been produced from different optimisation techniques.

All case studies took a reasonable amount of time to complete their entire process. No process iteration took longer than ten minutes to complete. The analysis stage usually was the largest performance bottleneck, which involved generating and solving FEA meshes. This stage could be quite fiddly, since it usually required hand tweaking the meshing parameters so that meshes would be generated correctly without any performance bottlenecks.

The ease of parameterising existing designs and simply inputting value ranges for the initialisation stage means that setting up requires little time and is convenient for the user to use. This convenience is exemplified in Case Study 2, where new variations for a bicycle seat were produced simply by parameterising it. In addition, the stages of the process are clearly modularised so the user can see exactly how the different stages advance the optimisation process, and more importantly see how they can tweak and modify them.

This tool is particularly unique in the sense that it is explicitly open-source and written in Python. This is much better than the alternatives available in Fusion 360 and CATIA since their optimisation tools are not open source, and are not built for expansion using their APIs and scripting languages. The author of this paper is not aware of any other project like this in existence that combines these features into an extensible, open-source product.

As it stands, the methods and algorithms use for generation, analysis, and refinement are relatively simplistic as there are many sophisticated methods available explored in the literature that would've greatly benefitted this project. However, many of these techniques have been the subject of theses and projects that have spanned several years by themselves, which made it infeasible to implement them into a project of this scope lasting just a few months.

Conclusions

This paper presents a proposal for a CAD plug-in which leverages a simple generative design routine to generatively optimise parts to be more suitable for additive manufacturing. A review of the literature reveals that there is a gap in generative design tools that assist the designer in producing optimal designs. The process of this tool was deduced down to a five stage methodology of Initialisation, Generation, Analysis, Refinement, and Ranking.

These five stages were then developed and tested modularly using FreeCAD as the environment to build the plugin on top of. These were then tied up underneath a Graphical User Interface (GUI) to make it intuitive and convenient to use for a typical designer. The tool was tested on five different case studies to verify its suitability in the context of typical design problems. Results show that the tool is quite effective at producing designs which are worthwhile and give quantifiable improvements.

The techniques used in each of the five stages are quite simple, as there was insufficient time during the project to implement anything too sophisticated. However, this is left as potential future work as this tool is meant to lay the foundations for making this overall procedure accessible.

Overall, this project is successful in demonstrating that generative design and Design for Additive Manufacturing can certainly complement each other. If these generative methods are leveraged in a way that makes them accessible and intuitive to designers that are accustomed to traditional paradigms, the proliferation of Generative Design in mechanical design and part production via 3D printing will be greatly accelerated.

Future work

Many features and techniques were researched that would've greatly benefitted this project. However, given the limited time of the project and limited knowledge of the student, these were unfeasible to develop at this level. This includes some key features that were mentioned in the project planning report.

Lattice generation

Lattice generation is a popular design tool that additive manufacturing is particularly well suited to taking advantage of. This was explored extensively early on in the project, with different latticing techniques compared to see which ones were more optimal than others in different use cases, in the hopes that it would be implemented as a refinement feature in the final tool.

In the end, this feature was scrapped because trying to automatically detect where lattices should have been generated was a fairly non-trivial task worthy of its own project. Requiring the designer to mark the regions of a part where latticing could be performed would've violated the requirement that the tool be intuitive enough to be used by a designer with no AM experience. Moreover, the intricate geometry generated by latticing would not be optimal for the FEA stage, as a large amount of nodes and elements in the mesh would be generated which dramatically increases computation time.

Computational offloading (GPU or other)

Many of the computationally expensive tasks performed by this tool could be easily parallelised. This includes voxelisation and FEA analysis. Moreover, because generations are analysed and refined independently from each other, it would be feasible to divide the workload between multiple workstations. This would be highly beneficial in producing many thousands of part generations, which might take hours on a single workstation PC.

Powerful GPU are ubiquitous in CAD workstations and have the potential to perform bulk maths operations much faster than the CPU, however they are only used for 3D rendering and thus often sit underutilised. The idea of offloading voxelisation to the GPU was explored briefly but later scrapped, as the CPU algorithm had sufficient enough performance on the PC used for testing that it wasn't worthwhile.

Shape grammars

Shape grammars are a very good way of developing complex and varied designs from simple rules. Zimmerman et al (15) developed a performance driven generative design framework which is built on top of another open-source tool developed for FreeCAD. The framework was applied to the design and optimisation of spokes for inline skate wheels. The results verified that shape grammars can generate structurally optimised designs within AM constraints.

Research on shape grammars also began early on in the project, but building an understanding of it proved to be quite difficult. There was not enough time to build an implementation in FreeCAD, as it is quite a niche technique so there are not many examples or libraries to build from. Hoisl (30) used FreeCAD to implement a visual and interactive 3D framework for design using spatial grammars, but this was discovered too late during development to be implemented.

Machine learning optimisation

The tool could do a more efficient job of exploring the design space and optimising for a particular metric by using machine learning between epochs of generation. It would be something similar to what Gunpinar et al (11) implemented where a sample of design parameters was generated and analysed, then the principal components were extracted to build a correlation model.

Implementing an ML optimization feature for this tool as it is currently implemented would be difficult as the generation algorithm is somewhat ineffective at producing parts that are optimal for additive manufacturing. If the generation stage was improved using features like shape grammars such that it could effectively improve parts for AM, it might be possible to see the effect of an ML technique.

References

1. Seepersad CC. Challenges and Opportunities in Design for Additive Manufacturing. 3D Print Addit Manuf. 2014;1(1):10–3.

2. Salem H, Abouchadi H, El Bikri K. Design for Additive Manufacturing: A Systematic Review. J Theor Appl Inf Technol. 2020;10(19):3043–54.

3. Ameen W, Al-Ahmari A, Mohammed MK. Self-supporting overhang structures produced by additive manufacturing through electron beam melting. Int J Adv Manuf Technol [Internet].

2019 Oct 1 [cited 2021 May 12];104(5–8):2215–32. Available from: https://doi.org/10.1007/s00170-019-04007-3

4. Bi S, Chen E, Gaitanaros S. Additive manufacturing and characterization of brittle foams. Mech Mater. 2020 Jun 1;145:103368.

5. Yap YL, Yeong WY. Shape recovery effect of 3D printed polymeric honeycomb: This paper studies the elastic behaviour of different honeycomb structures produced by PolyJet technology. Virtual Phys Prototyp [Internet]. 2015;10(2):91–9. Available from: https://doi.org/10.1080/17452759.2015.1060350

6. Sienkiewicz J, Płatek P, Jiang F, Sun X, Rusinek A. Investigations on the mechanical response of gradient lattice structures manufactured via slm. Metals (Basel). 2020;10(2).

7. Opgenoord MMJ, Willcox KE. Design for additive manufacturing: cellular structures in early-stage aerospace design. Struct Multidiscip Optim [Internet]. 2019 [cited 2021 Mar 26];60:411–28. Available from: https://doi.org/10.1007/s00158-019-02305-8

8. Flores I, Kretzschmar N, Azman AH, Chekurov S, Pedersen DB, Chaudhuri A. Implications of lattice structures on economics and productivity of metal powder bed fusion. Addit Manuf [Internet]. 2020;31(August 2019):100947. Available from: https://doi.org/10.1016/j.addma.2019.100947

9. Doubrovski Z, Verlinden JC, Geraedts JMP. OPTIMAL DESIGN FOR ADDITIVE MANUFACTURING: OPPORTUNITIES AND CHALLENGES. 2011;1–12.

10. Altaf K, Majdi Abdul Rani A, Raghavan VR. Prototype production and experimental analysis for circular and profiled conformal cooling channels in aluminium filled epoxy injection mould tools. Rapid Prototyp J. 2013 Jun 7;19(4):220–9.

11. Gunpinar E, Coskun UC, Ozsipahi M, Gunpinar S. A Generative Design and Drag Coefficient Prediction System for Sedan Car Side Silhouettes based on Computational Fluid Dynamics. CAD Comput Aided Des [Internet]. 2019;111:65–79. Available from: https://doi.org/10.1016/j.cad.2019.02.003

12. Singh V, Gu N. Towards an integrated generative design framework. Des Stud. 2012 Mar;33(2):185–207.

13. Krish S. A practical generative design method. CAD Comput Aided Des [Internet]. 2011;43(1):88–100. Available from: http://dx.doi.org/10.1016/j.cad.2010.09.009

14. Briard T, Segonds F, Zamariola N. G-DfAM: a methodological proposal of generative design for additive manufacturing in the automotive industry. Int J Interact Des Manuf [Internet]. 2020;14(3):875–86. Available from: https://doi.org/10.1007/s12008-020-00669-6

15. Zimmermann L, Chen T, Shea K. A 3D, performance-driven generative design framework: Automating the link from a 3D spatial grammar interpreter to structural finite element analysis and stochastic optimization. Artif Intell Eng Des Anal Manuf AIEDAM. 2018;32(2):189–99.

16. Xue T, Wallin TJ, Menguc Y, Adriaenssens S, Chiaramonte M. Machine learning generative models for automatic design of multi-material 3D printed composite solids. Extrem Mech Lett [Internet]. 2020;41:100992. Available from: https://doi.org/10.1016/j.eml.2020.100992

17. Hoisl F, Shea K. An interactive, visual approach to developing and applying parametric three-dimensional spatial grammars. Artif Intell Eng Des Anal Manuf AIEDAM [Internet]. 2011 Oct 12 [cited 2021 Apr 27];25(4):333–56. Available from: http://designmasala.com/miri/shaper2d/;

18. Zhang Y, Wang Z, Zhang Y, Gomes S, Bernard A. Bio-inspired generative design for support structure generation and optimization in Additive Manufacturing (AM). CIRP Ann [Internet]. 2020;69(1):117–20. Available from: https://doi.org/10.1016/j.cirp.2020.04.091

19. Sigmund O, Aage N, Andreassen E. On the (non-)optimality of Michell structures. Struct Multidiscip Optim. 2016;54(2):361–73.

20. Doubrovski EL, Tsai EY, Dikovsky D, Geraedts JMP, Herr H, Oxman N. Voxel-based fabrication through material property mapping: A design method for bitmap printing. CAD Comput Aided Des. 2015 Mar 1;60:3–13.

21. Vaidya R, Anand S. Optimum Support Structure Generation for Additive Manufacturing Using Unit Cell Structures and Support Removal Constraint. Procedia Manuf [Internet]. 2016;5:1043–59. Available from: http://dx.doi.org/10.1016/j.promfg.2016.08.072

22. Opgenoord MMJ, Willcox KE. Design for additive manufacturing: cellular structures in early-stage aerospace design. Struct Multidiscip Optim. 2019;60(2):411–28.

23. CALCULIX: A Three-Dimensional Structural Finite Elemente Program [Internet]. [cited 2021 Apr 27]. Available from: http://www.calculix.de/

24. Pederkoff C. GitHub - cpederkoff/stl-to-voxel: Turn STL files into voxels, images, and videos [Internet]. [cited 2021 Apr 29]. Available from: https://github.com/cpederkoff/stl-to-voxel

25. Katharopoulos A. simple-3dviz [Internet]. [cited 2021 Apr 29]. Available from: https://simple-3dviz.com/

26. Python parser for CalculiX .frd result files ($238) · Snippets · Snippets · GitLab [Internet]. [cited 2021 Apr 29]. Available from: https://gitlab.lrz.de/snippets/238

27. Hearing: 12.2 Interaural time delays: non-continuous sounds - OpenLearn - Open University - SD329_1 [Internet]. [cited 2021 May 3]. Available from: https://www.open.edu/openlearn/science-maths-technology/biology/hearing/content-section-12.2

28. PLA vs. ABS: What’s the difference? | 3D Hubs [Internet]. [cited 2021 May 3]. Available from: https://www.3dhubs.com/knowledge-base/pla-vs-abs-whats-difference/

29. Morgan HD, Levatti HU, Sienz J, Gil AJ, Bould DC. GE Jet Engine Bracket Challenge: A Case Study in Sustainable Design [Internet]. 2014 [cited 2021 May 8]. Available from: http://www.inimpact.org

30. Hoisl FR. Visual , Interactive 3D Spatial Grammars in CAD for Computational Design Synthesis. Dissertation. 2012.

Appendices