Topology Optimisation with Finite Element Analysis

AQA A-Level Computer Science Non-Exam Assessment Documentation - James Bray 2021/2022

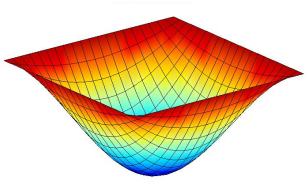
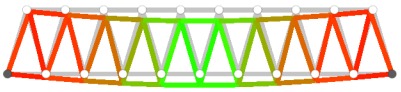
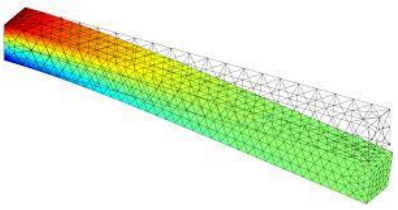
Contents

Analysis

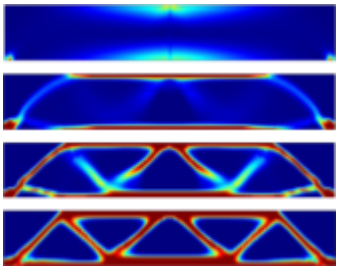
Introduction

When designers, architects and engineers are developing structures and buildings, they need to consider the types and amounts of each material used, as well as the structural integrity of each object. The aim of this project is to create a software application to accelerate the designing process, allowing engineers to analyse their structures, or to find more optimised solutions whilst maintaining structural integrity.

After researching the field, I found two algorithms that when used together could provide this capability. The outcome of the first algorithm explores the behaviour of structures based on their form and the forces acting on them at certain points. This is called **Finite Element Analysis** (FEA).

The second is called **Topology Optimisation** and utilises FEA to generate optimal structures given only the fixed points and forces. These generated structures were not commonly used in the past as they generate unique and abstract structures which the tools available in the industry were not capable of manufacturing. However, the tools today are more capable, making this approach more relevant due to its applications and cost-effective use of materials.


The end goal of this project is to build an application that makes it easy for users to input simulation features and generate an optimal model to match them. This will produce structures of minimal material, which improves any costs, weight issues and structural integrity problems an engineer might have to face when designing anything, improving productivity in their job.

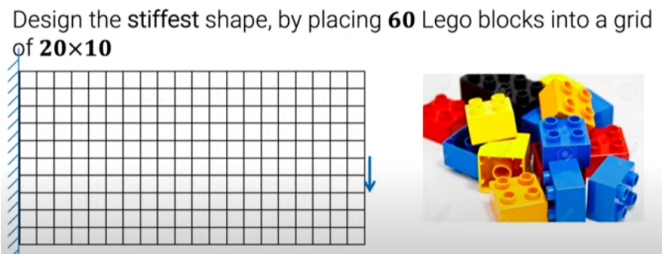
The End-User

The general users will be engineers or designers who would be designing a 2-dimensional structure requiring optimal use of materials. I will be providing my project to an A-level peer Boris (who is going into a career in Civil Engineering and Architecture) as a primary end-user. Additionally, Boris has access to a 3D printer and often thinks that the prints are wasteful and could be optimised. This software will help him generate better designs if the print is only designed for its strength and minimal material uses, and aesthetics are not a factor.

I will interview him before and after the project is made. I chose Boris due to his competency with computers and software as he is a computer science student. Boris also studies physics and engineering, this means I wouldn't need to focus so much on describing specialist terminology, and when he is operating the software, he will most likely understand the functionality.

Modelling the Problem

This software has many uses, however, for the sake of explaining the problem, I will use a specific case. The case I will be using is covered in this video: https://youtu.be/5ocnVS_HvdY

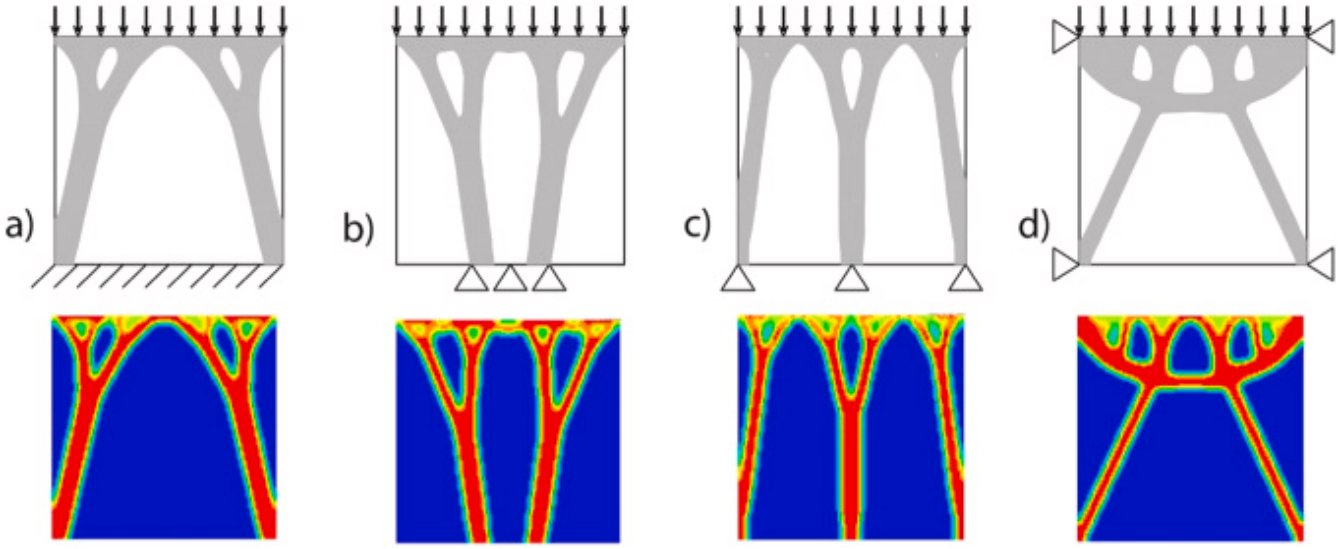


As seen on the right, let's say you were tasked with building a structure to hold up a force, with a limited amount of building material. Finding the stiffest possible structure is easy when you have unlimited material, but as you can only fill 30% of the space with lego, you'd need to determine how to strategically place the blocks for the stiffest structure

As seen before in the TopOpt interface, the force acting upon the structure is shown by an arrow, and the fixpoints are modelled along the left wall as a hatching pattern. In the correct terminology, these are known as the boundary conditions, which represent a requirement to be satisfied. The amount of the material used (a.k.a. the volume fraction) is also a boundary condition as that needs to be satisfied as well in this case. For general purposes, the user will input the boundary conditions so the app can run.

What we could do to solve this problem is try every possible combination of pieces and evaluate the stress after each one. However, there are 7.04×10^{51} combinations, which is more than the estimated number of stars in the universe which is 10^{24}. This means this solution is intractable.

To solve this problem, we need to use iterative approximation to guess where the best place to place a lego brick is. To do this, we start by filling the space with blocks, then we use FEA to find which bricks are undergoing the most stress. Then we can see which blocks we need to hold the force, and which ones aren't needed. So the algorithm eliminates bricks from the structure and after every iteration, the structure becomes stronger, until it is at its best. Then we have reached the optimal topology for the given boundary conditions.

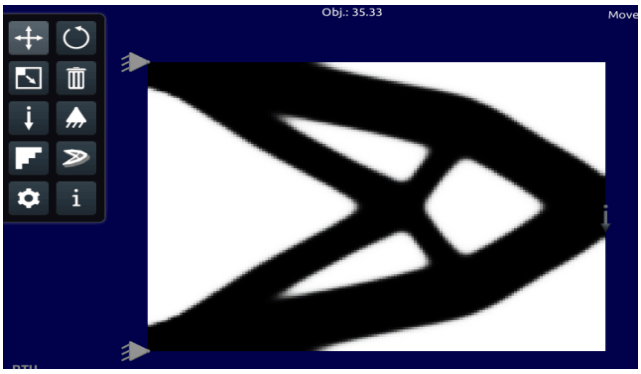


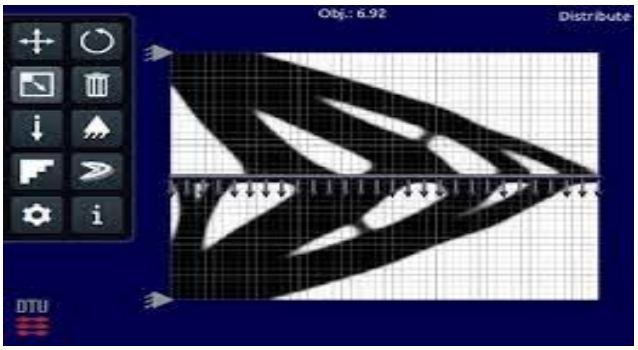
As seen above, however, this structure building app has many purposes, and it isn't just used to solve that one problem. As seen, maybe one structure (a) has the ability to use the entire ground to build from, but others (b and c) only have specific points. The purpose of my application will be to optimise models for different users with different structural requirements but will be tested on a small number of cases to evaluate the functionality.

Secondary Research

TopOpt Interface

TopOpt 2D is an existing solution that finds the optimal structure on a 2D plane, with forces and fixed points acting parallel to the plane. This application was developed by the Technical University of Denmark. This software was written using MATLAB and Python, with a front-end developed using Unity.





How TopOpt Receives Inputs
Initially, the program allows the user to input the conditions for the topology. The inputs are all built for manipulating icons that control these conditions. There are two types of "icons" that the user can put into the model, Fixpoints and Forces. These are shown below:


(Fixpoint)

This is to show a point in the structure that can't be moved


(Force)

This shows a force that acts upon a given point in the direction of the arrow

As seen on the left there is a toolbox. The first three tools are move, rotate and scale which are for manipulating existing icons. The next four are for adding and removing constraints and more advanced settings. There's also a volume slider, which is the amount of material the model is allowed to use

TopOpt ISPO

An ISPO such as the following describes what happens after each input is used. Each button/UI element causes data storage and subsequently a process, then an output. The features listed below are from the TopOpt application, and some will be shared with my solution. This table is useful as it outlines the main structure of the system, as well as considering the storage needed to allow the system to function.

Input	Storage	Process	Output
The user presses one of the add force or fixed point buttons	The location of the new item is added to the code	An icon is created for the user to drag and drop onto the model	An icon appears where the user specifies on the screen
The user edits fixed forces or nodes using move, rotate or scale	The properties of the change are stored in the algorithm	The program is changed to the tools "mode" (e.g. scale)	Icon adapts to the mode the user is in and changes visually
The volume constraint is changed	Value gets changed in the code	The algorithm runs again with a new value	The model adapts to the new volume
The export model button is pressed by the user	The mesh file is stored as a file before getting ready to export	The generated mesh in the app is converted to a file	Pop-up for the user to choose where they want the file stored

Observations

This software runs as a mobile application, it makes dragging and dropping convenient, but most engineers use PCs for designing and editing structures, which means that a mobile app would not be as practical due to its small screen. This has led me to decide on making my application Windows/Mac/Linux based.

I like the system's drag and drop feel when adding features to the topology, along with the range of tools available to manipulate the topology. The developers have added passive void (which defines an area where the material can't be), rotating forces and a volume to strength ratio control. I also like the way the structure adapts to the changes you make in runtime, and hope to be able to make the processing efficient enough to be able to replicate this.

However there are limitations: the simulation is in 2D and represents only a basic shape, and I feel there's not enough data displayed. I would include numerical data alongside the visual model in my app in case the user needed to know more advanced details about their model. Additionally, the deformation and stresses in the structure are hidden from the user and I feel it would be useful to use a colour gradient to show which area the stress is most intensive.

Both apps have a number called the objective number, which is the number that determines how close the structure is to convergence. Just watching the number change as the optimisation takes place isn't the best way of showing the user the optimisation process as it happens. I might use this number to plot a graph against time, so the user can see how the structure improves.

Consideration of Features

Below is a list of features I feel would be a good idea to implement into my project based on the features I have seen and used in TopOpt2D:

Must Have

- I want to include drag and drop functionality in my software for adding constraints. I want to develop an editor specifically for dragging and dropping fixpoints and forces.
- To move constraints around, they need to be placed in a grid, so I need to make a grid that is visible to the user, and after dropping an icon, it needs to snap to the nearest point.

Should have

- Different windows for the editor and the optimisation results. I would like to separate the inputs from the outputs to make them easier to distinguish for the user, and since the screen size is larger, the constraint editor will still be a reasonable size.
- In the TopOpt app, the tools aren't labelled and the help page is currently a link to a non-existent web page. I will change all of my tool buttons to labels, also I'll include a help page instructing the user on how to use the app ensuring the user doesn't get confused trying to operate my program.

Could have

- I will consider adding the scale and rotation buttons for constraints, as it reduces the number of objects stored in the UI, and allows for more efficient modelling.
- A deformation visualiser, as when the model is generated in TopOpt, there is no way of seeing how the model displaces under the force.

Won't have

- I won't include tools such as 'passive void' and 'passive material', which define areas that must be either solid or void. I think this will be achievable in the given time frame.
- I won't provide the ability to export the user's mesh, because I think connecting a file explorer so the user can define where the mesh is saved will be too complex of a task.

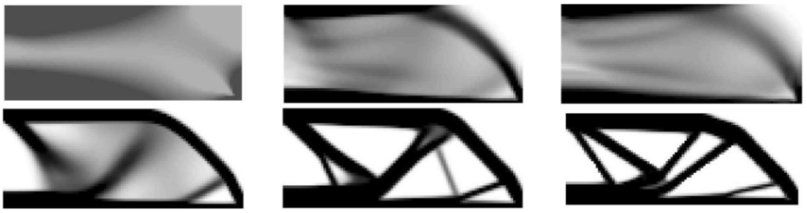
Complex Algorithms

Topology Optimisation

Topology optimisation is an umbrella term for optimising any geometric object under continuous deformation, and there are multiple methods to produce the same outcome. I have researched all of these and found two methods that I feel are best suited for this project.

Solid Isotropic Material with Penalisation (SIMP)

The SIMP method starts by taking a domain that it can work in. This method breaks the domain into elements, being either a square or a voxel for 2D or 3D. Each element has a density value between 0 and 1 which is initially set to the volume constraint (the fraction of the domain which is set as material). The density in the whole structure should equal the volume constraint and the algorithm redistributes it iteratively using FEA. It's best to think of this process like clay or play-dough. You start with a set amount, and you shift it around and test it until you're happy with the outcome. The density is just the amount of clay in that part of the model.

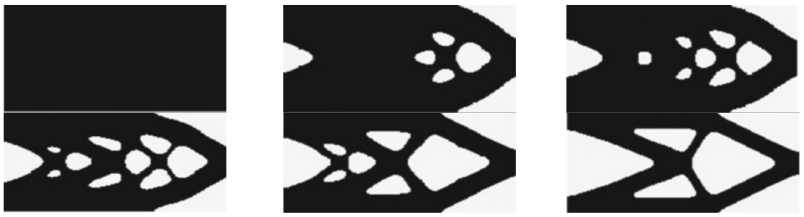


fig(1.0) - A topology optimisation process using the SIMP method

As the structure is being changed, there is a penalisation method that causes the density to tend to 1 or 0 quicker in each iteration. When it reaches the point where density isn't changing much, the model has converged to the optimal solution.

Evolutionary Structural Optimisation (ESO)

Evolutionary Structural Optimisation works similarly to the SIMP method such that it breaks the domain into elements and uses FEA to find the stress in each of the elements. But instead of redistributing the density, it starts with the whole domain set to 1, so it's all material. Then, after analysing the way the material behaves under deformation, it removes material that is superfluous to the design. This process is almost like whittling, as it just keeps removing material until it reaches the required amount of material left.



fig(1.1) - A topology optimisation using the ESO method

This is also an iterative process that analyses the structure of each iteration. The algorithm stops when the structure can't remove any more material without breaching the volume constraint. This is when the optimal solution has been produced.

Chosen Method

I have decided to use the SIMP method because I feel that it will be easier to set densities to 1 or 0 than to remove elements from the FEA entirely. The penalisation also allows for a quicker convergence, therefore it makes for a more efficient solution.

If I have time at the end, since the methods are very similar, I might be able to run them both side by side on the same optimisation and evaluate the effectiveness of each method. That will allow my app to generate 2 different structures which in theory, should be the same.
Since both methods use Finite Element Analysis, I did some research on how that works and how to generate stress values in each element using FEA.

Optimisation Flowchart

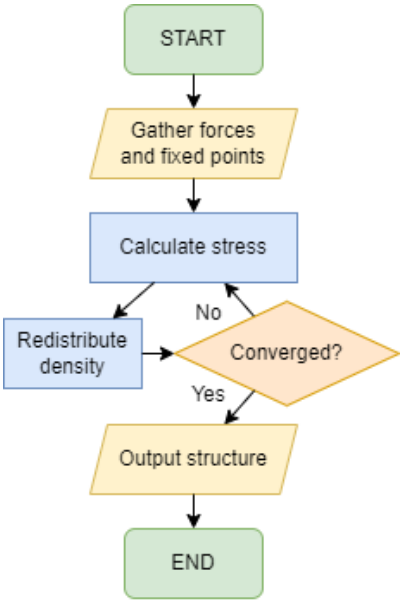
This is the general method for topology optimisation with finite element analysis, showing the steps and the order they need to be executed.

Fixed points and forces are taken from the UI of the app, where the user drags icons where they want the forces to be. Calculating stress, redistributing density and convergence is covered in complex algorithms, but here is the order in which they are used, and as seen, it's an iterative process.

This flowchart is an expanded version of the one in this video: https://youtu.be/e-F_piSm3Fc

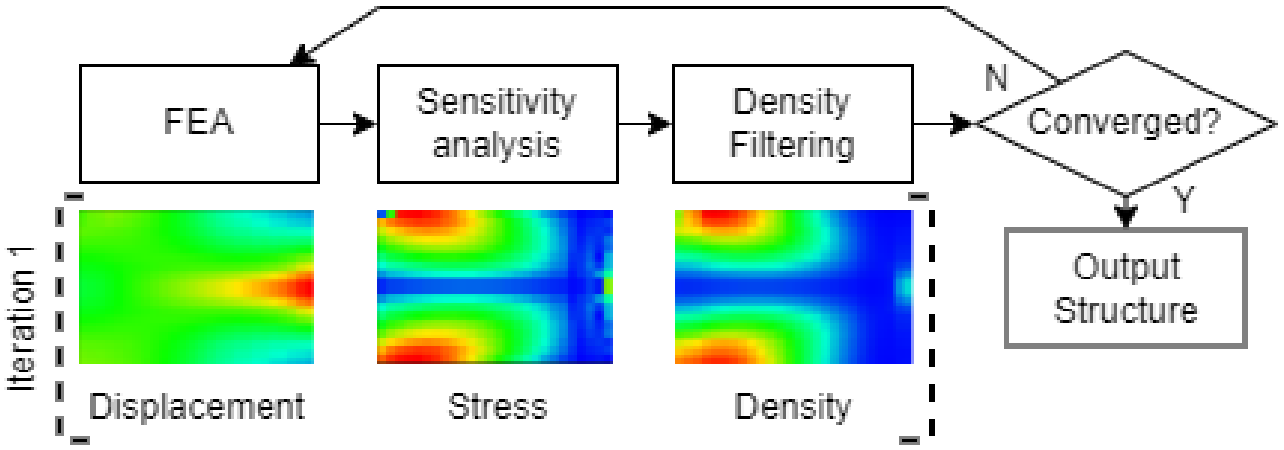
This process applies to both TopOpt and my application as this is the general structure for many topology optimisation processes

The main difference between the two algorithms is how they redistribute density in the structure. The following shows how the SIMP method does this.



Flowchart of SIMP Method

The following diagram shows the flowchart of topology optimisation and what I think the results of each stage in the first iteration is going to look like.



SIMP Pseudocode

The code below is taken from TopOpt's 99 line code, and demonstrates how to perform the steps shown in the flowchart

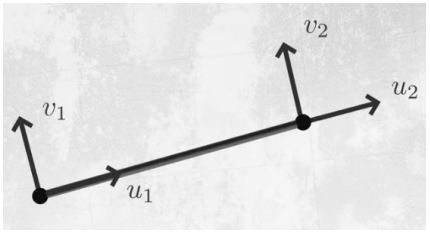
```
// Topology Optimisation
function top(nelx, nely, volfrac, penal, rmin);
	x(1:nely, 1:nelx) = volfrac; loop = 0; change = 1.0;

	// Start Iteration
	while (change > 0.01)
		loop = loop + 1; xold = x;

		[U] = FE(nelx, nely, x, penal);

		// OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
		[KE] = lk; c = 0.;
		for ely = 1:nely
			for elx = 1:nelx
				n1 = (nely + 1) * (elx-1) + ely;
				n2 = (nely + 1) * elx + ely;
				Ue = U([2 * n1-1; 2 * n1; 2 * n2-1; 2 * n2;
					2 * n2 + 1; 2 * n2 + 2; 2 * n1 + 1; 2 * n1 + 2], 1);
				c = c + x(ely, elx)^penal * Ue' * KE * Ue;
				dc(ely, elx) = -penal * x(ely, elx)^(penal-1) * Ue' * KE * Ue;

		[dc] = check(nelx, nely, rmin, x, dc); // Sensitivity filter
		[x] = OC(nelx, nely, x, volfrac, dc); // Design Update

	// Match the user-defined user volume (bisection algorithm)
	function OC(nelx, nely, x, volfrac, dc)
		l1 = 0; l2 = 100000; move = 0.2;
		while (l2-l1 > 1e-4)
			lmid = 0.5 * (l2 + l1);
			xnew = max(0.001, max(x-move, min(1, min(x+move, x*sqrt(-dc/lmid)))));
			if sum(sum(xnew)) - volfrac * nelx * nely > 0;
				l1 = lmid;
			else
				l2 = lmid;
		return(xnew);

	// Mesh independency filter
	function check(nelx, nely, rmin, x, dc)
		dcn = zeros(nely, nelx);
		for i = 1:nelx
			for j = 1:nely
				sum = 0.0;
				for k = max(i-round(rmin), 1): min(i + round(rmin), nelx)
					for l = max(j-round(rmin), 1):
						min(j + round(rmin), nely)
						fac = rmin-sqrt((i-k)^2 + (j-l)^2);
						sum = sum + max(0, fac);
						dcn(j, i) = dcn(j, i)+max(0, fac)*x(l, k)*dc(l, k);
				dcn(j, i) = dcn(j, i)/(x(j, i) * sum);
		return (dcn);
```

Finite Element Analysis (FEA)

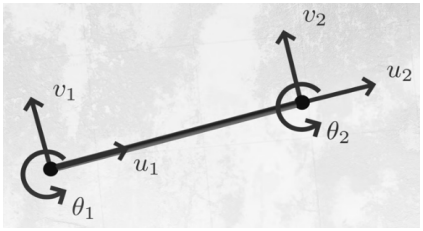
Source of information: https://youtu.be/GHjopp47vvQ

Finite Element Analysis breaks a structure into nodes and elements which form a model. For example, a bar contains 2 nodes at either end and the element would be the bar. In a 3D simulation, the elements would be voxels, and the nodes are the vertices of the voxel.

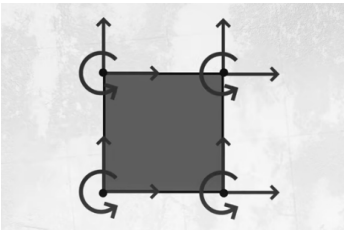
Each node has a given number of degrees of freedom (DOF) depending on modelling accuracy. A DOF represents a direction or rotation a node can move in once a force is exerted on it. Each DOF can also be called a displacement, which simply means a change in distance or angle.



fig(2.0) - 2D bar element with 2 DOFs, only including axial displacements



fig(2.1) - 2D beam element with 3 DOFs, including rotation



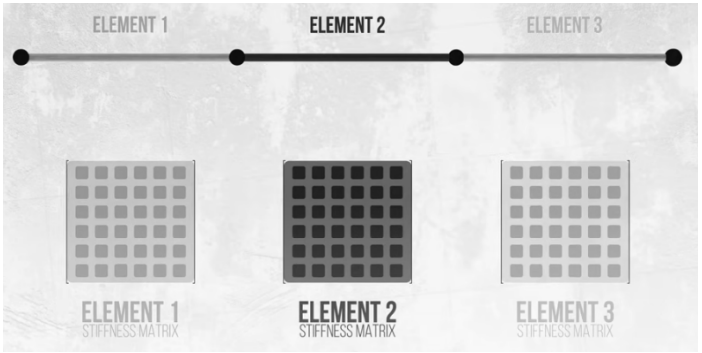
fig(2.2) - A 2D Quad element with 12 DOFs, including rotation

For each element in the model, linear equations are formed to generate a matrix called the Local Stiffness Matrix (LSM) which determines the effect each node has on other nodes in the element. After all required matrices have been formed, they are all assembled into one Global Stiffness Matrix (GSM). This is because elements share nodes, and the effect one element has on a node changes a neighbouring element, also, it reduces the number of values that need to be stored.

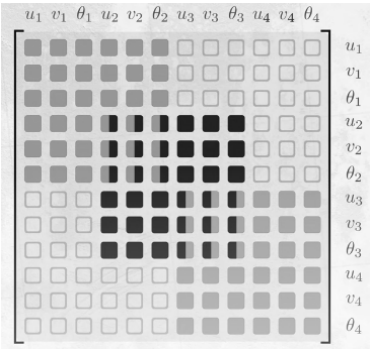
The GSM is a large system of linear equations determining the effect each force and displacement at each node has on every other node's forces and displacements. k_{ab} is the stiffness constant relating force *a* with displacement *b*.

$$\begin{Bmatrix} f_{x1} \\ f_{y1} \\ m_1 \\ f_{x2} \\ f_{y2} \\ m_2 \end{Bmatrix} = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} & k_{16} \\ k_{21} & k_{22} & k_{23} & k_{24} & k_{25} & k_{26} \\ k_{31} & k_{32} & k_{33} & k_{34} & k_{35} & k_{36} \\ k_{41} & k_{42} & k_{43} & k_{44} & k_{45} & k_{46} \\ k_{51} & k_{52} & k_{53} & k_{54} & k_{55} & k_{56} \\ k_{61} & k_{62} & k_{63} & k_{64} & k_{65} & k_{66} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \theta_1 \\ u_2 \\ v_2 \\ \theta_2 \end{Bmatrix}$$

fig(3.0) - a global stiffness matrix for element in fig(1.1)



fig(3.1) - a simplified representation of 3 LSMs for a model using the elements in fig(2.1)

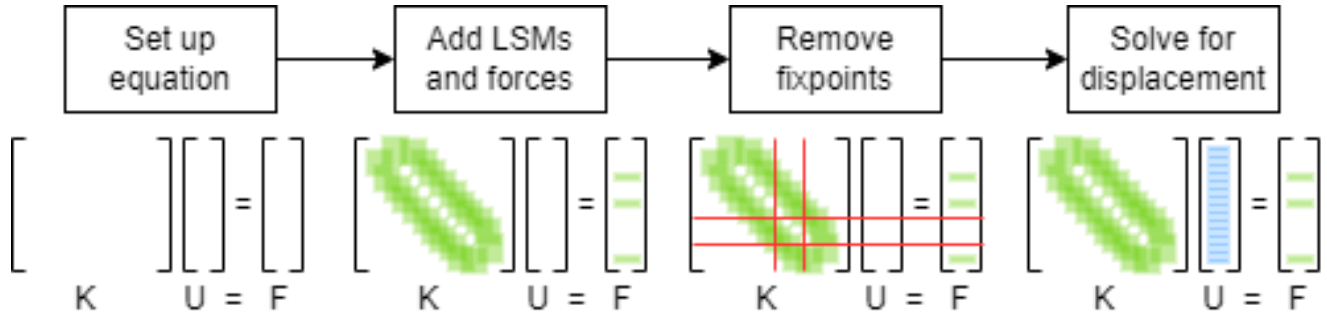


fig(3.2) - how the LSMs fall into the GSM for the model in fig(2.1)

The above example shows how to assemble the global stiffness matrix for three beam elements connected in a line. The element stiffness matrices are assembled and added into the GSM where the DOFs are the same. This is what is solved alongside the force vector, to find the displacements.

Flowchart of FEA

The following diagram shows what the matrix equation looks like during the process, where colours show the numbers inside the matrix. This is the process that the following pseudocode follows to find the displacement.



FEA Pseudocode

This code is taken from the TopOpt 99 line documentation and represents how they perform FEA on an example structure.

```
// Calculate the displacement of each Node in the Structure (FE Analysis)
function FE(nelx, nely, x, penal)
	[KE] = lk;
	K = sparse(2 * (nelx + 1) * (nely + 1), 2 * (nelx + 1) * (nely + 1));
	F = sparse(2 * (nely + 1) * (nelx + 1), 1);
	U = sparse(2 * (nely + 1) * (nelx + 1), 1);
	for ely = 1:nely
		for elx = 1:nelx
			n1 = (nely + 1) * (elx - 1) + ely;
			n2 = (nely + 1) * elx + ely;
			// Return the 2 DOFs for each of the 4 Nodes of an Element
			edof = [2 * n1 - 1; 2 * n1; 2 * n2 - 1; 2 * n2;
				2 * n2 + 1; 2 * n2 + 2; 2 * n1 + 1; 2 * n1 + 2];
			K(edof, edof) = K(edof, edof) + x(ely, elx) ^ penal * KE;

	// Assembles the force vector using the one force needed for a half mbb beam
	// (Note a:b:c means an array from a to c in increments of b)
	F(2, 1) = -1;
	fixeddofs = union([1:2:2 * (nely + 1)], [2 * (nelx + 1) * (nely + 1)]);
	alldofs = [1:2 * (nely + 1) * (nelx + 1)];
	freedofs = setdiff(alldofs, fixeddofs);
	//i.e. for any size of problem, generate half MBB beam with fixed DOFs

	// Solving (backslash means solve for displacements)
	U(freedofs, :) = K(freedofs, freedofs) \ F(freedofs, :);
	U(fixeddofs, :) = 0;
```

Possible Solutions

Choosing a Language

Python
Python is a very popular language to use, due to how readable the code is. It has a very large community, meaning assistance and documentation is abundant. Python also has many libraries which I can use to assist the mathematical side of my system, one of them being NumPy. I am very familiar with this language, and it has 2 interface modules that could be useful:

- **Pygame** is a 2D development module with an emphasis on game design, so objects and shapes are easy to draw and control. Pygame doesn't have any buttons or text boxes, however, limiting its use for user interface design.
- **OpenGL** is a module for python which allows the use of a 3D simulation environment with fast graphics. It is really good for plotting 3D objects, however, gathering user inputs is difficult, and I would probably have to make a command-line interface (user typed commands) for inputs and a separate window for the structures. I don't think this is ideal, as some structures need constant editing and swapping between two windows is inconvenient.

C# in Unity
C# is a compiled object-oriented programming (OOP) language, which makes it fast and ideal for mathematically intensive code. An OOP is ideal because my project might need to make use of instantiating and manipulating multiple objects. Unity has a wide community so help is easy to find if I were to need it. Unity makes use of C#, combined with many packages and inbuilt properties which makes the development process easier - for example, Math.NET will be useful for dealing with projects like mine and will make my code a lot cleaner.

Justification of chosen solution
I am choosing Unity because Pygame isn't 3D, OpenGL is complex, and inputs are difficult. Also, I have prior knowledge of C# and the software, allowing me to focus on the functionality of the project. Unity has a wide range of capabilities when it comes to building the application, therefore I can distribute it on many platforms easily.

Potential Libraries

Given that I am choosing unity, I need to find C# libraries to implement which make the development process easier. The following are the most useful and I am probably going to make use of all of the following:

MathNet.Numerics.LinearAlgebra is a .NET library that simplifies matrix and vector manipulation. The library includes data types that allow for large matrices and vectors compared to unity's built-ins which only allow small dimensionality. The library also includes methods that simplify my code and allow me to focus on the functionality of the algorithm. For example, solving a large system of linear equations. MATLAB has a built-in function called by a backslash which solves automatically but C# doesn't, therefore this package is needed.

System.Linq is a library for Language-Integrated Queries (LINQ) which allows for queries to be called on C# data types. The query format is similar to SQL and the methods allow for quick retrieval in large data structures. For example, if I needed to find the maximum value in a list, I would have to loop through each value using a temporary value to find the max, but this library allows me to do that in a single method call, making my code easier to read and maintain.

Universal RP is a Unity specific package I need to install if I want my meshes to be coloured. The package provides the ability to write custom shaders using a shader graph and this means I can build a vertex shader that colours the mesh at each vertex and blends the colours on the faces.

Primary Research

Interview of End-User

I have interviewed Boris and spoken to him throughout the analysis stage of this project - he has provided useful feedback which has helped me make some of the choices for this project. Below are a few of the interview questions I found most valuable:

Should I prioritise the quality of the 2D or the 3D topology optimisation?
I feel like 3D would be the most realistic environment to optimise the topologies as we live in a 3D world, however, I understand the complexity of a 3D environment and editor, and that will be a hard thing to accomplish successfully. Therefore I think you should try to make 2D optimisation your priority, as the environment and editor will be of a higher quality.

What platform would you like the system to be compatible with?
In my experience, Windows is the most popular operating system in the engineering industry. It would be great if the app could be made for different systems like Mac or Linux, even mobile could be a possibility, but Windows is the priority for me.

Would you like to be able to save and export structures?
Of course, it would be a great function if I could export my structures, they could be sent straight to the 3D printer or other CAM machines. I also think saving will be a valuable feature because if I needed to revisit a structure that I needed to change slightly, it would save a lot of time.

Do you know of any existing software?
I have heard of several analysis applications, which are all built for other large companies and are too expensive for me to use, so I couldn't tell you about them. I have seen one app called TopOpt which has done this before, and their software is free for mobile devices.

What are the positives and negatives of the current system?
TopOpt generates structures quickly, but it's only on mobile and the editor is not very aesthetically pleasing. Also, as I already mentioned, most software is too expensive to use, so free software will be a great thing to have when I start designing things that require it.

Interview Synopsis

There are many things these questions have told me about how my project will look and perform. I will be doing 2D, but I will need to consider the 3D editor in more detail because as Boris said, I would need to build an editor in 3D for the user to be able to input the conditions.

For this project, I will prioritise Windows as a target operating system, although choose a language that will enable future versions to be created for other platforms such as MacOS and Linux.

I agree with Boris in terms of saving and exporting the structures, which is something I will need to take into consideration when designing the project. I have done some research into the best file types for generated meshes, and so far what I have found is that STL, OBJ and FBX are the most popular.

I hadn't heard of the TopOpt application before my interview with Boris, therefore I feel researching how it works and what features the developers included could give me an insight as to what would be good to add to my software and how to improve upon what already exists.

Objectives

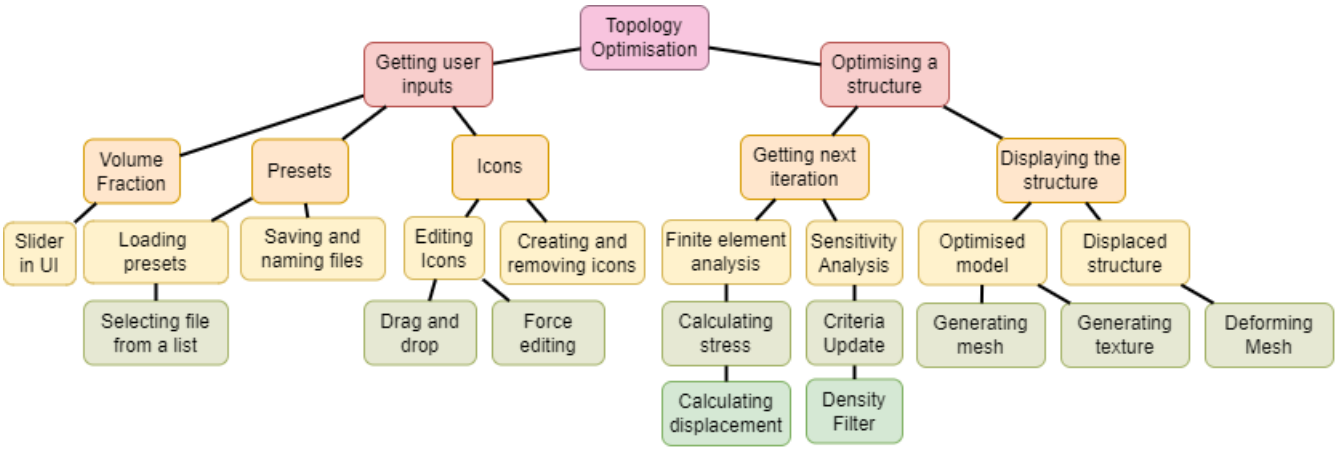
1. **General** (The overall requirements my system needs to meet)
   - 1.1. The system should run on a windows platform as an exe file
   - 1.2. The code should optimise a structure in under a minute.
   - 1.3. Inputs in the environment should be clear and validated before processing
   - 1.4. Outputs should be intuitive and clearly labelled.
2. **Finite Element Analysis** (The ability to calculate how a structure displaces under a force)
   - 2.1. My program should be able to fetch coordinates of the constraints on an editor grid
   - 2.2. My code should be optimised to be able to efficiently solve for displacements
   - 2.3. My UI should have a way of representing the displacement of the structure
     - 2.3.1. My system could either move the structure to visualise the deformation
     - 2.3.2. Or my system could represent displacements with a magnitude colour map
   - 2.4. My code will need to produce a value of stress / potential energy for each element
   - 2.5. After the FEA, this part of my code should return a matrix of stresses and a vector of displacements ready for the optimization process to manipulate.
3. **Topology Optimisation** (redistributing the density after the FEA)
   - 3.1. The topology optimisation process should be able to receive the outputs from FEA
   - 3.2. My code should efficiently be able to execute the bisection algorithm to match the requirements that the boundary conditions specify.
   - 3.3. After each iteration, the code should manipulate the density ready for the next.
   - 3.4. The process should be able to recognise when the structure is optimised and stop.
4. **Binary Files** (Loading and saving presets)
   - 4.1. My system should make use of a binary file to load preset conditions for models
   - 4.2. There should be a group of buttons at the top of the screen for changing presets
     - 4.2.1. ‘Save’ which takes a name input and lets the user save the current preset
     - 4.2.2. ‘Load’ which opens a dropdown for the user to select a preset from a list
     - 4.2.3. ‘Clear Presets’ which deletes all of the user’s saved presets
5. **User Interface**
   - 5.1. The main section of the screen will be divided into four sections
     - 5.1.1. ‘Constraint Editor’
       - 5.1.1.1. The user should be able to add, move and remove constraints.
       - 5.1.1.2. The forces will be shown as arrows and the fixpoints as a fixpoint icon
       - 5.1.1.3. Icons should be moveable via drag and drop and via the arrow keys
     - 5.1.2. ‘Convergence Graph’
       - 5.1.2.1. The user will be able to see the stress improving after each iteration
       - 5.1.2.2. The stress should be shown as a graph plot on scalable axes
     - 5.1.3. ‘Optimised Topology’
       - 5.1.3.1. This window should contain a display of the optimised topology
       - 5.1.3.2. Display type will be interchangeable between a mesh and a texture
     - 5.1.4. ‘Displacement visualisation’
       - 5.1.4.1. There will be a slider showing the exaggeration of displacement
   - 5.2. There should be a vertical input section on the right of the screen
     - 5.2.1. There will be a box with inputs for changing the force of the selected icon
     - 5.2.2. My UI should include a slider for the volume fraction of the structure
     - 5.2.3. There should be a section controlling the visual outputs of the optimisation
   - 5.3. Post-Processing
     - 5.3.1. There should be a grid of toggles controlling the post-processing options
       - 5.3.1.1. One option will be whether the representation is a texture or a mesh
       - 5.3.1.2. There should also be a control for the type of colour representation

Design

Introduction

I will be designing the UI of my app before the more mathematical backend, as it allows me to have more time to research the algorithm to ensure the implementation is the best it can be. This is also because the algorithm requires user inputs, and it will be easier to control via the UI instead of through code.

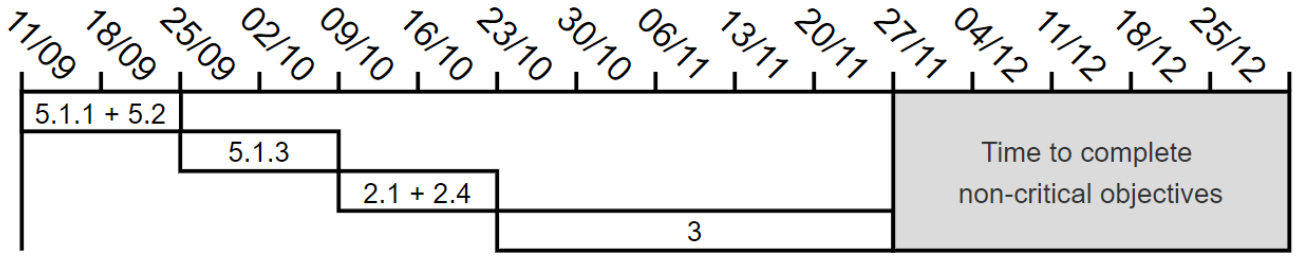
The following is a diagram representing the features or processes of my project and in which order they need to be completed (bottom to top). This helps me structure the design and development of my application by organising features visually.



Outlining a critical path

Below is the schedule I would like to keep to when programming my coursework. It lists the critical tasks that need to be completed in order for my program to function at its bare minimum.

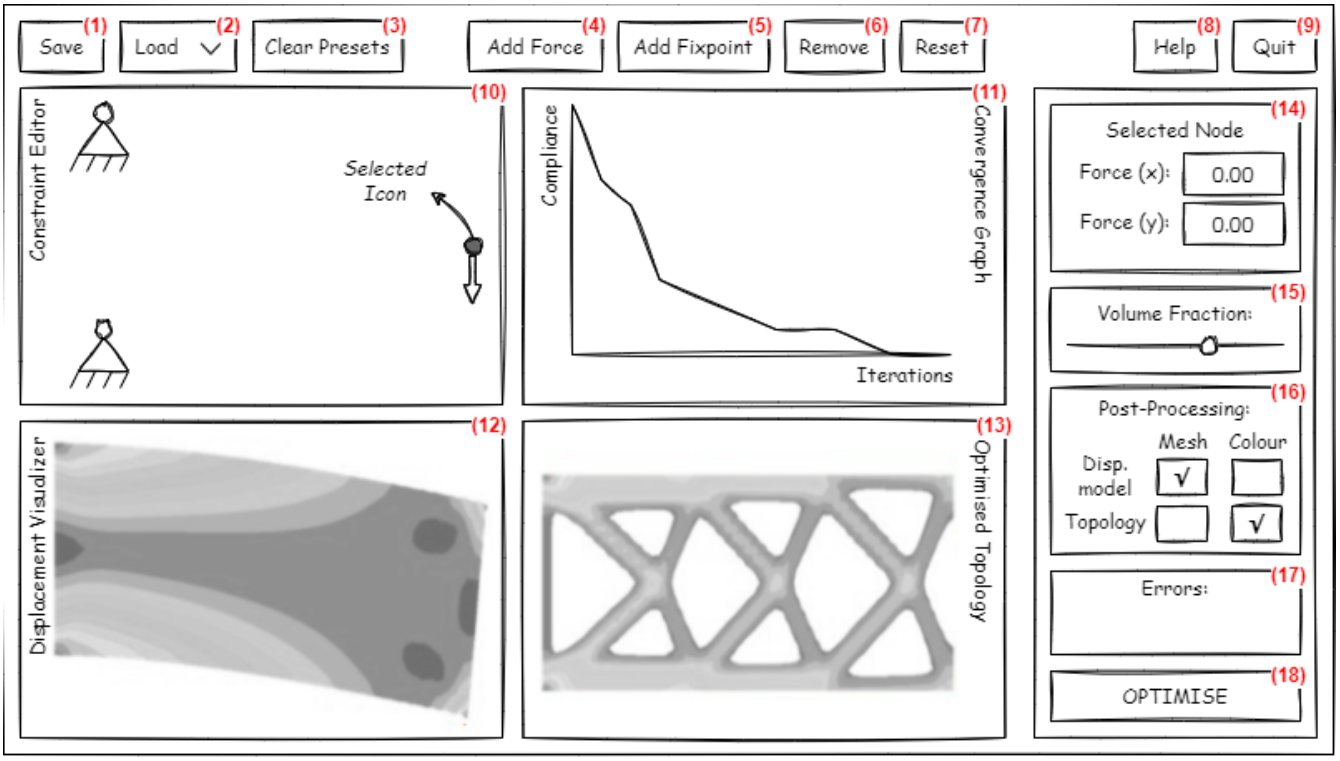
I have organised the tasks by their objective reference and placed them into a Gantt Cascade diagram so it's clear to see the tasks I need to complete. I have also left time at the end to implement the rest of my objectives, which aren't as important.



- 5.1.1 Build the inputs for the user interface, allowing creation and editing of constraints
- 5.2. Build a side panel for the other inputs needed for the algorithm
- 5.1.3. Create a display so the results can be seen when needed during the implementation
- 2.1. Make the user inputs accessible and in the right format for the FEA to work
- 2.4. Perform FEA successfully
- 3. Getting the optimization to work (I could break the design into different sections, but I wouldn't be able to schedule it as I don't know how long each task will take)

User Interface

Interface Design



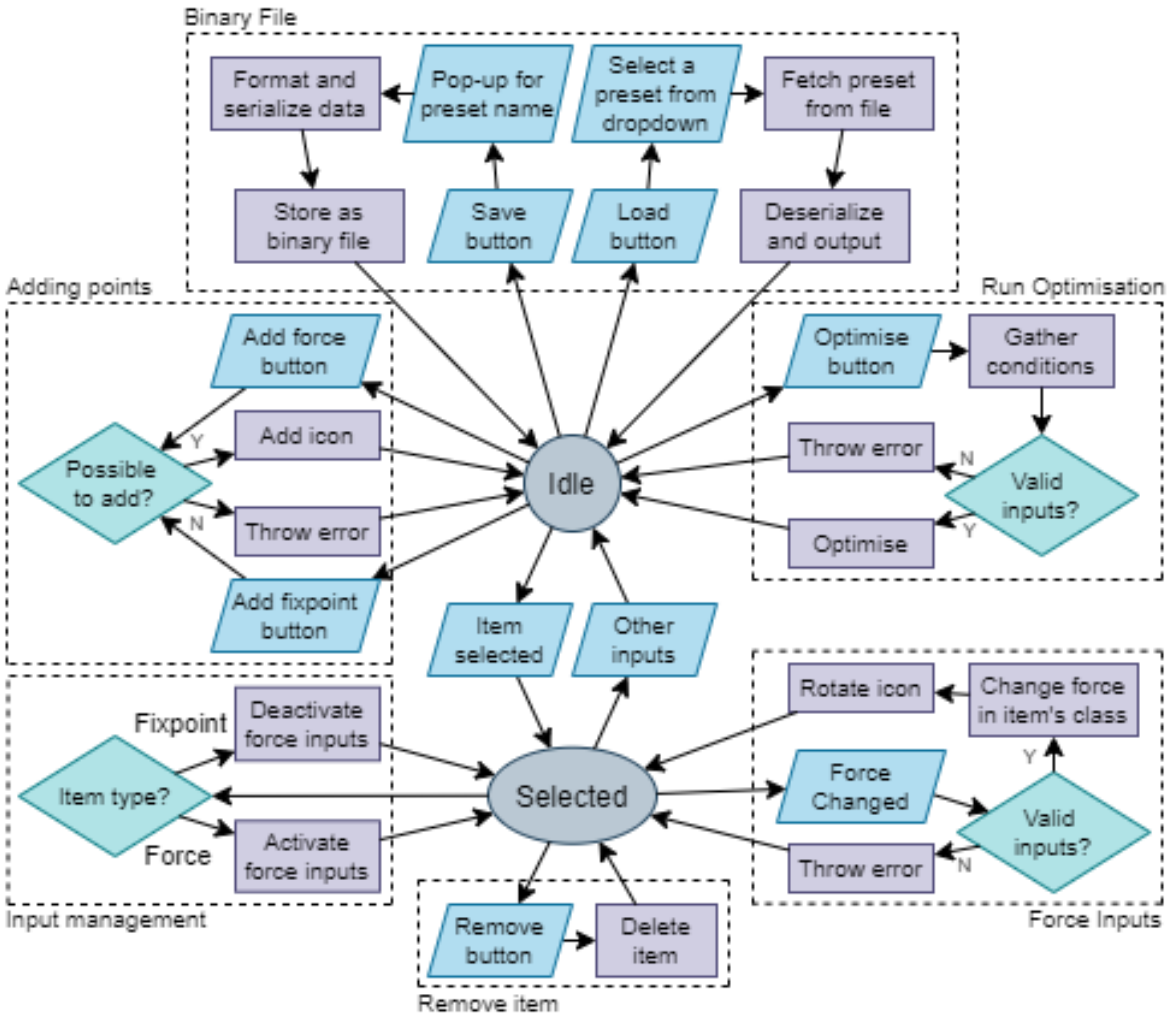
(1) saves the current preset
(2) lets the user select a preset
(3) delete all saved presets
(4) create a new fixpoint
(5) create a new fixpoint
(6) delete current icon
(7) clear editor
(8) open help menu
(9) close application
(10) edit constraints
(11) shows optimisation data
(12) shows displaced model
(13) shows optimisation result
(14) edit forces for current icon
(15) change volume fraction
(16) change how models look
(17) show any errors here
(18) start the process

Objectives Met

- ☑ The main section of the screen will be divided into four sections
- ☑ The user should be able to add, move and remove constraints.
- ☑ The forces will be shown as arrows and he fixpoints as a fixpoint icon
- ☑ Icons should be moveable via drag and drop and via the arrow keys
- ☑ The stress should be shown as a graph plot on scalable labelled axes
- ☑ One window should contain a display of the optimised topology
- ☑ Display type will be interchangeable between a mesh and a texture
- ☑ There should be a vertical input section on the right of the screen
- ☑ There will be a box with inputs for changing the force of the selected icon
- ☑ My UI should include a slider for the volume fraction of the structure
- ☑ There should be a section controlling the visual outputs of the optimisation
- ☑ There should be a grid of toggles controlling the post-processing options

UI Flowchart

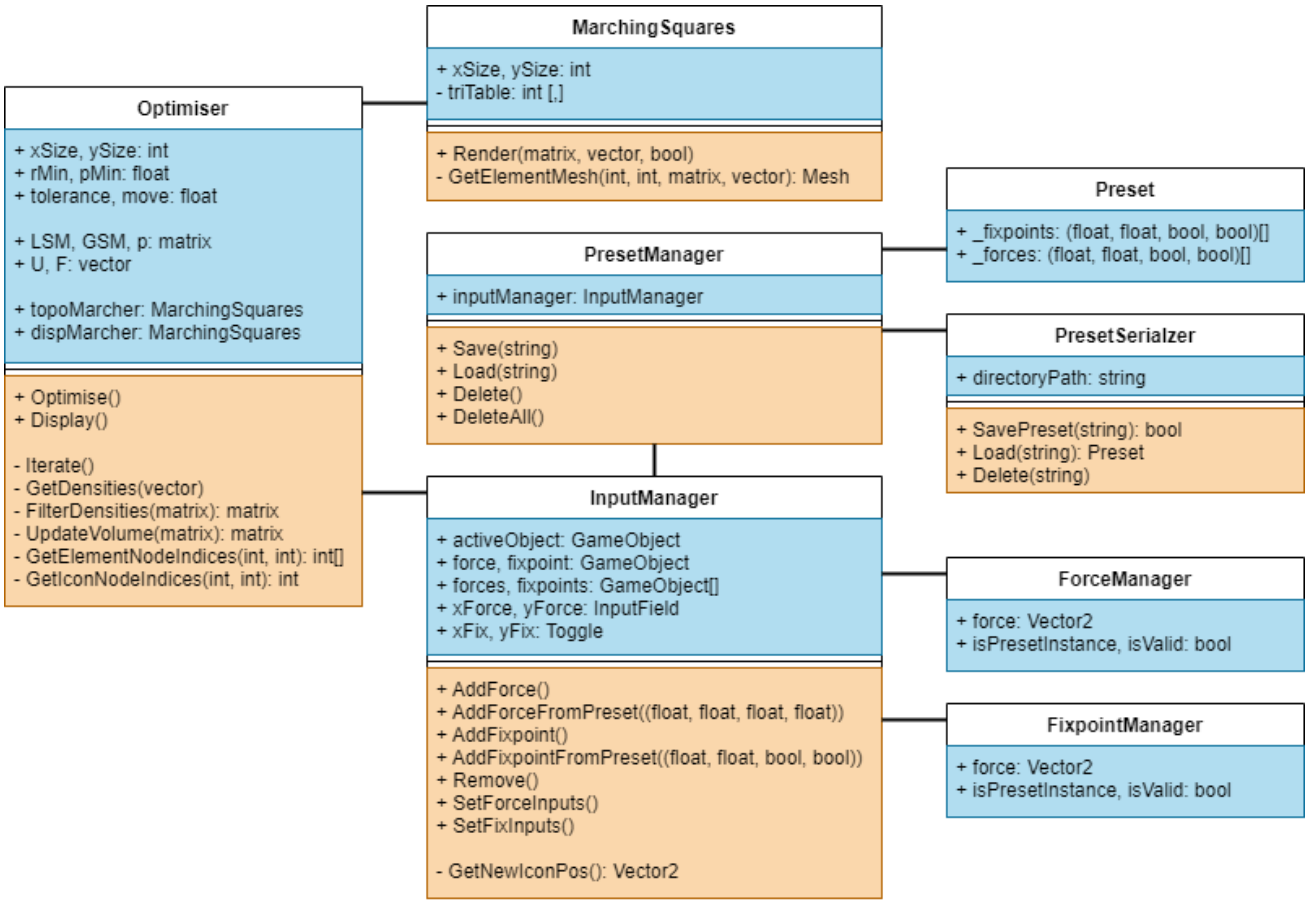
The diagram below is a combination of a finite state machine and a flowchart, due to the fact the program is in an idle state until an event takes place. When the interface is interacted with, the program leaves this state and follows a procedure represented by the flowcharts.



Input	Storage	Process	Output
Save Button	The preset in a new binary file	A user named file created and stored	None
Load Button	Fetch presets	Load preset into the app	Output selected preset
Add force/fixpoint	None	Add class and find offset position	Icon created in the model editor
Remove Button	None	Class and icon deleted	Icon deletes
Optimise Button	None	Gather all conditions and run the algorithm until the model is converged	Visual representation of each iteration in the model and the convergence graph

Class Diagram

The following classes are the main classes I designed for the application.



Optimiser runs the calculations needed to optimise the structure and returns a density matrix
MarchingSquares takes a matrix from the optimiser and renders a displaced/regular mesh
InputHandler manages the constraint editor and the forces and fixpoints within it
PresetManager saves and loads presets into the application using the PresetSerializer
PresetSerializer converts the preset class int a binary file and stores it in the right directory
Preset class is all the data needed to store recreate a structure from a binary file
ForceManager holds the data about each force constraint and validates inputs at runtime
FixpointManager holds the data about each fixpoint and validates the inputs

Attribute Dictionary

Optimiser

Name	Validation	Description
xSize	integer	Holds the number of elements in the x dimension
ySize	integer	Holds the number of elements in the y dimension
rMin	> 1	The radius that the density filter uses to blur the densities
pMin	0 to 1	The minimum density the elements can have
tolerance	0 to 1	The error tolerance of the bisection algorithm
move	0-1	The maximum an element can change in one iteration
p	matrix (0-1)	The matrix holding the density value for every element
LSM	matrix	The local stiffness matrix (same for every element)
GSM	matrix	The global stiffness matrix for the structure
U	vector	The displacement of every degree of freedom
F	vector	The force vector containing the user input forces
topoMarcher	MarchingSquares	Holds the script that renders the undisplaced mesh
dispMarcher	MarchingSquares	Holds the script for rendering the displaced mesh

InputManager

Name	Validation	Description
activeObject	fixpoint or force	Holds the 'selected' constraint so the user can edit it
force	GameObject	The prefab used to instantiate a force object into the app
fixpoint	GameObject	The fixpoint prefab used when adding fixpoints
forces	List of force objects	Holds all the fixpoint objects in the editor
fixpoints	List of fixpoints	Holds all of the force objects in the editor
xForce	InputField	Entry box used to change the value of the horizontal force
yForce	InputField	Input for changing the vertical component of the force

Marching Squares

Name	Validation	Description
xSize	Optimiser xSize - 1	Holds the number of element meshes in the x dimension
ySize	Optimiser ySize - 1	Holds the number of element meshes in the y dimension
triTable	Int list from -1 to 7	Indices of triangles to be drawn, -1 marks end of data

PresetManager, Preset and PresetSerializer

Name	Validation	Description
_forces	Non-zero forces	Holds forces as four floats: pos x, pos y, force x, and force y
_fixpoints	x or y must be fixed	Holds fixpoints as 2 position floats and 2 bools if its a roller
inputManager	InputManager	References the input manager so it can instantiate presets
directoryPath	Valid file path	Holds the path to where the presets are stored

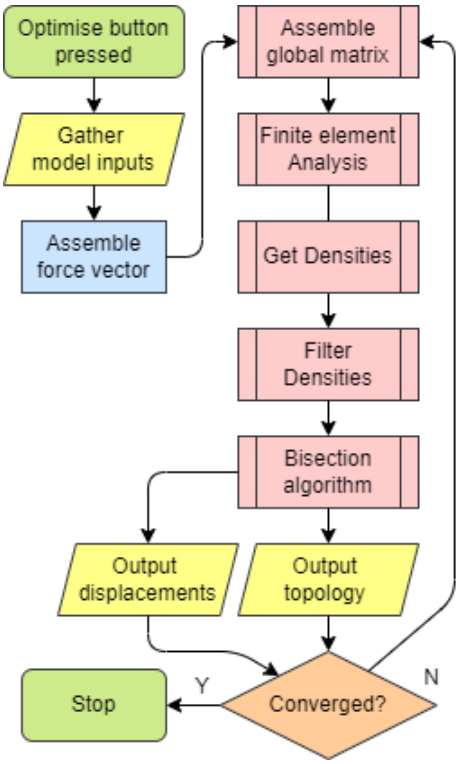
Algorithm Design

Topology Optimisation

Optimise

This section includes flowcharts and pseudocode representing the order in which the algorithm will be executed in the background of my application. It's broken off into subroutines to make it easier to understand.

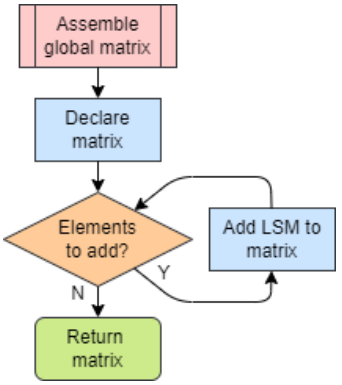
```
PROCEDURE Optimise
	F, U = new Vector
	K, p, pNew = new Matrix

	// assemble force vector
	FOREACH force IN forces
		// 'GetIndex' returns index of force
		F[GetIndex(force)] = force.x
		F[GetIndex(force) + 1] = force.y
	ENDFOREACH

	// assemble GSM
	K = GetGlobalMatrix()

	// finite element analysis
	U = GetDisplacementVector(K, F)
	p = GetDensities(U)

	// filter densities using buffer 'pNew'
	pNew = FilterDensities(p)

	// bisection algorithm
	p = CriteriaUpdate(pNew)

	Display(p)
ENDPROCEDURE
```



Assemble Global Matrix (GetGlobalMatrix)

This function adds every element's LSM into the global matrix. I have chosen to comment out the main code because it's hard to understand all the indices in the code and what they represent

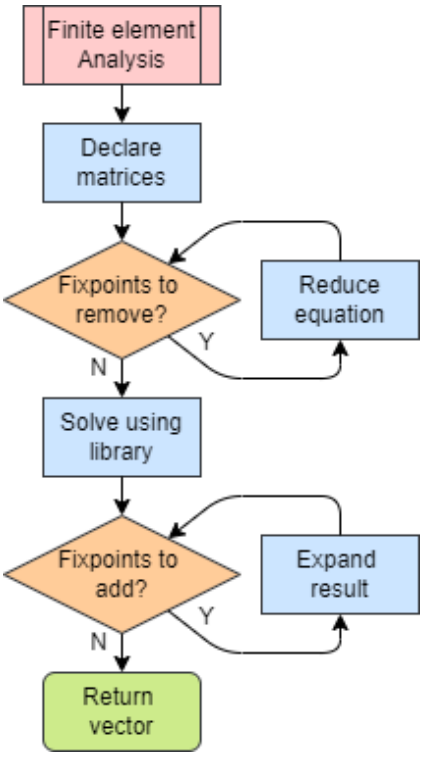
```
FUNCTION GetGlobalMatrix
	// declare matrix
	K = new Matrix

	FOREACH element IN elements
		// ...
		// add each LSM to 'K'
		// ...
	ENDFOREACH

	RETURN K
ENDFUNCTION
```



Finite Element Analysis (GetDisplacementVector)

Reduces the system of linear equations to incorporate the fixpoints. Then it solves the equation and expands the vector result, placing a zero at every index where there is a fixpoint.

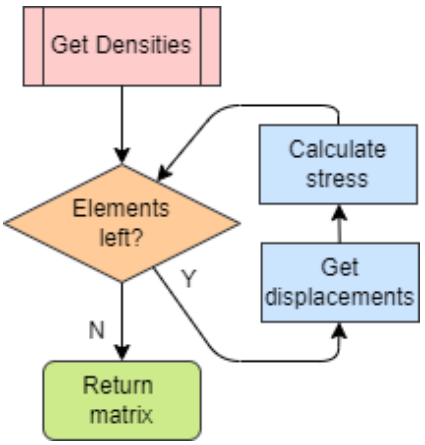
```
FUNCTION GetDisplacementVector(K, F)
      U = new Vector

      // reduce equation using fixpoints
      FOREACH fixpoint IN fixpoints
            // 'GetIndex' returns index of fixpoint
            K.Remove(GetIndex(fixpoint))
            K.Remove(GetIndex(fixpoint) + 1)

            F.Remove(GetIndex(fixpoint))
            F.Remove(GetIndex(fixpoint) + 1)
      ENDFOREACH

      // linear algebra solver from 'Math.NET'
      U = K.Solve(F)

      // expand equation using fixpoints
      FOREACH fixpoint IN fixpoints
            // fill fixpoints with 0 displacement
            K.Add(GetIndex(fixpoint), 0)
            K.Add(GetIndex(fixpoint) + 1, 0)
      ENDFOREACH
      RETURN U
ENDFUNCTION
```



Finding Stress (GetDensities)

Finds the stress in each element and returns it as a matrix of values

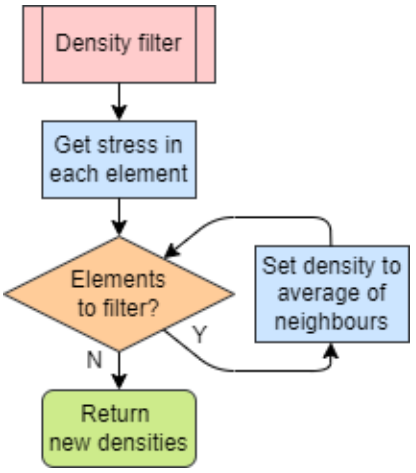
```
FUNCTION GetDensities(U)
      p = new Matrix
      FOREACH value IN p
            // 'GetIndicesOf' gets displacement indices
            // and returns a vector
            n = GetIndicesOf(value)

            // gets stress using local stiffness matrix
            value = n.Transpose * LSM * n
      ENDFOREACH
      RETURN p
ENDFUNCTION
```



The Density Filter (FilterDensities)

Basically, the density filter blurs the structure so any irregularities get reduced and the structure has the ability to make new connections within itself

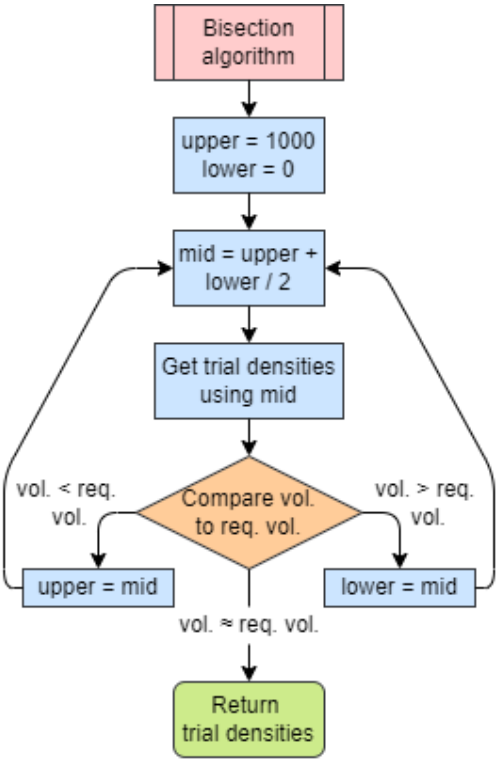
```
FUNCTION FilterDensities(p)
      rmin = 2 // filter radius
      FOR j = 0 to ySize
            FOR i = 0 to xSize
                  adj = new List // adjacent elements
                  FOR v = -rmin to rmin
                        FOR u = -rmin to rmin
                              adj.Add(p[i+u, j+v])
                        ENDFOR
                  ENDFOR
                  pNew[i, j] = adj.Average()
            ENDFOR
      ENDFOR
      RETURN pNew
ENDFUNCTION
```



Bisection Algorithm (CriteriaUpdate)

Finds the correct multiplier to use when updating the density so the structure matches the user's required volume. This function is effectively a binary search to find that multiplier.

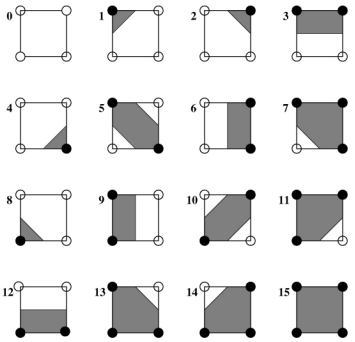
```
PROCEDURE CriteriaUpdate(p)
      // declare variables
      lower = 0, upper = 10000

      // get user input for 'reqVol'
      reqVol = GetVolume(), vol = Infinity
      tolerance = 0.1

      WHILE (vol - reqVol).Abs() > tolerance
            mid = (upper + lower) / 2
            FOREACH value IN p
                  // update value using mid
                  value = // new density
            ENDFOREACH

            vol = p.Sum / (xSize * ySize)
            IF vol > reqVol
                  lower = mid
            ELSE
                  upper = mid
            ENDIF
      ENDWHILE
PROCEDURE
```



Marching Squares

To represent the structures generated, there will be two views for the user: a texture/grid view where an element's density will be represented by the colour of the pixel, or a mesh view. To have a mesh view, I need to use a method to smooth the mesh to an unpixelated shape, this algorithm is called marching squares.

Marching squares is an algorithm for smoothing a grid of values into a mesh. Provided you have a grid of points and you know which points are inside or outside the shape, the algorithm generates a mesh for that shape.



As seen in fig(4.0), these 16 combinations cover all possible cases. To reference each case, we can treat each vertex as a bit: 1 for inside, 0 for outside. This way we can use this 4-bit number to reference a table and find the correct mesh to draw. Using case 9 as an example, the binary value would be 1001, which has the denary value of 9. So now the algorithm would look at the 9th index in a mesh lookup table and that will determine which midpoints and vertices need to be connected.

However, this will not produce an entirely smooth mesh since the mesh is drawn from the midpoints and the vertices.

Pseudocode

```
FUNCTION GenerateMesh(p)
	Mesh mesh = new Mesh()
	FOR j = 0 to ySize
		FOR i = 0 to xSize
			arr = [false, false, false, false]
			// get values of the corners of the square
			FOR v = 0 to 1
				FOR u = 0 to 1
					IF p[i + u, j + v] > 0.5f
						arr[i + 2 * j] = true; // inside
					ELSE
						arr[i + 2 * j] = false; // outside
					ENDIF
				ENDFOR
			ENDFOR
			Mesh elementMesh = new Mesh;
			elementMesh.vertices = relativePoints;
			elementMesh.triangles = triTable[arr.ToDec];
			mesh = Combine(mesh, elementMesh)
		ENDFOR
	ENDFOR
	RETURN mesh
ENDFUNCTION
```

Presets

The following pseudocode class stores all the information needed to recreate a preset, and it stores it in a serializable format, meaning it can be converted to binary and back:

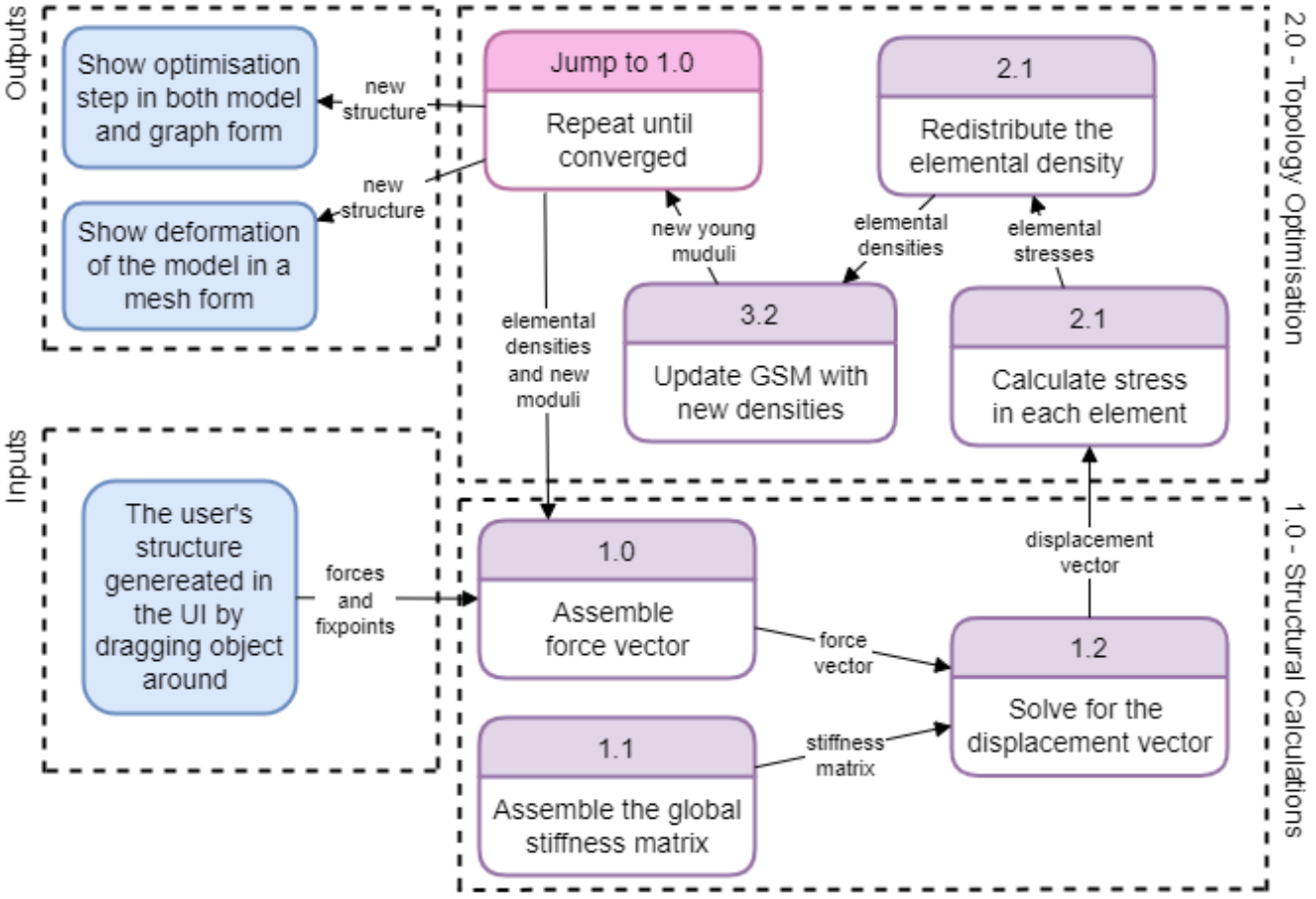
```
CLASS PresetClass
    // the class which holds all required data for a preset
    public serializable (float, float, bool, bool)[] _fixpoints
    public serializable (float, float, float, float)[] _forces

    public PROCEDURE new // constructor for PresetClass
        _fixpoints = new (float, float, bool, bool)[]
        _forces = new (float, float, float, float)[]

        FOREACH GameObject IN forces
            // get data and add to list
        ENDFOREACH

        FOREACH GameObject IN fixpoints
            // get data and add to list
        ENDFOREACH
    ENDPROCEDURE
ENDCLASS
```

The PresetSerializer reads and writes the preset as requested in runtime

```
CLASS PresetSerializer
    public static string directoryPath, filePath

    public static FUNCTION Save(string name) // true if save was successful
        filePath = directoryPath + "\" + name

        // ensures there's always a directory to store presets
        IF directoryPath == null
            new Directory(directoryPath)
        ENDIF

        // if file doesn't already exist, create preset and save
        IF filePath == null
            PresetClass currentPreset = new PresetClass()
            File file = new File(name)
            file.OpenWrite(Serialize(currentPreset))
            file.Close()
            RETURN true
        ELSE
            RETURN false
        ENDIF
    ENDFUNCTION

    public static FUNCTION Load(string name)
    // returns the class stored in the file
        PresetClass data = Deserialize(File.OpenRead(GetFilePath(name)))
        file.Close()
        RETURN data
    ENDFUNCTION
ENDCLASS
```

Data Flow Diagram

This is a logical level 1 data flow diagram showing how data runs through my system. I added an unconventional feature ("Jump to 1.0") to my diagram which is essentially representing a while loop in my system. If the concept isn't clear from this diagram, see the flowchart on the next page which explores the algorithm in higher detail.



Data Structures

In my project I'm going to include two data structures which were imported from math.NET: **Matrices** and **Vectors**. Vectors contain an array of values, and a matrix contains a 2D grid of values, both supporting single and double precision, real and complex floating-point numbers.

Both data structures have two types of storage layout:

Dense which uses a single array to store every value, this is a standard storage technique and allows for fast access. However, for matrices with not that many non zero values, memory space is wasted

Sparse which uses two arrays to store every value that is non zero and its corresponding index. For matrices, CSR compression format is used. Sparse layouts are effective with storage, however, the access time is significantly longer.

There are many built-in methods in this library that I will use throughout my code, such as 'solve', which takes in a matrix and a vector, and returns the result from the linear system of equations. As seen on the right, input A and B, it will return x. This is used in the FEA process for the GSM and the force vector. There are many other methods however I will cover them in the implementation stage.

$$\begin{bmatrix} 1 & 3 & 1 \\ 2 & 2 & 2 \\ 3 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 5 \end{bmatrix}$$

A x b

Implementation

NOTE: UI screenshots have been colour corrected for printing, and the tests ran throughout show only the first iteration to save time

Development Plan

As per my design section, I will complete the following objectives in this order:

Critical Objectives

5.1.1. Build the inputs, allowing creation and editing of constraints
5.2. Build the inputs, allowing the user to change numerical data
5.1.3. Create a display for the results
2.1. Make the user inputs accessible and in the right format for the FEA
2.4. Perform FEA successfully
3. Getting the optimization to work

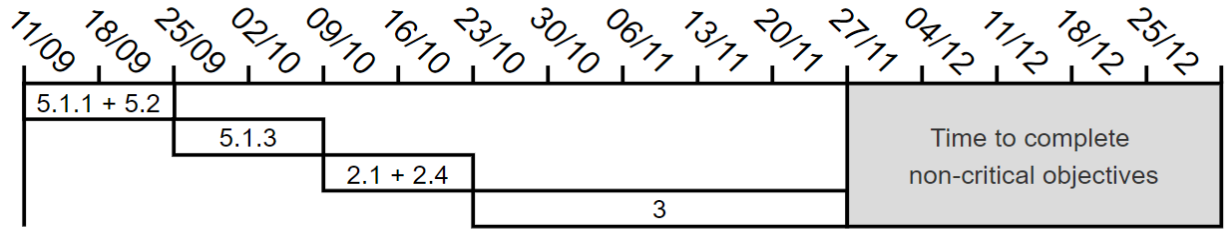
Non-Critical Objectives

4. Develop preset saving / loading
5.1.4. Code marching squares algorithm for mesh generation and displacement

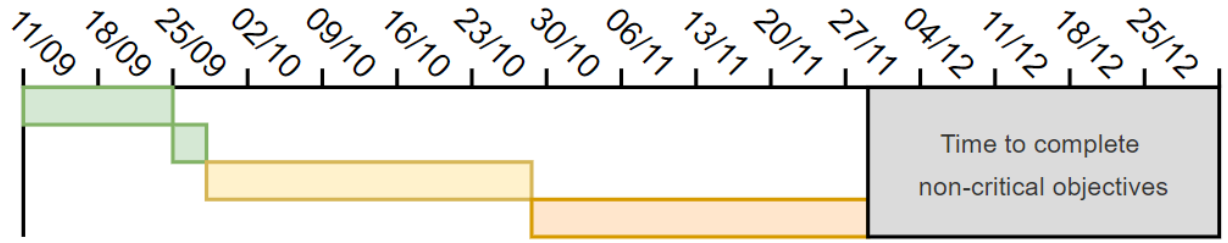
Progress Log

Obj ref:	**Date Completed:**	**Test Results:**
5.1.1 - Constraint editor	17 / 09 - on track	Successful
5.2 - Other Inputs	25 / 09 - on track	Successful
5.1.3 - Input Formatting	28 / 09 - on track	Successful
2.1 - Outputting Texture	03 / 10 - ahead of schedule	Successful
2.4 - Performing FEA	29 / 10 - behind schedule	Successful
3 - Performing Optimisation	28 / 11 - behind schedule	Successful
4 - Implementing Presets	13 / 12 - on track	Successful
5.1.4 - Displacement Visualiser	24 / 12 - on track	Successful

Expected Progres



Actual Progress



Building the Constraint Editor — Obj Ref 5.1.1

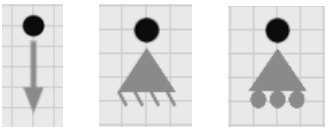
This module has a UI focus, and I will be implementing the following objectives

- The user should be able to add, move and remove constraints
- The forces will be shown as arrows and the fixpoints as a fixpoint icon
- Icons should be moveable via drag and drop and via the arrow keys

Setting up the Editor

Firstly I made a box containing all of the constraints, and the grid they will be placed on. This means the user can see exactly where their points are going in case they want to align or centre anything. This also helps the user gauge some alignment with the end structure so they can see what effect their changes have on the result.

I then decided to create my constraint icons (force, fixpoint and roller) to be placed on the grid:

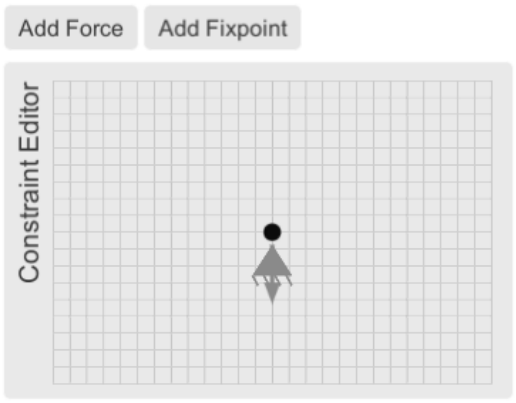


Adding Constraints

Next, the user needed to be able to add these fixpoints onto the grid. I made an 'InputHandler' class which will hold all the input features, however for now I will make an 'Add Force' and an 'Add Fixpoint' button, so I can add constraints.

```
// holds the list of the constraint objects
[HideInInspector] public static List<GameObject> forces = new List<GameObject>();
[HideInInspector] public static List<GameObject> fixpoints = new List<GameObject>();

// holds the presets of the constraints
public GameObject force;
public GameObject fixpoint;

// adds default force when 'add force' is pressed
public void AddForce()
{
    // puts a limit on how many forces there are
    if (forces.Count <= 5)
    {
        GameObject n = Instantiate(force, Vector3.zero, Quaternion.Euler(0, 0, -90)); // instantiate
        n.transform.SetParent(transform, false); // set as a child of this object
        forces.Add(n); // add object to the force list
    }
    else { ErrorText.ShowError("Maximum number of forces reached"); }
}

// adds default fixpoint when 'add fixpoint' is pressed
public void AddFixpoint()
{
    // puts a limit on how many fixpoints there are
    if (fixpoints.Count <= 20)
    {
        GameObject n = Instantiate(fixpoint, Vector3.zero, Quaternion.Euler(0, 0, 90)); // instantiate
        n.transform.SetParent(transform, false); // set as a child of this object
        fixpoints.Add(n); // add object to the fixpoint list
    }
    else { ErrorText.ShowError("Maximum number of forces reached"); }
}
```

Offsetting the icons

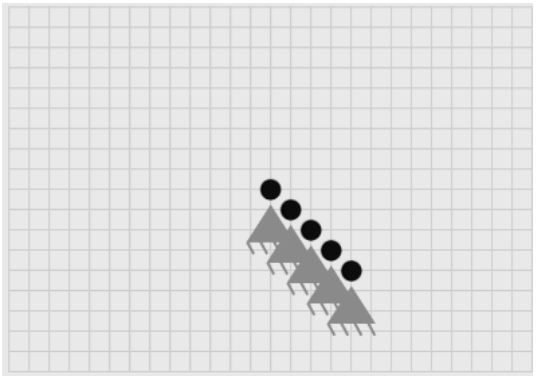


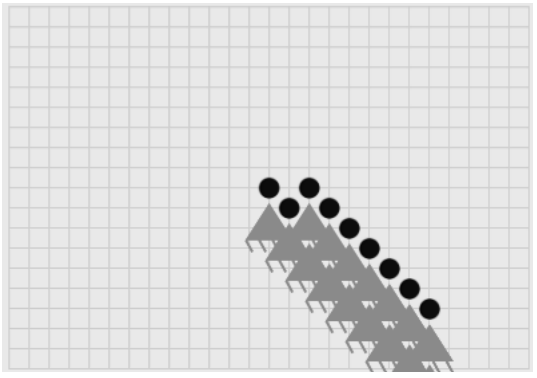
I decided to add these functions like buttons and test if they worked or not, and they did. I was successfully able to add fixpoints and forces, however, they were overlapping.

I didn't like how this looked, because you couldn't tell how many icons were added. Hence, I decided to add a function which offsets the icons until an empty point is found. This would mean the user can see every new constraint they have added and don't have to move other constraints out of the way to access their new one

```
// adds icons with an offset
public Vector2 GetNewIconPos()
{
    bool found;
    // starts new diagonal line if all the points are taken
    for (int j = 0; j <= 4; j += 2)
    {
        // offsets down and right until outside of the box
        for (Vector2 i = new Vector2(13 + j, 9); i.y > 0; i += new Vector2(1, -1))
        {
            found = true;

            // check if point is taken
            foreach (GameObject n in forces)
            {
                if (n.GetComponent<RectTransform>().anchoredPosition == i) { found = false; }
            }
            foreach (GameObject n in fixpoints)
            {
                if (n.GetComponent<RectTransform>().anchoredPosition == i) { found = false; }
            }
            if (found) { return i; }
        }
    }
    return Vector2.zero;
}
```

I tested the adding buttons with this code and the results were successful:





Drag and Drop

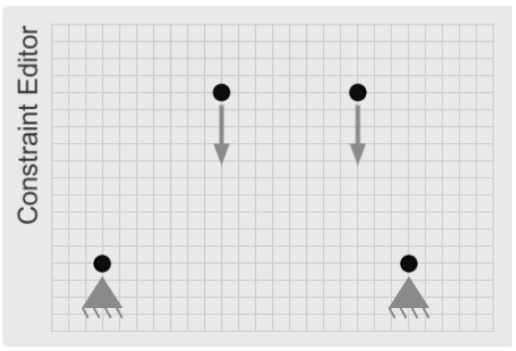
With the above completed, I needed to write some code which allows the user to drag and drop the constraint on a point within the grid. Unity has a UI class called 'IDragHandler' which manages what happens when a UI object is clicked and dragged. I created a class which inherits methods from IDragHandler called 'Draggable' and this makes every object it's attached to moveable.

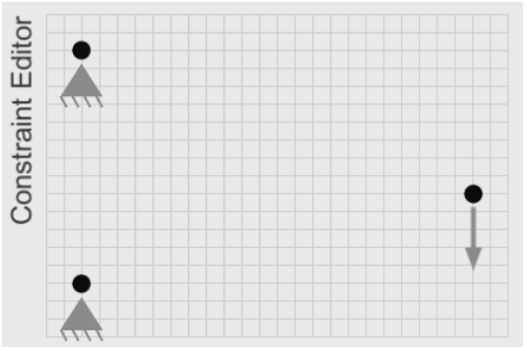
```
public class Draggable : MonoBehaviour, IDragHandler
{
    public RectTransform environment; // are to drag inside
    public RectTransform rt; // transform of the attached object

    public void OnDrag(PointerEventData eventData)
    {
        // adds change in moues to the position
        rt.anchoredPosition += eventData.delta;

        // if outside of area, undo the change
        if (!environment.rect.Contains(rt.anchoredPosition)) { rt.anchoredPosition -= eventData.delta; }
    }

    public void OnEndDrag(PointerEventData eventData)
    {
        // snaps to grid
        rt.anchoredPosition = new Vector2(Mathf.Round(rt.anchoredPosition.x), Mathf.Round(rt.anchoredPosition.y));
    }
```

Also, I wanted the user to be able to move the constraints with keyboard shortcuts as well because if their mouse can't get the constraint to snap to the right point, they can just use the arrow keys, and this stops the user from getting frustrated. (Update is called every frame)

```
    void Update()
    {
        int x = 0, y = 0;
        if (Input.GetKeyDown("right")) { x++; }
        if (Input.GetKeyDown("left")) { x--; }
        if (Input.GetKeyDown("up")) { y++; }
        if (Input.GetKeyDown("down")) { y--; }
        if (Input.GetAxis("shift") != 0) { x *= 5; y *= 5; }

        // if outside the area, undo the change
        transform.parent.localPosition += new Vector3(x, y, 0);
        if (!environment.rect.Contains(transform.parent.localPosition))
        {
            transform.parent.localPosition -= new Vector3(x, y, 0);
        }
    }
```

I tested this script and was able to drag the constraints around and not go outside the grid. I also tested my arrow key feature and it worked as expected.





Removing Constraints

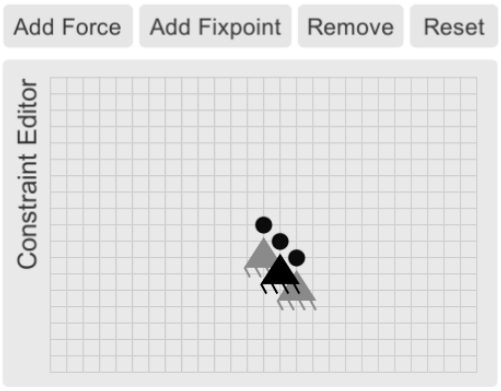
Then decided to add a button which removes constraints. I needed a way for the user to select which icon they wanted to remove, so I decided to make an attribute called 'activeObject' in the input handler which held the icon the user selected. The user can select an icon by clicking it or dragging it. I needed to inherit from 'IPointerClickHandler' so I could detect a click.

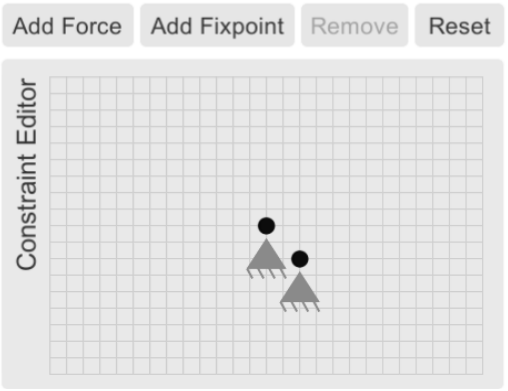
```
public class Draggable : MonoBehaviour, IDragHandler, IPointerClickHandler
{
    public RectTransform environment;
    public RectTransform rt;

    public void OnDrag(PointerEventData eventData)...

    public void OnEndDrag(PointerEventData eventData)...

    public void OnPointerClick(PointerEventData eventData)
    {
        if (eventData.button == PointerEventData.InputButton.Left)
        {
            InputHandler2D.activeObject = this.transform.parent.gameObject;
        }
    }
```

The following function 'Remove' removes the selected icon and 'Reset' removes all constraints.

```
// removes active constraint from domain
public void Remove()
{
    fixpoints.Remove(activeObject);
    forces.Remove(activeObject);
    Destroy(activeObject);
}

// removes all constraints from domain
public void Reset()
{
    foreach (GameObject n in forces) { Destroy(n); }
    foreach (GameObject n in fixpoints) { Destroy(n); }
    forces = new List<GameObject>();
    fixpoints = new List<GameObject>();
}
```

I set the colour of an object to change if it was selected, so the user can see which one of their icons they are going to remove. As seen below the selected icon can be removed:



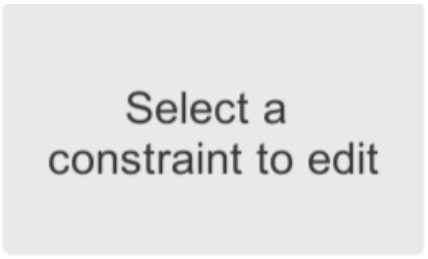


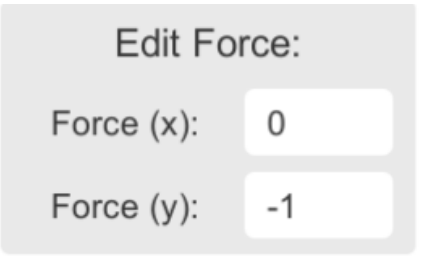
Adding Other Inputs — Obj Ref 5.2

The goal of this section is to:
- Allow the user to edit their constraints, including fixpoint axes and force vectors
- Make the icons visually adapt to the user's changes
- Let the user change the volume slider value and the displacement value
- Set up post-processing options for later on

Editing Constraints

I will have a 'Selected Icon' box which changes depending on whether the selected icon is null, a force or a fixpoint. I created these three states in my UI:

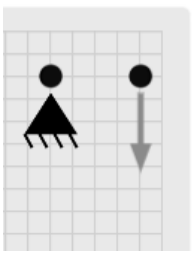


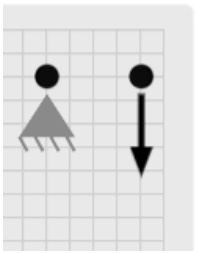


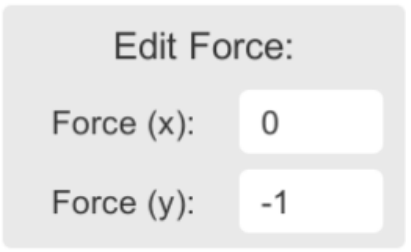
The following code switches between these three panels:
'fixOption' and 'forceOption' hold the two panels above, and they are layered on top when needed

```
public class InputHandler2D : MonoBehaviour
{
    Attributes


    void Update()
    {
        if (activeObject != null)
        {
            if (forces.Contains(activeObject))
            {
                forceOption.SetActive(true); fixOption.SetActive(false);
            }
            else if (fixpoints.Contains(activeObject))
            {
                fixOption.SetActive(true); forceOption.SetActive(false);
            }
        }
        else { fixOption.SetActive(false); forceOption.SetActive(false); }
    }
```

I did a test to check the panel changed when a new point was selected, as seen it was successful:







Fetching Data from the Panels

In the Input Manager, I made two functions for fetching the data from the inputs, alongside a 'FixpointManager' and a 'ForceManager' class to attach to each type of icon. This is where the constraint information will be stored and validated.

Here are the two functions which are called whenever a value is changed:

```
// called when force values change in UI
public void SetForceInputs()
{
    float x = float.Parse(xForce.text);
    float y = float.Parse(yForce.text);
    activeObject.GetComponentInChildren<ForceManager>().force = new Vector2(x, y);
}


// called when fixpoint toggles change in UI
public void SetFixInputs()
{
    activeObject.GetComponentInChildren<FixpointManager>().x = xFix.isOn;
    activeObject.GetComponentInChildren<FixpointManager>().y = yFix.isOn;
}
```

This is the force manager. It throws an error and switches its validity status to false if the force has no magnitude, the inputs on forces are already validated as numbers via unity.

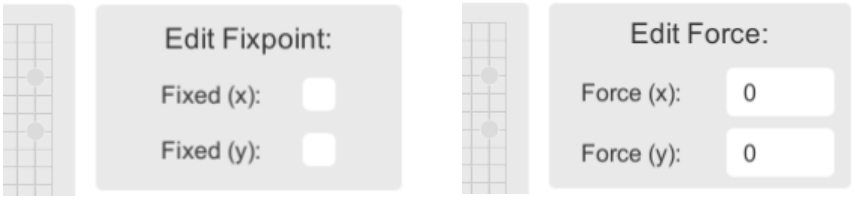
In addition to changing the values inside the code, I added a rotation to the force arrow so the user can see which direction their force points in.

```
public class ForceManager : MonoBehaviour
{
    public Vector2 force;
    public bool isValid;
    public Image arrow;

    public void Update()
    {
        if (force == Vector2.zero)
        {
            isValid = false;
            ErrorText.ShowError("Magnitude of force cannot be zero");
        }
        else
        {
            isValid = true;
            float angle = Mathf.Atan(force.y / force.x) * Mathf.Rad2Deg;
            if (force.x < 0) { angle += 180; }
            transform.parent.transform.rotation = Quaternion.Euler(0, 0, angle);
        }

        arrow.enabled = isValid;
    }
}
```

This code below is the fixpoint manager, which stores two bools which determine whether the fixed direction is in the x-direction or the y-direction. The icon remains as a fixpoint icon if the directions are both fixed, an upright roller if the y is fixed, and a sideways roller if the x is fixed.

```
public class FixpointManager : MonoBehaviour
{
    public bool x, y;
    public bool isValid;
    public Image roller, fix;

    public void Update()
    {
        if (!x && !y)
        {
            ErrorText.ShowError("Fixpoints must have at least one dimension fixed");
            roller.GetComponent<Image>().enabled = false;
            fix.GetComponent<Image>().enabled = false;
            isValid = false;
        }
        else
        {
            isValid = true;
            if (x && y)
            {
                transform.parent.transform.rotation = Quaternion.Euler(0, 0, 90);
                roller.GetComponent<Image>().enabled = false;
                fix.GetComponent<Image>().enabled = true;
            }
            else if (x)
            {
                transform.parent.transform.rotation = Quaternion.Euler(0, 0, 0);
                roller.GetComponent<Image>().enabled = true;
                fix.GetComponent<Image>().enabled = false;
            }
            else if (y)
            {
                transform.parent.transform.rotation = Quaternion.Euler(0, 0, 90);
                roller.GetComponent<Image>().enabled = true;
                fix.GetComponent<Image>().enabled = false;
            }
        }
    }
}
```

In addition to the colour changes when selected, I added a colour change to red if the constraint is erroneous, so the user knows which icon the error message is talking about. Also, if an icon is invalid, it will all be disabled apart from the draggable circle. This code is now in the constraint managers, and the test below shows the colour change and disabling works.

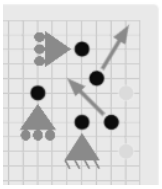


The icons are barely visible unfortunately due to the brightness of the colour red I am using, but as seen the icons still change to just a circle when the data is invalid.

I added three forces to my grid and four fixpoints with varying inputs to see if the code works. The results follow:

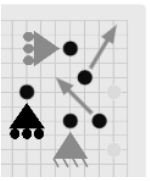
Test:
Null fixpoint

Result:
Success



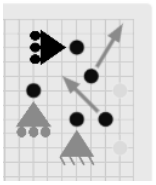
Test:
Roller (y)

Result:
Success



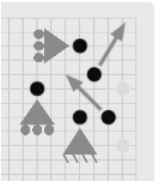
Test:
Roller (x)

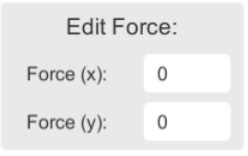
Result:
Success



Test:
Null force

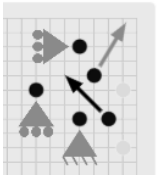
Result:
Success

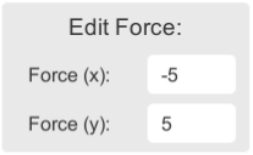




Test:
Random force

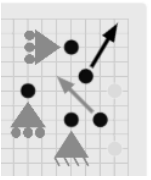
Result:
Success

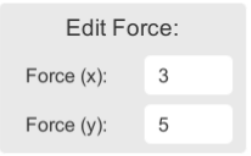




Test:
Random force

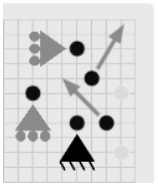
Result:
Success





Test:
Fixpoint

Result:
Success

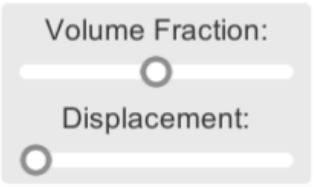


That is everything I need to do with the icons for now, everything is tested and working, therefore, I can easily test my code using my UI when I need to and I don't need to hard code the inputs.

Adding Sliders

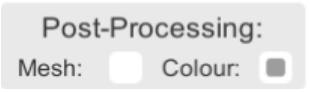
Unity has built-in slider widgets, which makes adding the volume and displacement slider a matter of just implementing them in my UI.

These sliders have easily accessible values and the minimum value and maximum value are easily changeable with unity.



Post Processing

Similar to the sliders, unity has toggles which I put in a panel and assigned to change whether the resulting structure is rendered as a mesh or texture, and whether it's coloured or not.



Creating an Output — Obj Ref 5.1.3

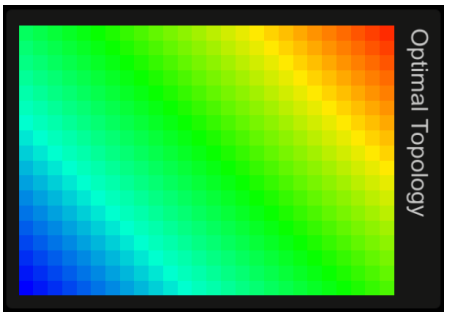
For now, so I can see my results, I am going to create a texture which displays values from 0-1 as a colour from a gradient which follows a gradient from blue to red.

The following code generates a texture, which lets me access each pixel and change the colour

```
// output model defined by post processing toggles
public void Display()
{
    // initialize texture
    topoTexture.enabled = true;
    Texture2D texture = new Texture2D(xSize, ySize);
    topoTexture.texture = texture;

    // set each pixel using densiy matrix
    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            // access colour from gradient
            texture.SetPixel(i, j, gradient.Evaluate((float)p[i, j]));
        }
    }
    texture.Apply();
}
```

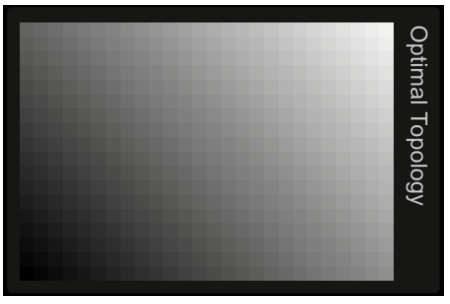
To test this is working, I added an image to the UI and applied the texture. As a test function, I set every pixel value to (i + j) / (xSize + ySize), and this should produce a diagonal gradient from the bottom left corner. Below is the code and the result:

```
public void Display()
{
    topoTexture.enabled = true;
    Texture2D texture = new Texture2D(xSize, ySize);
    topoTexture.texture = texture;
    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            texture.SetPixel(i, j,
                gradient.Evaluate((float)(i + j) / (float)(xSize + ySize)));
        }
    }
    texture.Apply();
}
```



As per my objectives, I will use the colour toggle in the post processing panel to give the user the option of a black and white texture. The code and example result of this feature follows:

```
if (topoColor.isOn)
{
    texture.SetPixel(i, j, gradient.Evaluate((float)p[i, j]));
}
else
{
    texture.SetPixel(i, j,
        new Color((float)p[i, j], (float)p[i, j], (float)p[i, j], 1f));
}
```



Formatting the Inputs — Obj Ref 2.1

The purpose of the code in these functions is to translate the user inputs to useable values

Index Functions

As this problem is grid-based, accessing indices for degrees of freedom is difficult, therefore I built two functions which calculate which indices to access.

The first function gets the index of the **x DOF** given the **position of the node** in the grid. The y DOF can be found by adding 1 to the result of this calculation.

```
// returns the DOF index of a node
int ndof(int x, int y)
{
    return 2 * ((ySize + 1) * x + ySize - y);
}
```

The second function returns a **list of all DOFs** of an element given the **element's position** in the grid. This is a list of 8, as there are four nodes, each with an x and a y.

```
// returns the DOF indices for an element
int[] edofs(int x, int y)
{
    int n1 = ndof(x, y);
    int n2 = ndof(x + 1, y);
    return new int[] { n1 - 2, n1 - 1, n2 - 2, n2 - 1, n2, n2 + 1, n1, n1 + 1 };
}
```

Assembling the Force Vector

Below is a comparison of my pseudocode from the design section and the code used in my software. In my pseudocode, this was not a function, but I implemented it as a function so my code would look cleaner and I could isolate post-processing debugging.

Pseudocode:

```
// assemble force vector
FOREACH force IN forces
       F[GetIndex(force)] = force.x
       F[GetIndex(force) + 1] = force.y
ENDFOREACH
```

Implemented Code:

```
private Vector<double> GetForceVector()
{
    Vector<double> f = new DenseVector((xSize + 1) * (ySize + 1) * 2);
    foreach (GameObject force in InputHandler2D.forces)
    {
        Vector2 pos = force.GetComponent<RectTransform>().anchoredPosition;
        int n = ndof((int)pos.x, (int)pos.y);
        f[n] = force.GetComponentInChildren<ForceManager>().force.x;
        f[n + 1] = force.GetComponentInChildren<ForceManager>().force.y;
    }
    return f;
}
```

To test this function I added forces to the first few nodes in the structure and manually calculated their actual index so I could compare them. The force at the [0, 0] was added at [0], I added a force at [1, 0], and it got added at [2], then I added one at [0, 1], that got added at [18]. These results are all that was expected, therefore I am satisfied my code works.

Performing FEA — Obj Ref 2.4

After this my program should be able to find the x and y displacements of every node in the structure given the user's constraints. I should also return a matrix of positive values representing how much potential energy an element has, this will be used later for the optimisation process.

The LSM

The LSM is a standard matrix of values which can be hardcoded to save computation as the LSM is constant no matter what the problem conditions are.

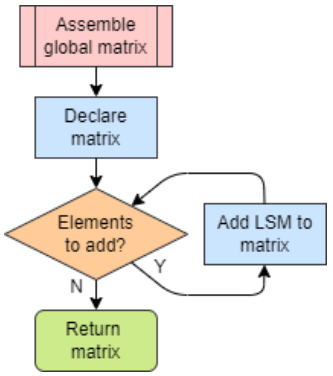
```
// called at the start of runtime
void Start()
{
    EPE = new DenseMatrix(xSize, ySize);
    LSM = DenseMatrix.OfArray(new double[,] {
    { 0.4945054945054945,0.1785714285714285,-0.3021978021978022,-0.01373626373626375,-0.2472
    { 0.1785714285714285,0.4945054945054945,0.01373626373626375,0.0549450549450549,-0.17857
    { -0.3021978021978022, 0.01373626373626375, 0.4945054945054945, -0.1785714285714285, 0.0
    { -0.01373626373626375,0.0549450549450549,-0.1785714285714285,0.4945054945054945,0.0137
    { -0.2472527472527473,-0.1785714285714285,0.0549450549450549,0.01373626373626375,0.4945
    { -0.1785714285714285,-0.2472527472527473,-0.01373626373626375,-0.3021978021978022,0.178
    { 0.0549450549450549,-0.01373626373626375,-0.2472527472527473,0.1785714285714285,-0.302
    { 0.01373626373626375,-0.3021978021978022,0.1785714285714285,-0.2472527472527473,-0.0137
}
```

These values are added to the GSM in the 'GetGlobalMatrix' function.

Assembling the GSM

My pseudocode for this function was very vague as I didn't have a good enough understanding of the function during my design section.

```
FUNCTION GetGlobalMatrix
      // declare matrix
      K = new Matrix

      FOREACH element IN elements
            // add each LSM to 'K'
      ENDFOREACH

      RETURN K
ENDFUNCTION
```



Since then, I have grasped an understanding of this function and managed to replicate the results from TopOpt. My GSM assembly function is on the following page...

```
// assembles the global stiffness matrix
private Matrix<double> GetGlobalMatrix()
{
    int matrixSize = (xSize + 1) * (ySize + 1) * 2;
    Matrix<double> K = new DenseMatrix(matrixSize, matrixSize);

    // for each element
    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            int[] edof = edofs(i, j);
            int xInd = 0, yInd = 0;
            foreach (int y in edof)
            {
                foreach (int x in edof)
                {
                    // add the DOF into the GSM weighted by density
                    K[x, y] += p[i, j] * LSM[xInd, yInd];
                    xInd++;
                }
                yInd++;
                xInd = 0;
            }
        }
    }
    return K;
}
```

Testing the functionality of this function was difficult, as there isn't a standard GSM to compare against, therefore I had to implement the open-source MATLAB code from TopOpt so I could get the GSM. The challenge with doing this is that MATLAB uses 1 based indexing and C# uses 0 based, so translation was slow and difficult, however after each attempt at getting the right indices, I managed to match the two GSMs:

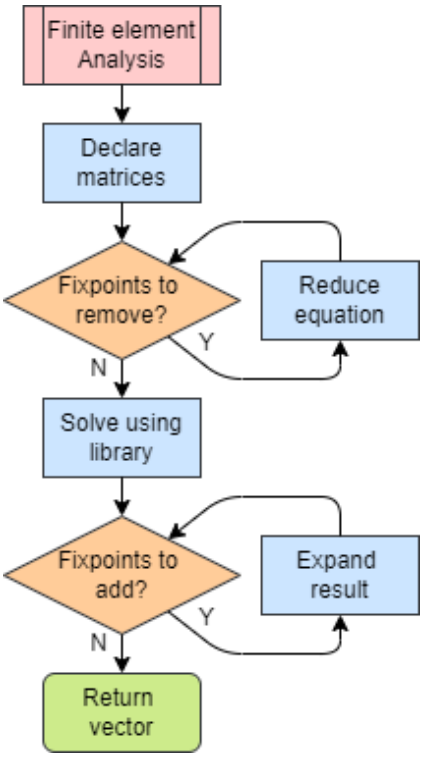
```
0.494505494505495,-0.178571428571428,-0.302197802197802,0.0137362637362638,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
-0.178571428571428,0.494505494505495,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,0,0,0,0,0,0,0,
0.0137362637362638,0.0549450549450549,0,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,0,0,0,0,
0,0,-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,0,0,0,0,0,
0,0,0,0.0137362637362638,0.0549450549450549,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,0,0,
0,0,0,0,0,-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,0,0,
0,0,0,0,0,0.0137362637362638,0.0549450549450549,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,
0,0,0,0,0,0,0,-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,
0,0,0,0,0,0,0,0,0,0.0137362637362638,0.0549450549450549,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,
```

```
0.494505494505495,-0.178571428571428,-0.302197802197802,0.0137362637362638,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
-0.178571428571428,0.494505494505495,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,0,0,0,0,0,0,0,
0.0137362637362638,0.0549450549450549,0,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,0,0,0,0,
0,0,-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,0,0,0,0,0,
0,0,0,0.0137362637362638,0.0549450549450549,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,0,0,
0,0,0,0,0,-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,0,0,
0,0,0,0,0,0.0137362637362638,0.0549450549450549,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,0,0,0,
0,0,0,0,0,0,0,-0.302197802197802,-0.0137362637362638,0.989010989010989,0,-0.302197802197802,0.0137362637362638,0,0,0,
0,0,0,0,0,0,0,0,0,0.0137362637362638,0.0549450549450549,0,0.989010989010989,-0.0137362637362638,0.0549450549450549,0,0,
```

Getting the Displacements

My pseudocode for this function is as follows:

```
FUNCTION GetDisplacementVector(K, F)
      U = new Vector

      // reduce equation using fixpoints
      FOREACH fixpoint IN fixpoints
            // 'GetIndex' returns index of fixpoint
            K.Remove(GetIndex(fixpoint))
            K.Remove(GetIndex(fixpoint) + 1)

            F.Remove(GetIndex(fixpoint))
            F.Remove(GetIndex(fixpoint) + 1)
      ENDFOREACH

      // linear algebra solver from 'Math.NET'
      U = K.Solve(F)

      // expand equation using fixpoints
      FOREACH fixpoint IN fixpoints
            // fill fixpoints with 0 displacement
            K.Add(GetIndex(fixpoint), 0)
            K.Add(GetIndex(fixpoint) + 1, 0)
      ENDFOREACH
      RETURN U
ENDFUNCTION
```



However, when it came to implementing this function, I realised that removing the fixpoint indices and adding them back in was harder than I thought, as the size of the data structures change, and therefore so do the indices. I decided to split this into two functions called 'ReduceEquation' and 'ExpandResult' and a line of code for the solve function.

These two new functions make use of yet another new function called 'GetFixpointIndices' which returns a sorted list of indices to remove from the equation.

```
// returns a sorted list of DOFs for all fixpoints
List<int> GetFixpointIndices()
{
    List<int> indices = new List<int>();
    foreach (GameObject f in InputHandler2D.fixpoints)
    {
        Vector2 pos = f.GetComponent<RectTransform>().anchoredPosition;
        if (f.GetComponentInChildren<FixpointManager>().x) { indices.Add(ndof((int)pos.x, (int)pos.y)); }
        if (f.GetComponentInChildren<FixpointManager>().y) { indices.Add(ndof((int)pos.x, (int)pos.y) + 1); }
    }
    indices.Sort();
    return indices;
}
```

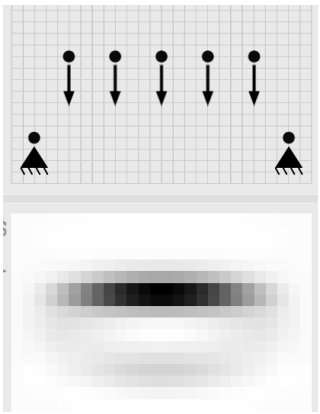
'ReduceEquation' and 'ExpandResult' are on the next page...

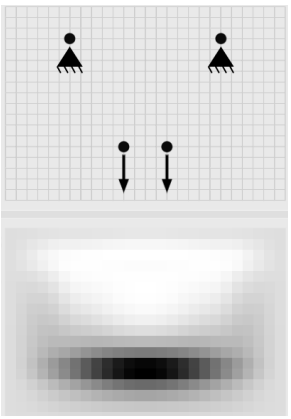
```
// remove fixed DOFs from force vector and GSM
private void ReduceEquation()
{
    // convert to matrix as vector doesn't support row/column removal
    Matrix<double> F_asColMat = F.ToColumnMatrix();
    List<int> toRemove = GetFixpointIndices();
    for (int i = toRemove.Count - 1; i >= 0; i--)
    {
        F_asColMat = F_asColMat.RemoveRow(toRemove[i]);
        GSM = GSM.RemoveRow(toRemove[i]).RemoveColumn(toRemove[i]);
    }
    F = new DenseVector(F_asColMat.ToColumnArrays()[0]);
}
// add zeros in place of the fixpoints
private void ExpandResult()
{
    Vector<double> zero = new DenseVector(1);
    List<double> U_asList = U.ToArray().ToList();
    List<int> toAdd = GetFixpointIndices();
    for (int i = 0; i < toAdd.Count; i++)
    {
        if (toAdd[i] >= U_asList.Count) {  U_asList.Add(0); }
        else { U_asList.Insert(toAdd[i], 0); }
    }
    U = new DenseVector(U_asList.ToArray());
}
```

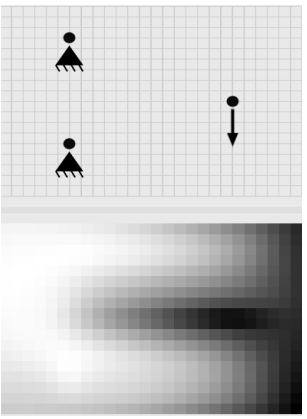
These functions can't be tested as the matrices and vectors are too large to display or notice a difference with the human eye, however, if I test them all together, I will be able to see on the coloured texture produced whether these have worked or not. These FEA functions are called in the following order:

```
GSM = GetGlobalMatrix(); // assemble GSM
F = GetForceVector(); // assemble force vector
ReduceEquation(); // remove fixpoints
U = GSM.Solve(F); // solve linear system
ExpandResult(); // add fixpoints
```

The following test outputs displacements to the display, the expected results are groupings of displacement mostly around the force (black is the most displacement, white is the least)







Given that these example cases produce the expected result, the test was successful and these functions all work.

Calculating Potential Energy

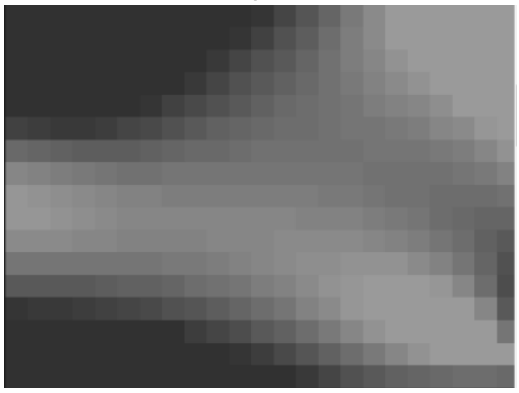
Converting to potential energy is a simple equation. Hooke's law states that potential energy is given by EPE = 0.5 ku^2, in my system, k is the LSM and U is the displacement vector. Therefore the following code is all that is needed to convert from displacements to potential energy:

```
// using the displacement vector, compute elemental energy, then output density
Matrix<double> GetPotentials()
{
    Matrix<double> temp = new DenseMatrix(xSize, ySize);
    Vector<double> u = new DenseVector(8); // holds the displacements of the elements

    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            // adds the displacements to the element dislacement vector
            int[] edof = edofs(i, j);
            for (int x = 0; x < 8; x++) { u[x] = U[edof[x]]; }

            // calculate stress
            temp[i, j] = (u.ToRowMatrix() * LSM * u)[0];
        }
    }
    return temp;
}
```

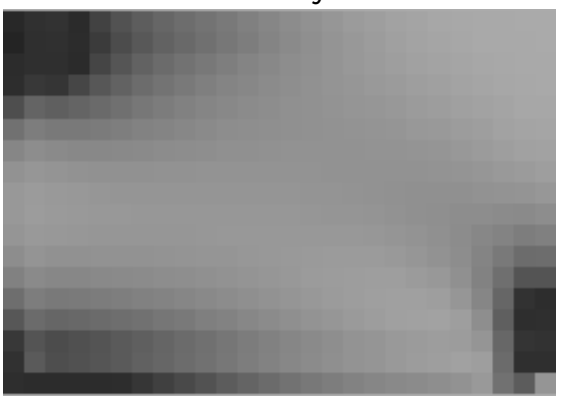
This function outputs a matrix of values which can be passed forward to the optimisation functions for redistribution.

To test this, I can use the first iteration of the MATLAB code to test what the rough shape of the potential energies is supposed to look like for a half MBB beam. This is obviously before the density filter and the criteria update have been applied. For other setups, I have made a rough prediction of where the domain will be most stressed.

MATLAB prediction



Result of my code



As seen, there are concentrations of material in all corners but the top right are the same. The diagonal connection between the top left and the bottom right corners is visible, however, it is not as prevalent in my result. Overall, I feel my code is evaluating the potential energies successfully.

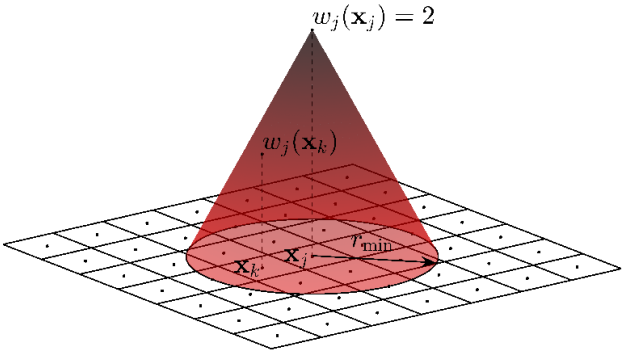
Performing Topology Optimisation — Obj Ref 3

After this section, my program should be able to produce a structure which is the optimal structure for withstanding the load and matches the user's volume constraint.

Filtering the Densities

After FEA, the potential energies are not distributed very evenly and are often concentrated in pinpoints within the structure. The density filter effectively blurs the densities to allow for the structure to reach further when making connections.

The filter uses a weighted average of the surrounding element densities to calculate the new density for that element.



A weighted average formula takes into account the priority of each element in the sum, in this case, the multiplier is the distance from that element (the closer elements have more effect)

In my design section, I wrote some pseudocode for this algorithm:

```
FUNCTION FilterDensities(p)
      rmin = 2 // filter radius
      FOR j = 0 to ySize
            FOR i = 0 to xSize
                  adj = new List // adjacent elements
                  FOR v = -rmin to rmin
                        FOR u = -rmin to rmin
                              adj.Add(p[i+u, j+v])
                        ENDFOR
                  ENDFOR
                  pNew[i, j] = adj.Average()
            ENDFOR
      ENDFOR
      RETURN pNew
ENDFUNCTION
```

I converted this code to C# and used it in my application. The only problem with the pseudocode is that I didn't consider index out of range errors, however, I fixed that during my implementation by adding if statements to the for loops. My code and tests are on the following page...

```
Matrix<double> Filter(Matrix<double> matrix)
{
    Matrix<double> temp = new DenseMatrix(xSize, ySize);
    // for each element
    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            // initialise list for storingg wieghted densities
            List<double> weights = new List<double>();

            // search in a small square for elements within the raduis
            for (int v = -(int)rMin; v < (int)rMin; v++)
            {
                if (j + v > 0 && j + v < ySize) // ignore out of bounds
                {
                    for (int u = -(int)rMin; u < (int)rMin; u++)
                    {
                        if (i + u > 0 && i + u < xSize) // ignore out of bounds
                        {
                            double dist = Sqrt(v * v + u * u);
                            if (dist < rMin)
                            {
                                // add element with a weight of it's distance ratio
                                weights.Add((dist / rMin) * matrix[i + u, j + v]);
                            }
                        }
                    }
                }
            }
            temp[i, j] = weights.Average();
        }
    }
    return temp;
}
```

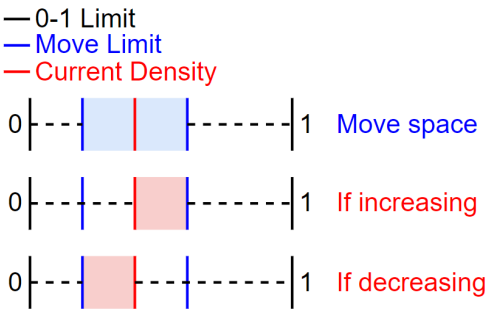
The result of adding this to my code should be a blurred effect on the texture. This should be visible, and easy to test. The following are some of the results:

TopOpt Result | My Unfiltered Result | My Filtered Result

The results of this function match what was expected, and bring the image closer to what is expected. The next function should make the image a match for the TopOpt image, this is called the criteria update

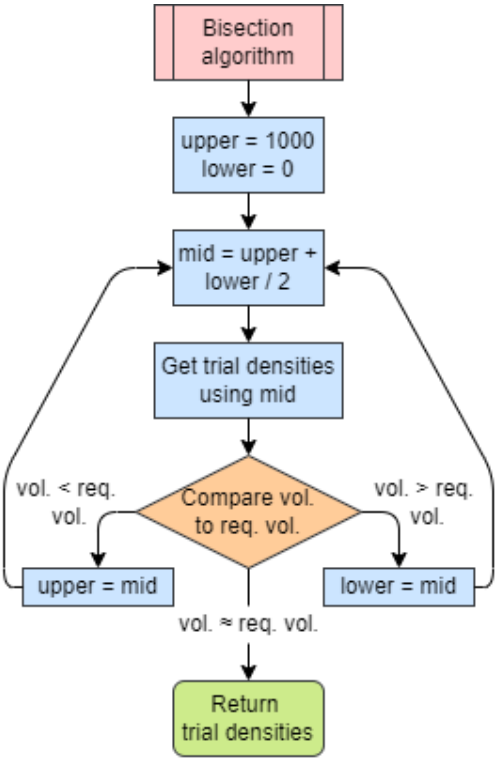
Criteria Update

The criteria update selectively adds or removes density from elements to ensure the entire volume of the structure matches the user's criteria. It does this via a scale factor called lambda and four limits. The move limit is a constraint to stop elements from changing too much and is normally set to around 0.2, so each element with a density of 0-1 can change by ±0.2. The other two limits are 0 and 1, so the element doesn't exceed its boundaries. These limits set up the problem to find lambda. To find lambda I am using a bisection algorithm, which is effectively a binary search.



The limits above set up a function for the 'trial density' which is calculated every time a guess is made at lambda. As these are directly related, we can increase or decrease lambda to increase or decrease the trial density of that iteration. Below is the pseudocode from the design section:

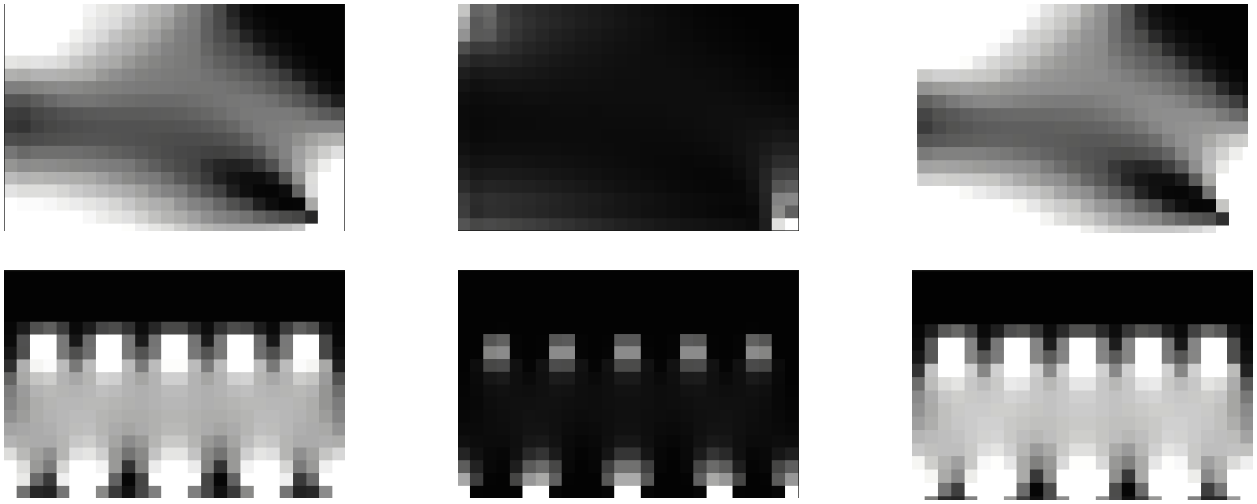
```
FUNCTION CriteriaUpdate(p)
      // declare variables
      lower = 0, upper = 10000

      // get user input for 'reqVol'
      reqVol = GetVolume(), vol = Infinity
      tolerance = 0.1

      WHILE (vol - reqVol).Abs() > tolerance
            mid = (upper + lower) / 2
            FOREACH value IN p
                  // update value using mid
                  value = // new density
            ENDFOREACH

            vol = p.Sum / (xSize * ySize)
            IF vol > reqVol
                  lower = mid
            ELSE
                  upper = mid
            ENDIF
      ENDWHILE
ENDFUNCTION
```



On the following page is the bisection algorithm from my implemented code...

```
// update to match criteria
Matrix<double> pTrial = p;
double lambda, lower = 0, upper = 10000000;
for (int attempts = 0; attempts < 100; attempts++)
{
    // break if conditions are met (within a tolerance band)
    if (Abs(pTrial.RowSums().Sum() / (xSize * ySize) - reqVol) < tole

    // set lambda to be the midpoint
    lambda = (upper + lower) / 2;

    // find trial structure
    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            // trial function (move is the move limit)
            pTrial[i, j] = Max(pMin, Max(p[i, j] - move, Min(1f,
                Min(p[i, j] + move, p[i, j] * Sqrt(-dc[i, j] / lambda)
        }
    }

    // eliminate half of the search area
    if (pTrial.RowSums().Sum() / (xSize * ySize) > reqVol) { lower = l
}
p = pTrial;
```

TopOpt Result | My Filtered Result | My Final Result



As seen, my result now matches TopOpt's result for the first iteration. This test is therefore successful and I need to now set up the main loop so it iterates until it has converged to the best structure.

Implementing Presets — Obj Ref 4

Now, I thought it would be a good idea to implement presets so further testing would be easier. This means I can just click a button and the structure I made beforehand will appear.

Preset class

Here is the pseudocode compared to my actual code for the preset class which holds all of the data required to store a preset:

```
CLASS PresetClass
    // the class which holds all required data for a preset
    public serializable (float, float, bool, bool)[] _fixpoints
    public serializable (float, float, float, float)[] _forces

    public PROCEDURE new // constructor for PresetClass
        _fixpoints = new (float, float, bool, bool)[]
        _forces = new (float, float, float, float)[]

        FOREACH GameObject IN forces
            // get data and add to list
        ENDFOREACH

        FOREACH GameObject IN fixpoints
            // get data and add to list
        ENDFOREACH
    ENDPROCEDURE
ENDCLASS
```

```
[System.Serializable]
public class Preset
// class for storing data in a useable format
{
    [SerializeField] public List<(float, float, bool, bool)> _fixpoints;
    [SerializeField] public List<(float, float, float, float)> _forces;

    public Preset()
    {
        _fixpoints = new List<(float, float, bool, bool)>();
        _forces = new List<(float, float, float, float)>();

        foreach (GameObject n in InputManager.fixpoints)
        {
            _fixpoints.Add((n.GetComponent<RectTransform>().anchoredPosition.x,
                            n.GetComponent<RectTransform>().anchoredPosition.y,
                        n.GetComponentInChildren<FixpointManager>().x,
                        n.GetComponentInChildren<FixpointManager>().y));
        }

        foreach (GameObject n in InputManager.forces)
        {
            _forces.Add((n.GetComponent<RectTransform>().anchoredPosition.x,
                        n.GetComponent<RectTransform>().anchoredPosition.y,
                        n.GetComponentInChildren<ForceManager>().force.x,
                        n.GetComponentInChildren<ForceManager>().force.y));
        }
    }
}
```

Serialising the Presets

This class will convert the preset class into binary and store it in a valid file location. Here is my code for that:

```
CLASS PresetSerializer
    public static string directoryPath

    public static FUNCTION Save(string name) // true if save was successful
        filePath = directoryPath + "\" + name

        // ensures there's always a directory to store presets
        IF directoryPath == null
            new Directory(directoryPath)
        ENDIF

        // if file doesn't already exist, create preset and save
        IF filePath == null
            PresetClass currentPreset = new PresetClass()
            File file = new File(name)
            file.OpenWrite(Serialize(currentPreset))
            file.Close()
            RETURN true
        ELSE
            RETURN false
        ENDIF
    ENDFUNCTION

    public static FUNCTION Load(string name)
        // returns the class stored in the file
        PresetClass data = Deserialize(File.OpenRead(name))
        file.Close()
        RETURN data
    ENDFUNCTION

    public static FUNCTION Delete(string name)
        // deletes preset
        File.Delete(name)
    ENDFUNCTION
ENDCLASS
```

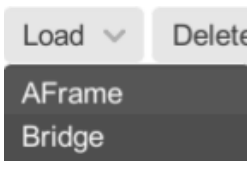
On the next page is the class I created. I added functions for getting the directory path and file path so the only thing that needs passing into the functions is the actual name of the preset.

```
public static class PresetSerializer
{
    // methods which return paths to the directory and the preset file
    public static string GetDirectoryPath() { return Application.persistentDataPath + "/Presets"; }
    public static string GetFilePath(string name) { return GetDirectoryPath() + "/" + name + ".preset"; }

    // saves preset (returns bool to handle errors)
    public static bool SavePreset(string name)
    {
        // ensures directory exists
        if(!Directory.Exists(GetDirectoryPath())) { Directory.CreateDirectory(GetDirectoryPath()); }

        if (!File.Exists(GetFilePath(name)))
        {
            Preset currentPreset = new Preset(); // create new preset
            BinaryFormatter formatter = new BinaryFormatter(); // initialise a formatter
            FileStream stream = new FileStream(GetFilePath(name), FileMode.Create); // create new file
            formatter.Serialize(stream, currentPreset); // convert to binary
            stream.Close(); // close file stream

            return true;
        }
        else { ErrorText.ShowError("Name already exists"); return false; }
    }

    // loads preset files
    public static Preset LoadPreset(string name)
    {
        BinaryFormatter formatter = new BinaryFormatter(); // initialises binary formatter
        FileStream stream = new FileStream(GetFilePath(name), FileMode.Open);
        Preset data = formatter.Deserialize(stream) as Preset; // returns values in useable format
        stream.Close(); // closes connection
        return data;
    }

    public static void DeletePreset(string name) { File.Delete(GetFilePath(name)); }
}
```

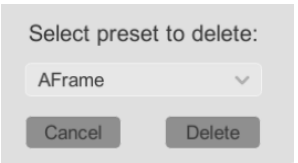
Setting up the Inputs

For saving presets, I created a button at the top of the screen in the toolbar labelled ‘save’, and set it up such that a pop up appears when it is pressed. This pop up as seen on the right consists of two buttons and an input field for the name of the preset. The input has validation attached to only allow alphanumeric characters with a character limit of 20. The ‘cancel’ button gets rid of the pop-up and the save button calls the save function in the PresetManager

For loading presets, I used a dropdown menu. When it’s opened, it calls a function in the preset manager which accesses all of the current preset names and then displays them in a list. Each option is a button, and when one is selected it calls ‘load’ and the preset is instantly loaded to the constraint editor



Deletion of a preset is also done using a button, on click, a pop up appears with a dropdown to select the desired preset and a delete button to confirm the user's choice. It also has a cancel button to get rid of the pop up in case the user changes their mind.



Using the Presets

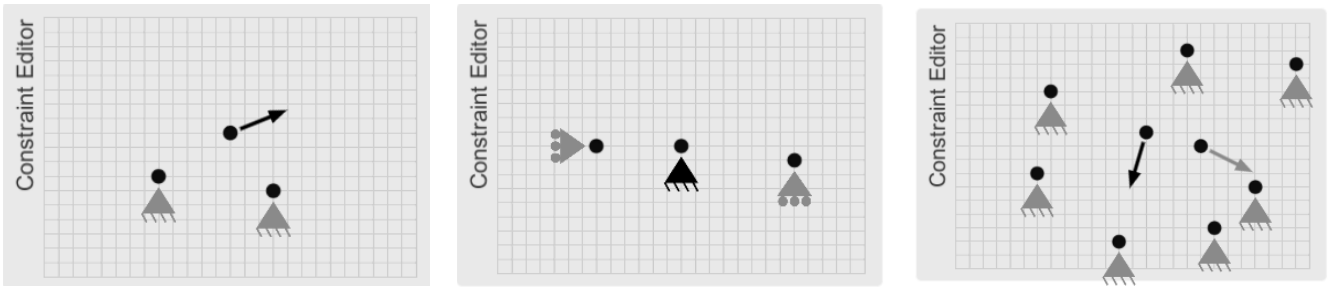
The PresetManager class handles the calling and manipulation of the other two classes. Here my entire PresetManager Class:

```
public class PresetManager : MonoBehaviour
{
    public Dropdown loadMenu;
    public Dropdown deleteMenu;
    public GameObject inputBarrier;
    public GameObject savePopUp;
    public InputManager inputManager;

    public void Save(InputField input)
    {
        if (input.text == "") { ErrorText.ShowError("Preset name cannot be empty"); }
        else if (PresetSerializer.SavePreset(input.text))
        {
            inputBarrier.SetActive(false);
            savePopUp.SetActive(false);
        }

        UpdateLoadMenu();
    }

    public void Load(Text name)
    {
        inputManager.Reset();
        Preset preset = PresetSerializer.LoadPreset(name.text);
        foreach ((float, float, bool, bool) data in preset._fixpoints)
        {
            inputManager.AddFixpointFromPreset(data);
        }
        foreach((float, float, float, float) data in preset._forces)
        {
            inputManager.AddForceFromPreset(data);
        }
    }

    public void Delete(Dropdown menu)
    {
        PresetSerializer.DeletePreset(menu.options[menu.value].text);
        UpdateLoadMenu();
    }

    public void DeleteAll()
    {
        if (Directory.Exists(PresetSerializer.GetDirectoryPath()))
        {
            Directory.Delete(PresetSerializer.GetDirectoryPath(), true);
        }
        Directory.CreateDirectory(PresetSerializer.GetDirectoryPath());

        UpdateLoadMenu();
    }
```

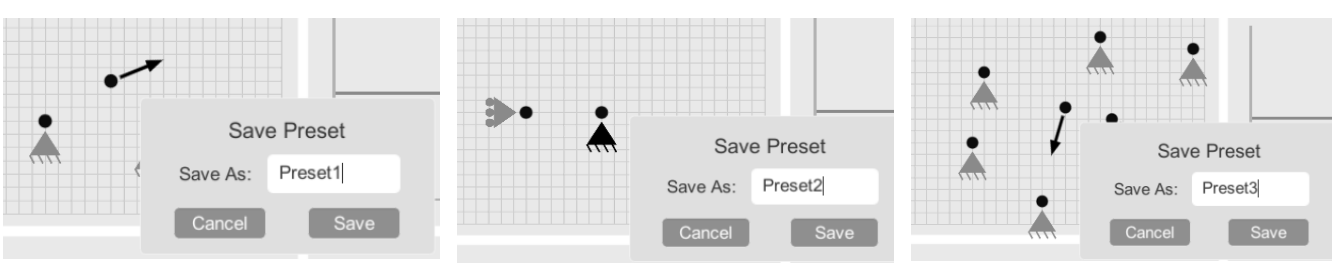
```
    public void UpdateLoadMenu()
    {
        DirectoryInfo info = new DirectoryInfo(PresetSerializer.GetDirectoryPath());
        var fileInfo = info.GetFiles();

        loadMenu.ClearOptions();

        foreach (var file in fileInfo)
        {
            Dropdown.OptionData option = new Dropdown.OptionData();
            option.text = Path.GetFileName(file.ToString()).Replace(".preset", "");
            loadMenu.options.Add(option);
        }
    }

    public void UpdateDeleteMenu()
    {
        DirectoryInfo info = new DirectoryInfo(PresetSerializer.GetDirectoryPath());
        var fileInfo = info.GetFiles();

        deleteMenu.ClearOptions();

        foreach (var file in fileInfo)
        {
            Dropdown.OptionData option = new Dropdown.OptionData();
            option.text = Path.GetFileName(file.ToString()).Replace(".preset", "");
            deleteMenu.options.Add(option);
        }
        deleteMenu.value = 0;
    }
}
```

To test the functionality of my preset system, I will now create 3 unique models, load all three in turn, ensuring the forces and fixpoints are identical, delete one, then delete the other two.

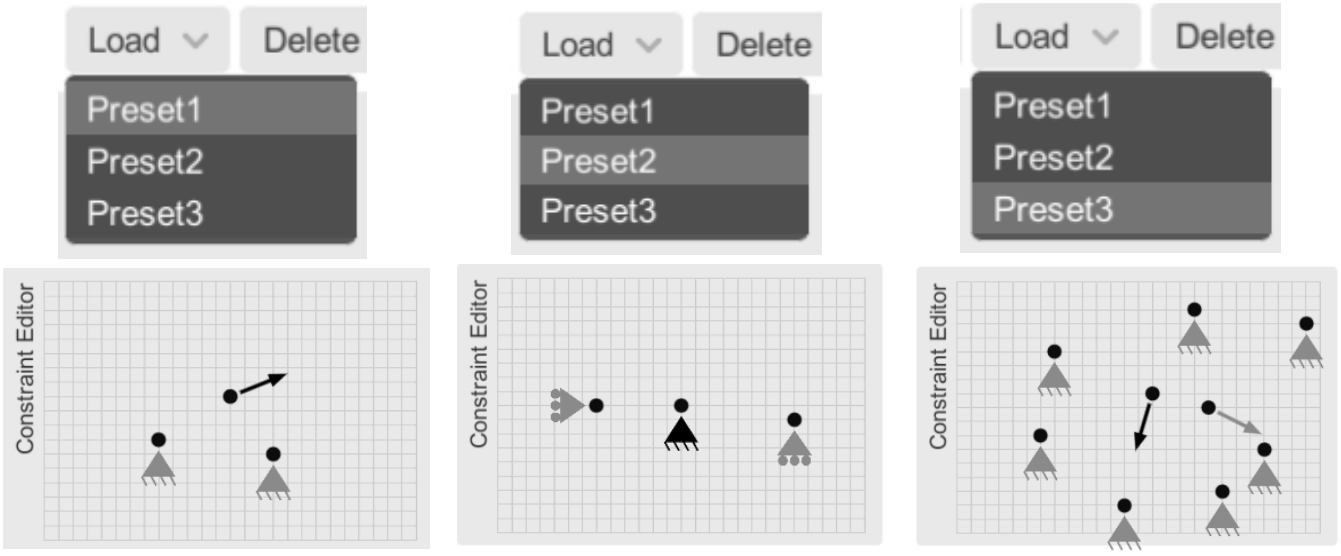
Creating



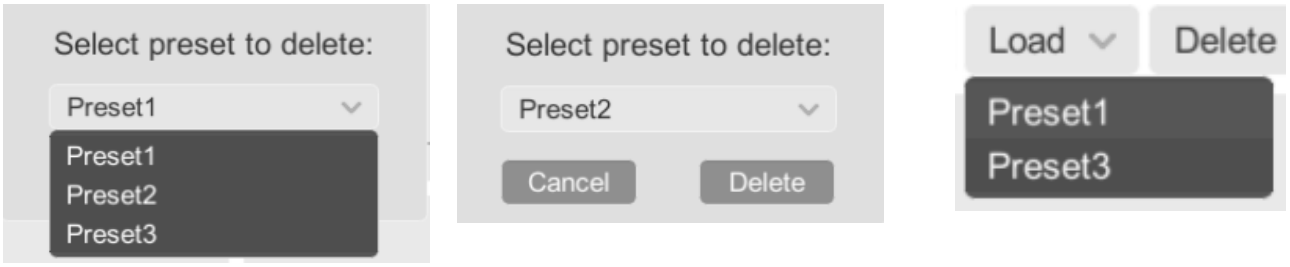
Saving



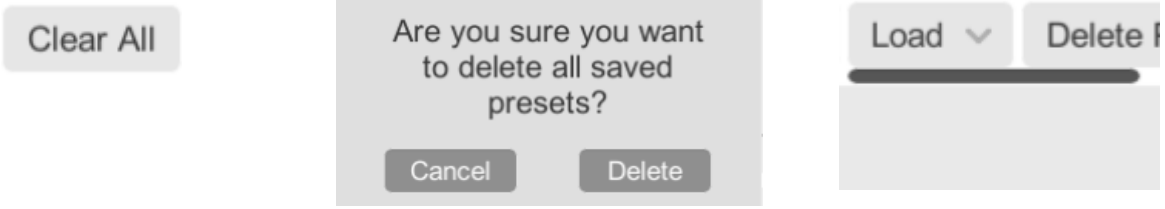
Loading



Deleting One



Deleting All



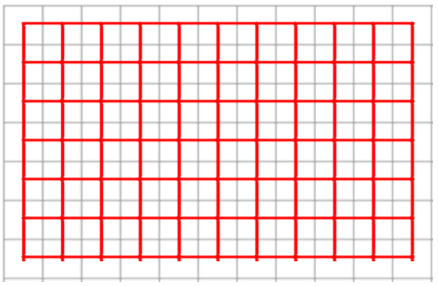
As seen above, all buttons, features and pop-ups are working exactly how they should be. In the testing section of my coursework, I will make more vigorous attempts to find faults in my system, however, the core functionality works for now, which means testing will be a lot quicker when building the mesh generator as follows.

Mesh Generation — Obj Ref 5.1.4

When this is complete, there should be an output section of the screen which displays a mesh of the structure, and that mesh should deform in the same way as the structure would if it was under the user's conditions. The amount it displaced should be changeable by a slider.

The Initial Mesh

Making an interpolated mesh using marching cubes requires the data points to be on the vertices of the grid that it's based upon. This is a problem, because it's the densities that are used for the generation, and they are taken from the cells (elements). I decided to solve this by making a mesh from a grid that is one row and column smaller than the structure because then it's the right size for all the data. As seen on the right, the grey grid is the structure of the elements, and the red is the grid that will be used to render the mesh.



Using this grid of data and the pseudocode from the design section, I was able to implement the code to generate a mesh. Here is the pseudocode from my design section:

```
FUNCTION GenerateMesh(p)
      Mesh mesh = new Mesh()
      FOR j = 0 to ySize
            FOR i = 0 to xSize
                  arr = [false, false, false, false]
                  // get values of the corners of the square
                  FOR v = 0 to 1
                        FOR u = 0 to 1
                              IF p[i + u, j + v] > 0.5f
                                    arr[i + 2 * j] = true; // inside
                              ELSE
                                    arr[i + 2 * j] = false; // outside
                              ENDIF
                        ENDFOR
                  ENDFOR
                  Mesh elementMesh = new Mesh;
                  elementMesh.vertices = relativePoints;
                  elementMesh.triangles = triTable[arr.ToDec];
                  mesh = Combine(mesh, elementMesh)
            ENDFOR
      ENDFOR
      RETURN mesh
ENDFUNCTION
```

The following page contains the code I used in my application...

This is the lookup table to find which triangles should be drawn to match the shape.

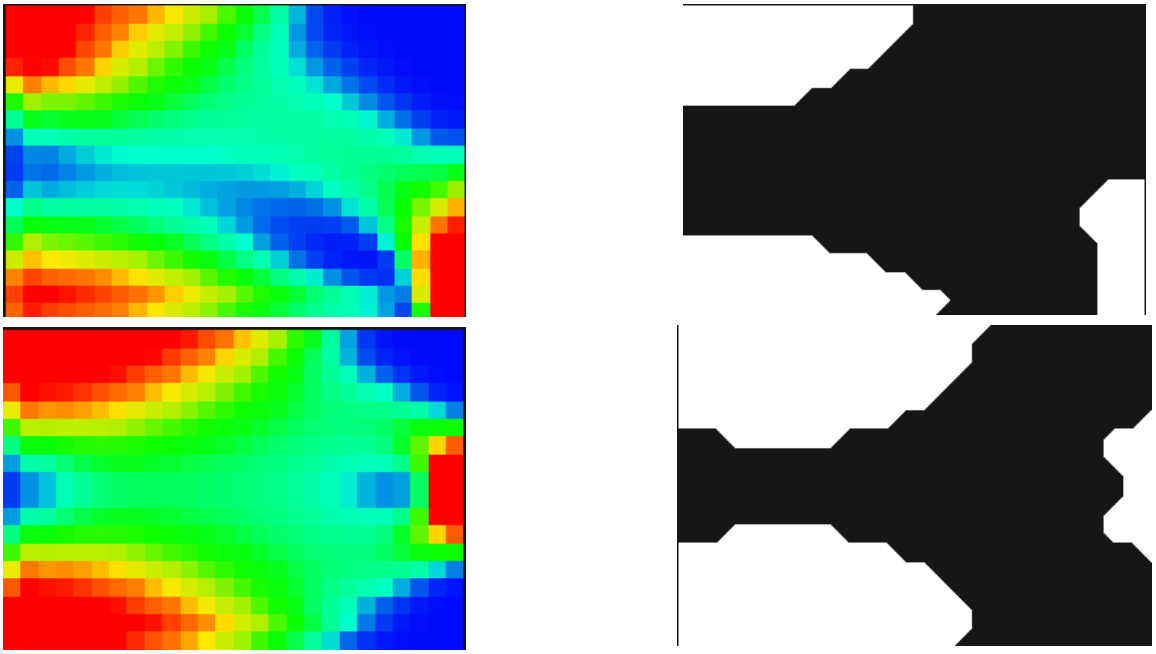
```
[HideInInspector] // holds the indices of vertices for each triangle
private static List<int[]> triTable = new List<int[]>() {
    new int[] { },                          //0000
    new int[] { 7, 6, 3 },                  //0001
    new int[] { 6, 5, 2 },                  //0010
    new int[] { 3, 7, 5, 3, 5, 2 },         //0011
    new int[] { 7, 0, 4 },                  //0100
    new int[] { 3, 0, 4, 3, 4, 6 },         //0101
    new int[] { 7, 0, 4, 2, 6, 5 },         //0110
    new int[] { 3, 0, 4, 3, 4, 5, 3, 5, 2 }, //0111
    new int[] { 5, 4, 1 },                  //1000
    new int[] { 7, 6, 3, 5, 4, 1 },         //1001
    new int[] { 6, 4, 2, 2, 4, 1 },         //1010
    new int[] { 3, 7, 2, 2, 7, 4, 2, 4, 1 }, //1011
    new int[] { 7, 0, 1, 7, 1, 5 },         //1100
    new int[] { 0, 1, 5, 0, 5, 6, 0, 6, 3 }, //1101
    new int[] { 1, 2, 6, 1, 6, 7, 1, 7, 0 }, //1110
    new int[] { 0, 1, 3, 1, 2, 3 } };       //1111
```

This is the function that the optimiser script calls to render the whole mesh. I decided to make GetElementMesh a separate function to abstract the code from the code which combines the meshes.

```
public void Render(Matrix<double> p, bool isColored)
{
    // combines every element mesh
    CombineInstance[] combine = new CombineInstance[(xSize) * (ySize)];
    int meshIndex = 0;
    for (int i = 0; i < xSize; i++)
    {
        for (int j = 0; j < ySize; j++)
        {
            combine[meshIndex].mesh = GetElementMesh(i, j, p);
            combine[meshIndex].transform = Matrix4x4.Translate(new Vector3(i + 0.5f, j + 0.5f, 0f));
            meshIndex++;
        }
    }

    // renders mesh
    GetComponent<MeshFilter>().mesh = new Mesh();
    GetComponent<MeshFilter>().mesh.CombineMeshes(combine);
}
```

```
public Mesh GetElementMesh(int x, int y, Matrix<double> p)
{
    bool[] arr = new bool[4];

    //for each vertex
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            // generate four bits, one for each vertex
            if (p[x+i, y+j] >= 0.5f) { arr[i+2*j] = true; } // true = inside
            else { arr[i+2*j] = false; }
        }
    }

    // convert to integer
    BitArray bitArray = new BitArray(arr);
    int[] array = new int[1];
    bitArray.CopyTo(array, 0);

    // output the pixel mesh
    Mesh voxelMesh = new Mesh();
    voxelMesh.vertices = relativePoints;
    voxelMesh.triangles = triTable[array[0]];
    return voxelMesh;
}
```

I thought it would be a good idea to test my code before I attempt interpolation, so here are the result of the mesh after one iteration of the optimisation:



Since the cutoff point is 0.5, which decides whether the point is inside or outside the shape, that means the line for the structure should cut through the colour green on the texture. This is a successful test because the mesh draws lines around all of the green parts of the texture.
Instead of interpolating the mesh, what I decided to do next was add colour, so the user could see the stresses within the mesh, as well as the texture.

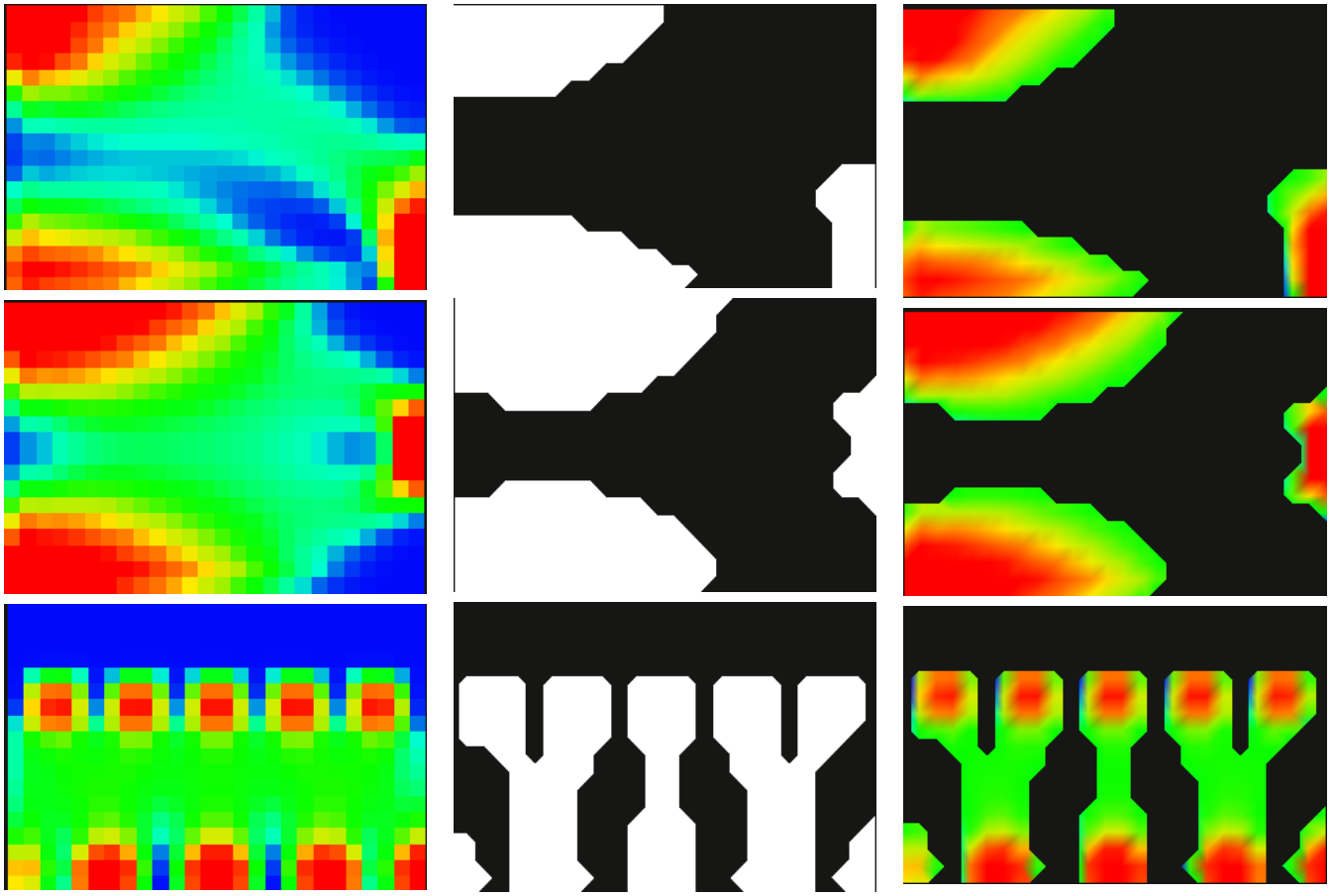
Adding Colour

For this, I need to use the density of each element to set the colours at each vertex. Luckily, unity's Universal Render Pipeline asset has a 'vertex shader' which means if I set each vertex colour in a mesh, unity will automatically interpolate between each corner in a mesh and make a gradient.

The following code is used to set all of the colours of the mesh at once:

```
// sets the corner of each mesh to a color based on stress
// a vertex shader handles the color gradient
if (isColored)
{
    Mesh chunkMesh = GetComponent<MeshFilter>().mesh;
    Color[] colors;
    colors = new Color[chunkMesh.vertices.Length];
    for (int i = 0; i < colors.Length; i++)
    {
        colors[i] = gradient.Evaluate((float)p[(int)chunkMesh.vertices[i].x, (int)chunkMesh.vertices[i].y]);
    }
    chunkMesh.colors = colors;
}
```

I tried this with the previous two meshes and the results looked successful:



After running this test, it is clear to see how the mesh relates back to the structure, and where the cutoff point is (the green border).

Interpolation

This is where the midpoints of the element meshes need to be changed to stop the mesh from looking so blocky. I need to perform a linear interpolation changing the coordinate vector of each midpoint in either the x or y to sit somewhere between the two vertices. This is how I did it:

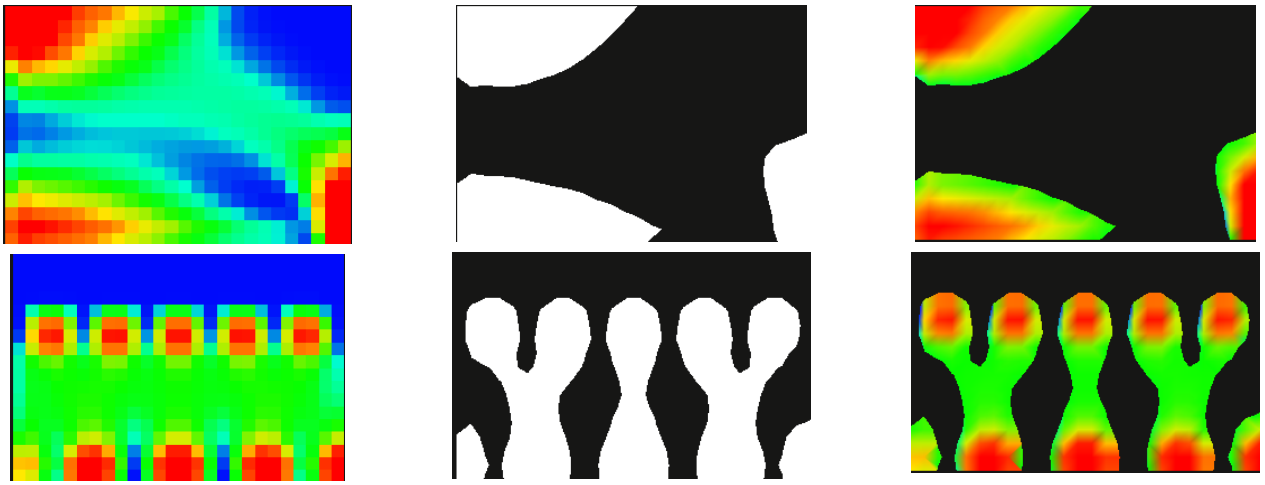
```
Vector3[] relativePoints = new Vector3[8]
{ new Vector3 (-0.5f, 0.5f, 0f), new Vector3 ( 0.5f, 0.5f, 0f), // vertices
  new Vector3 ( 0.5f,-0.5f, 0f), new Vector3 (-0.5f,-0.5f, 0f), //
  new Vector3 (0f, 0.5f, 0f),    new Vector3 ( 0.5f, 0f, 0f),   // midpoints
  new Vector3 (0f,-0.5f, 0f),    new Vector3 (-0.5f, 0f, 0f)};  //

relativePoints[4] = new Vector3(Mathf.InverseLerp((float)p[x, y + 1], (float)p[x + 1, y + 1], 0.5f) - 0.5f, 0.5f, 0f);
relativePoints[5] = new Vector3(0.5f, Mathf.InverseLerp((float)p[x + 1, y], (float)p[x + 1, y + 1], 0.5f) - 0.5f, 0f);
relativePoints[6] = new Vector3(Mathf.InverseLerp((float)p[x, y], (float)p[x + 1, y], 0.5f) - 0.5f, -0.5f, 0f);
relativePoints[7] = new Vector3(-0.5f, Mathf.InverseLerp((float)p[x, y], (float)p[x, y + 1], 0.5f) - 0.5f, 0f);
```

I ran this code using the half MBB model:

This has made the mesh smoother, however, it was running quite slowly until I realised it was interpolating elements inside and outside the shape, which didn't need interpolating at all. To improve performance, I only interpolated on the edges of the structure, where it's needed:

```
//if the element is on the edge of the mesh, interpolate midpoints
if ((arr[0] || arr[1] || arr[2] || arr[3]) && !(arr[0] && arr[1] && arr[2] && arr[3]))
{
    relativePoints[4] = new Vector3(Mathf.InverseLerp((float)p[x, y + 1], (float)p[x + 1, y + 1], 0.5f) - 0.5f, 0.5f, 0f);
    relativePoints[5] = new Vector3(0.5f, Mathf.InverseLerp((float)p[x + 1, y], (float)p[x + 1, y + 1], 0.5f) - 0.5f, 0f);
    relativePoints[6] = new Vector3(Mathf.InverseLerp((float)p[x, y], (float)p[x + 1, y], 0.5f) - 0.5f, -0.5f, 0f);
    relativePoints[7] = new Vector3(-0.5f, Mathf.InverseLerp((float)p[x, y], (float)p[x, y + 1], 0.5f) - 0.5f, 0f);
}
```

This simple optimisation allowed the code to run faster, and keep processing down to a minimum. I decided to run more tests to see if the colour worked and if there were any errors.



Displacement Visualiser

I have a separate section for a displaced mesh, with a different mesh entirely. I will start by colouring this mesh in a way where the user can see what displacements are happening in their structure, as plain white means you can't see what's happening on the inside of the model. After trialling different colours and patterns, I decided to use the following code to generate the pattern seen on the right (colours inverted):

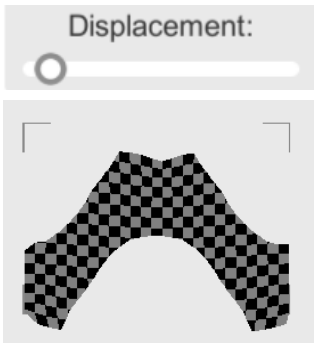
```
Color[] colors;
colors = new Color[voxelMesh.vertices.Length];
for (int i = 0; i < colors.Length; i++)
{
    if (((x + y) % 2) == 0) { colors[i] = Color.grey; }
    else { colors[i] = Color.white; }
}
voxelMesh.colors = colors;
```

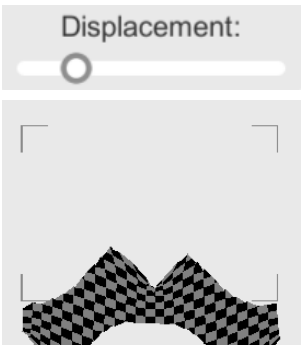
To get each element's node displacements, I wrote a function called eDisps which returns 8 floats for the horizontal and vertical displacements of the corresponding nodes:

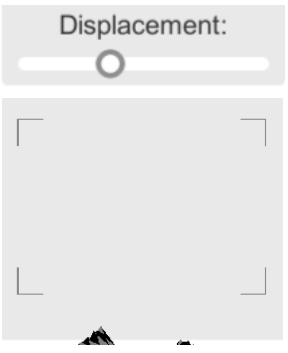
```
List<float> eDisps(int x, int y)
{
    int n1 = 2 * ((ySize + 2) * x + 1 + ySize - y);
    int n2 = 2 * ((ySize + 2) * (x + 1) + 1 + ySize - y);
    int[] e = new int[] { n1 - 2, n1 - 1, n2 - 2, n2 - 1, n2, n2 + 1, n1, n1 + 1 };

    List<float> l = new List<float>();
    foreach (int i in e) { l.Add((float)U[i]); }
    return l;
}
```

All I need to do now is add these displacements to the relative points, so that's exactly what the code below does. I multiplied the displacement by the 'dispExg' so I could control how much the model displaced.

```
List<float> u = eDisps(x, y) *= dispExg.value;
relativePoints[0] += new Vector3(u[0], u[1], 0f);
relativePoints[1] += new Vector3(u[2], u[3], 0f);
relativePoints[2] += new Vector3(u[4], u[5], 0f);
relativePoints[3] += new Vector3(u[6], u[7], 0f);
```





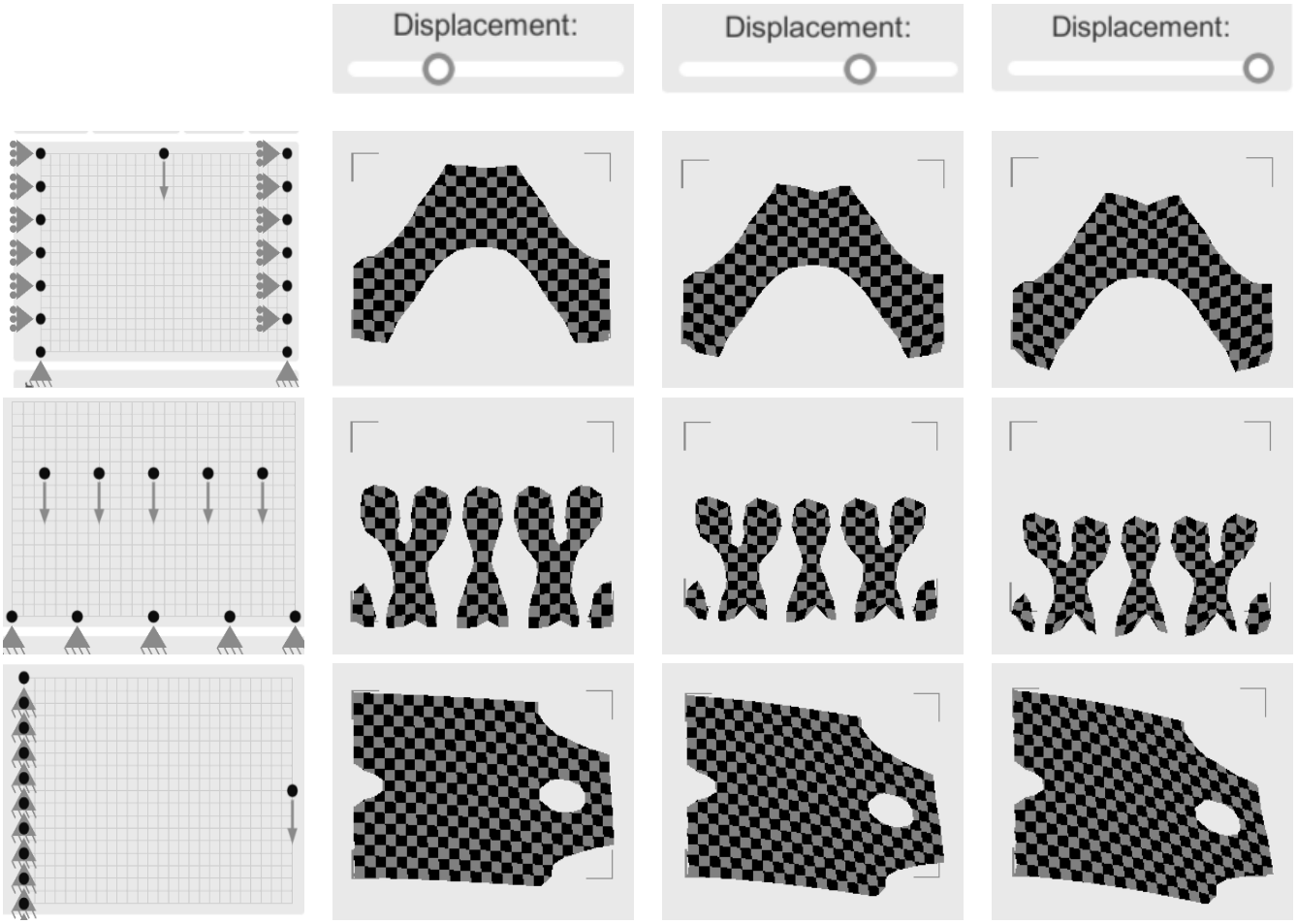


Testing highlighted that the displacements make the model go off-screen, which can't happen. I will fix this by setting the displacements to all be between 0 and 1 so they only move on screen.

Here is the code I wrote in order to interpolate the vector before processing:

```
// if theres a displacement vector passed in, interpolate it between 0 and 1
if (U != null)
{
    double max = 0;
    for (int i = 0; i < U.Count; i++) { if (Abs(U[i]) > max) { max = Abs(U[i]); } }
    for (int i = 0; i < U.Count; i++) { U[i] /= max; }
    U *= dispExg.value;
}
```

This worked successfully and the elements would only move on unit maximum. I changed the slider's maximum value to be the width of the border that the model can move into. The following tests show the structures under displacement and all tests were successful:



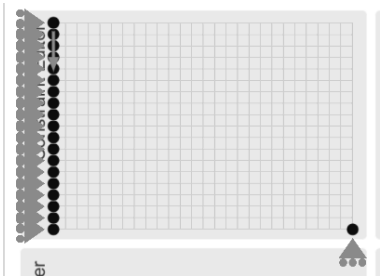
The structures are displacing exactly how they should, in the direction of the force. Also all of the fixpoints remain fixed, which means every index is exactly where it should be.

This is the end of my technical solution to the problem of topology optimisation. I will now thoroughly test my application and then evaluate potential improvements and changes.

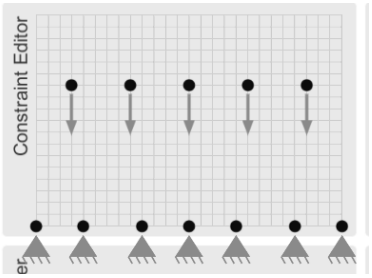
Testing

Throughout the testing section I am going to use specific presets to test my system. So if I refer to any of the following, look back here to find what the constraint set-up looks like:

Half MBB Beam

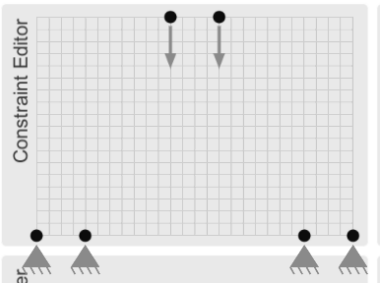


Bridge

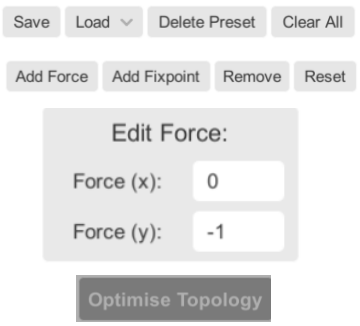
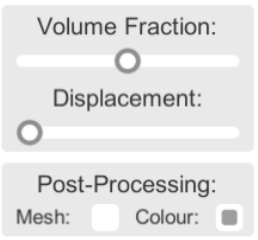


Cantilever

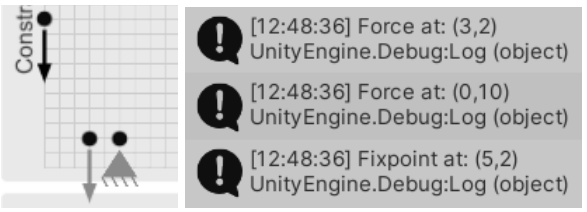
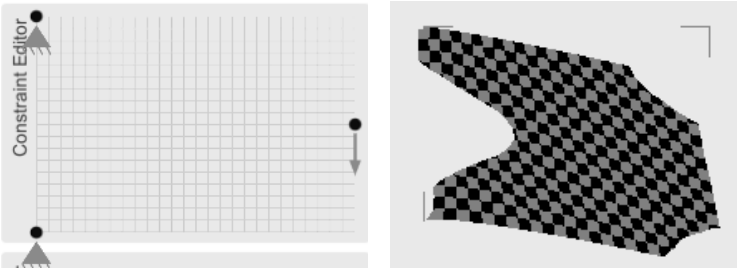
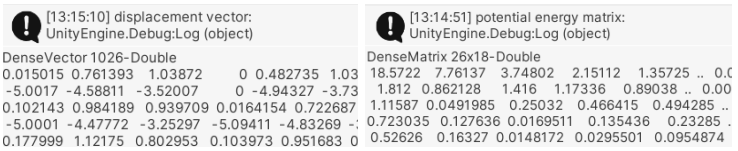
A Frame



Objective 1 - General

Obj	Criteria	Test Data	Evidence	Pass
1.1	Should run on windows as an exe file	The program build	 The build is runnable as seen in the screenshot above	
1.2	The code should optimise in < 60 seconds	Half MBB preset	 I used a Google timer whilst optimising a half MBB beam, Repeated three times for consistency	
1.3	Inputs should be validated	After the GUI objective tests, I am going to add an “error handling” section in which I will deliberately try to find a fault in my error catching. This will stand as evidence for data validation.		
1.4	Inputs should be intuitive and clearly labelled	None	  In my opinion the inputs are all clearly labelled	

Objective 2 - FEA

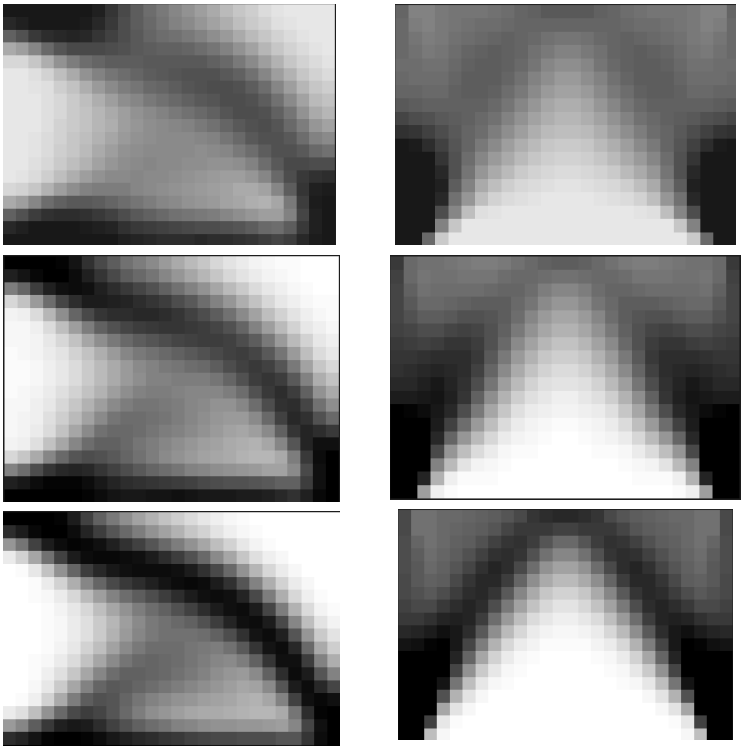
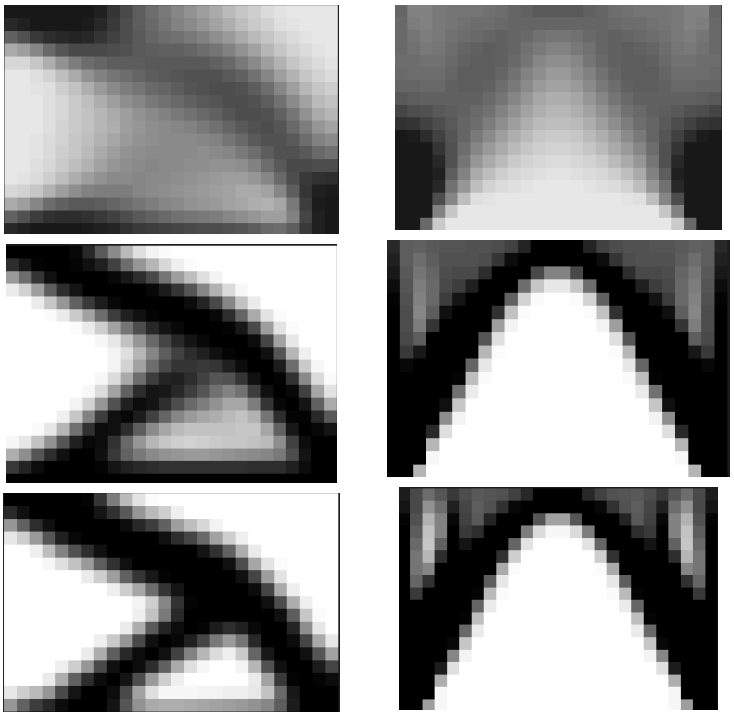
Obj	Criteria	Test Data	Evidence	Pass
2.1	The program should be able to fetch coordinates of constraints	Constraints placed at (3,2) (0, 10) and (5,2)		
2.3 2.3.1	.My UI should have a way of showing the displacement of the structure	Cantilever preset		
2.4 2.5	My FEA code should return a disp. Vector and a matrix containing potential energies	Bridge preset	 The zero values in the displacement matrix are all in the places of the fixpoints, therefore the result is valid	

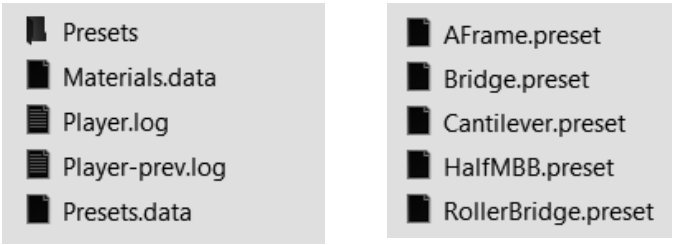
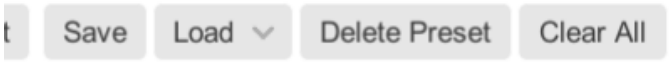
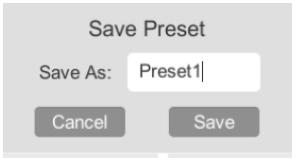
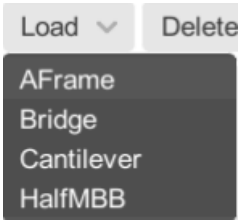
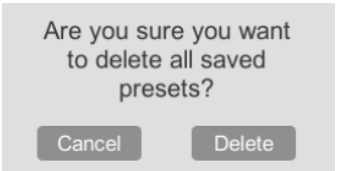
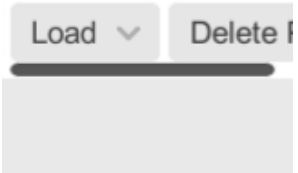
Objective 3 - Topology Optimisation

Obj	Criteria	Test Data	Evidence		Pass

3.1	The program should be able to receive FEA results	None	The FEA functions are in the same class as the topology optimisation functions and data is passed through public attributes, this code shows how the data flows:	`F = GetForceVector();`
				`GSM = GetGlobalMatrix();`
				`ReduceEquation();`
				`U = GSM.Solve(F);`
				`ExpandResult();`
				`p = GetPotentials();`
				`CriteriaUpdate();`	

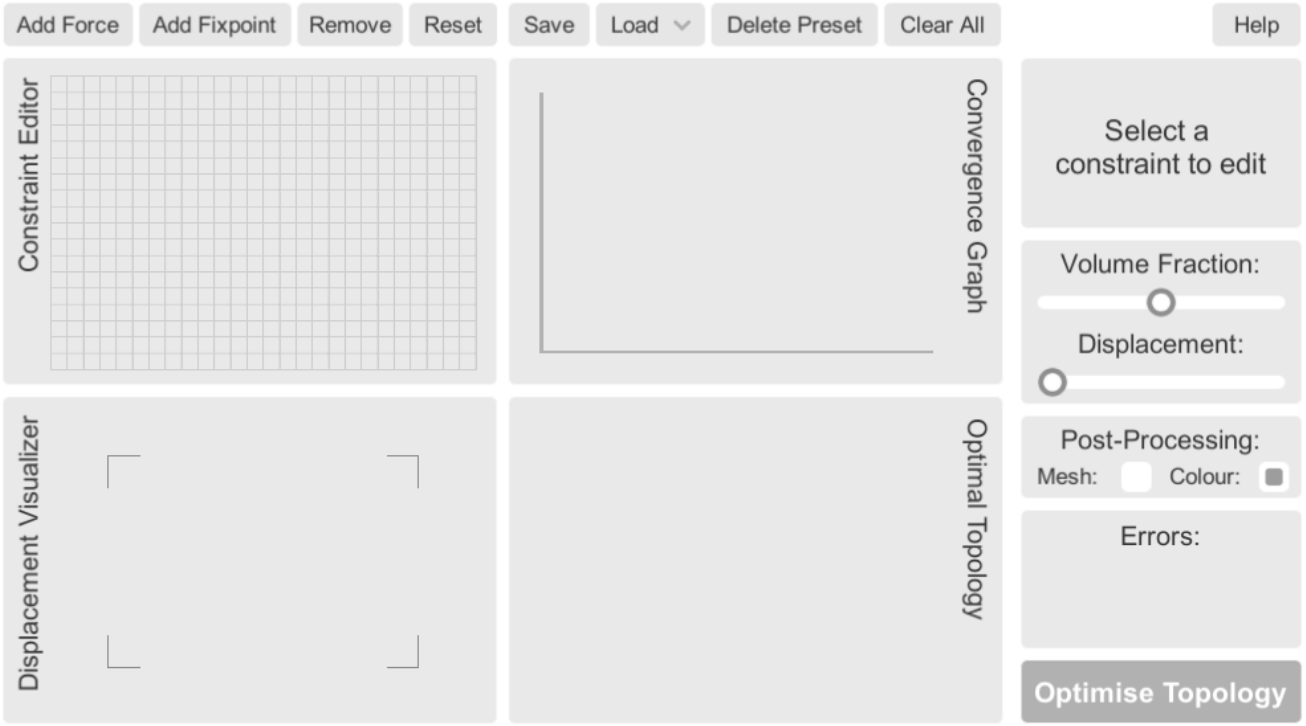
3.2	To use the bisection algorithm to match the volume requirement.	Half MBB with 0.5 volume constraint	I used the convergence graph to plot the volume of the current iteration and compare it to the required volume (the straight line) As seen it uses bisection to find the correct value for lambda.		

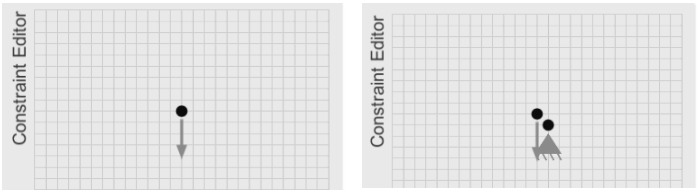
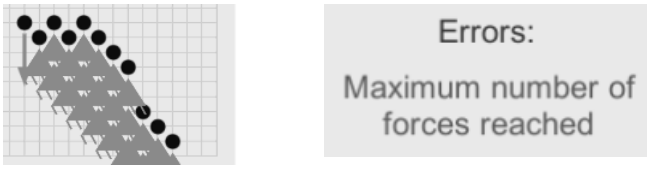
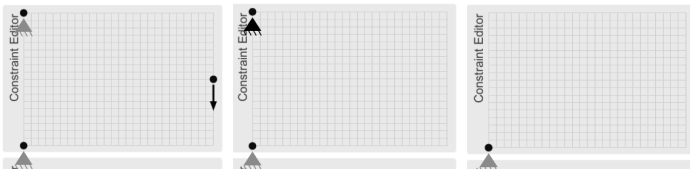
Obj	Criteria	Test Data	Evidence	Pass
3.3	After each iteration, the code should manipulate the density ready for the next.	Half MBB Bridge	 Three consecutive iterations show change in density is based from the iteration before.	
3.4	The process should be able to recognise when the structure is optimised and stop.	Half MBB Bridge	 Two sample iterations followed by the final iteration show the optimisation stops when the structure is converged	

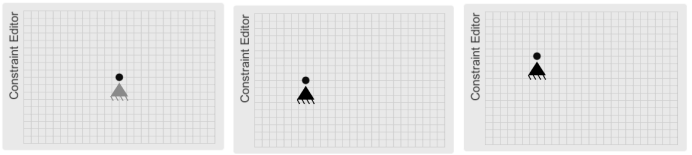
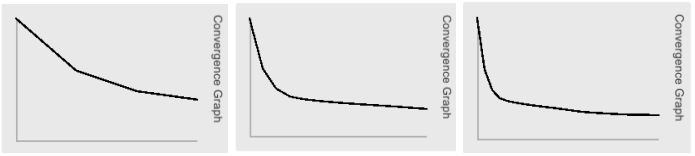
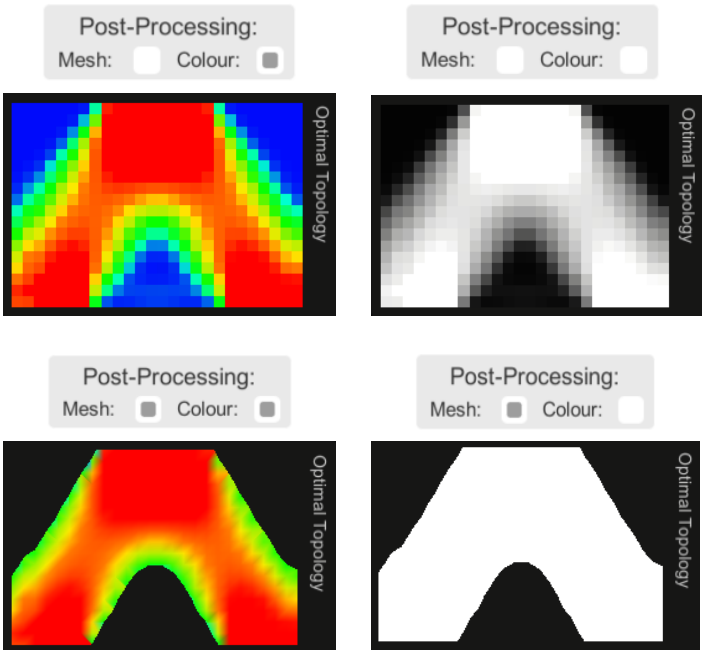
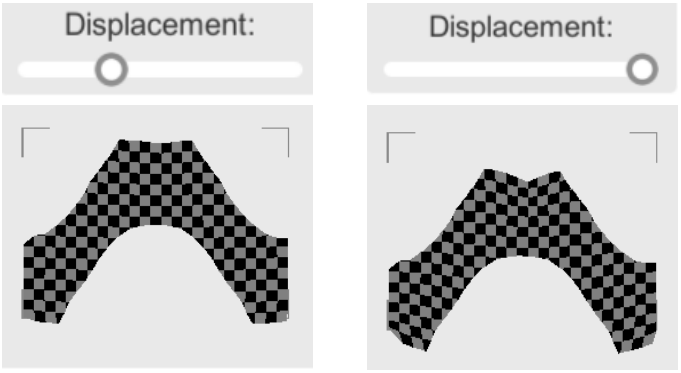
Objective 4 - Binary Files				
Obj	Criteria	Test Data	Evidence	Pass
4.1	The program should be able to store binary files containing condition presets	All current presets in their file location		
4.2	There should be a group of buttons at the top of the screen	None	 I added a button called 'Delete Preset' which allows the user to delete a single preset instead of needing to clear them all from the system.	
4.2.1	A save pop up appears for the name	None	 This pop up appears when the user clicks the 'Save' button, it also closes when either 'Save' or 'Close' is clicked	
	The user cant enter chars that aren't alphanumeric or names >20 characters	Trial 1: 'a_b#c 1*2&3' Trial 2: 'iiiiiiiiiiiiiiiiiiii' (21 'i's)	 When attempted, characters won't type  After 20, the user can't type any more characters	
	The user can't save presets with duplicate names	Trial 1: 'Duplicate' Trial 2: 'Duplicate'		
4.2.2	'Load' opens dropdown for the user to select a preset from a list	None	 Load functionality was tested in the preset part implementation section. There are also no errors to test for here, as it is a drop down list, and is already validated.	
4.2.3	'Clear Presets' deletes all of the user's saved presets	None	 	

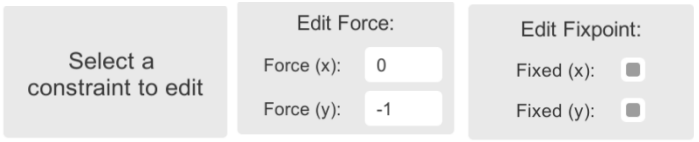
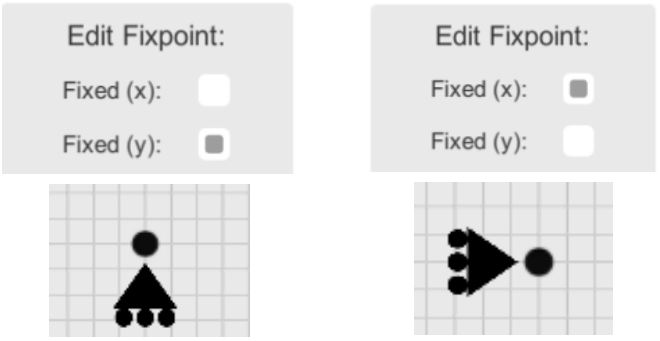
Objective 5 - User Interface

Below is an overall view of my user interface, so it's clear where each screen grab is taken from.



Obj	Criteria	Test Data	Evidence	Pass
5.1	The screen is divided into four sections	None	See GUI above	
5.1.1.1	Ability to add constraints	'Add force' and 'Add Fixpoint'		
	Stops the user from adding > 20 constraints	'Add Fixpoint' pressed until error thrown		
	Ability to remove selected icon	Remove each icon from cantilever		
5.1.1.2	Correct icons for constraints	Force & Fixpoint	As seen throughout, the icons are correct in my GUI for fixpoints, rollers and forces.	

Obj	Criteria	Test Data	Evidence	Pass
5.1.1.3	Ability to move icons	'Add Fixpoint', drag left, up arrow press		
	Icon can't be dragged off the grid	3 attempts at dragging and using arrow keys to move icon off grid		
5.1.2.1 5.1.2.2	The user should be able to see the stress plotted after each step on interpolated graph axis	HalfMBB	 As seen the graph is interpolated as the x and y stretch to fit the largest values in its axis. The graph data looks correct as the structure is improving.	
5.1.3.1 5.1.3.2 5.3 5.3.1 5.3.1.1 5.3.1.2	A window should show the final topology as a mesh or texture with or without colour (controlled by post process panel)	A Frame All possible toggle combinations in the post processing panel		
5.1.4.1	There will be a slider which changes the displacement visualiser	A Frame		

Obj	Criteria	Test Data	Evidence	Pass
5.2	There will be a vertical input section to the right	None	See GUI, there is an input panel on the right hand side of the screen.	
5.2.1	There will be a box for changing the force and fixpoints	Have nothing selected, then select a force, then a fixpoint		
	Changing fixpoint axis changes icon	Horizontal roller Vertical roller		
	Fixpoints must have a fixed axis	A fixpoint with no axis ticked	 And the code won't run	
	Forces must not be zero	A (0, 0) force	 And the code won't run	
5.2.3	There should be a slider for volume and a section for other outputs	None		

Evaluation

Objective Evaluation

Objective	Test Results	Comments / Improvements
1.1. The system should run on windows as an executable file.	The file is runnable and easy to distribute to people who wish to run the app on windows.	If I had more time, perhaps I could have made a web or mac build so more people could use the software.
1.2. The code should optimise a structure in under a minute.	My code optimises a structure in under half a minute so this objective has been met.	I could optimise this code further by implementing a shader which would run the matrix calculations for me.
1.3. Inputs should all be validated before processing.	Throughout testing I ran multiple error tests and all were successfully caught.	Error testing was successful, I don't think there are any improvements to be made.
1.4 Outputs should be intuitive and clearly labelled.	Outputs are divided into windows and each has a title labelling what that section is.	My application has a help menu even if the UI isn't clear. The user needs to know what TO is before using though, so an introduction could be a good feature.
2. To be able to perform Finite Element Analysis on a structure	The test results throughout the implementation and testing show the code can find where a structure is most stressed	The only improvement I would make to the finite element section of my code is the speed of the 'solve' function in math.NET, if I had the time, I would find the source code and rewrite it in shader language so it ran on the GPU.
3. To be able to perform Topology Optimisation on a structure	The test results show that the app can successfully find the best structure based on the boundary conditions	I think it would be good to compare the physical strengths of these structures by 3D printing them and comparing them to others, that way it's a fair test to find out if it is truly the best structure.
4. Saving, loading and deleting presets	All the preset tests ran fine, and all the validation passed.	Currently, the user is able to save a preset with invalid constraints in it, if I were to continue working on this project I would make it so that wasn't possible. The feature I added where the user can delete a single preset is also a good improvement to my apps functionality
5.1.1. The constraint editor	All tests passed, including input validation, therefore the user can't break the system	The constraint editor does everything it should. I think it would be good to add more keyboard shortcuts to make inputs quicker and more efficient for the user.

Objective	Test Results	Comments / Improvements
5.1.2. The convergence graph	The graph can successfully plot a range of values due to the interpolation	I would add more labels to the graph and potentially data points. Also, if the gradient is steep, the line width doesn't stay consistent, so I would amend the line generator component to fix this.
5.1.3. Optimised Topology	This window successfully displays the final structure in four different ways based off the post processing inputs	When switching between the model and the mesh, the structure isn't perfectly aligned, it's a minor and easy fix. Also, the colour gradient on the mesh could be interpolated to show the full gradient.
5.1.4. Displacement Visualisation	The model displaces as expected, and all fixpoints remain in place	The only issue is the fact that the mesh is one row and column shorter than the structure, so the displacements on the edge of the structure are excluded. However, I don't think this is fixable due to the way the marching squares algorithm works.
5.2. The input section	Everything functions correctly and the error messages pop up as expected	The added feature of rollers (fixpoints only in one dimension) add complexity to the structures the user can create, and leads to a more accurate representation of the problem they need to solve.
5.3. Post-processing	The two toggles control the display of the model and the display updates each time a toggle is changed.	This panel is fairly simplistic, and there isn't any error handling to consider as all possible inputs are valid.

End-User Feedback

Positive Feedback

"First, looking at your software, I think it looks smooth and aesthetically pleasing. The layout gives me the impression that everything belongs in a place, and the layout is efficiently designed.

The inputs are all labelled and there's nothing I feel like I wouldn't know how to use. Whenever I'm not sure about a feature, I just open the help menu, and the explanations in there are very helpful. To a new user I feel that they could quickly get a grasp of how to use the software.

The side panel is well organised and I like how clear the errors are when they pop up informing the user of a problem. The positioning of the error box is well designed as well, as it's right next to the optimise button.

I found dragging the icons in the constraint editor made it quite difficult to get it to go exactly where I wanted them to go, so your keyboard shortcuts with the arrow keys help with this.

Overall, I think the UI is well designed, and the code does exactly what it should."

Constructive Feedback

“There are only a few things I could improve with this application, and they are only minor.

Firstly, I think for the two sliders, there should be a better way of entering an exact value, like a label telling me where the slider value is, or even an input field so i can input the exact value for the slider. Dragging and guessing where the slider is means I can't use exact values which are needed for calculating material costs.

The only thing I need to say about the layout of the UI is the titles for each section are sideways. I can read them fine, and I appreciate the screen size makes it hard to organise, but I feel like all text should be oriented upright.

When editing the constraints, if an icon is placed on an edge, the image overlaps the title and sometimes covers the displacement editor. Maybe you could solve this problem with a scalable region to edit the structure in, so they don't interfere with any other section.

Overall, I feel like these are only minor fixes and they have little to no effect on the overall functionality of the system. The final thing is just to make it run faster, but that could be said about almost any system which needs a large amount of processing.”

All Improvements

- Align the mesh and texture in the optimised model display
- Stop user saving empty or invalid presets
- Add an introduction explaining what the app’s purpose is
- Use a full colour gradient for the mesh in the optimised display
- Add more labels to the convergence graph and numerical data
- Make the graph’s line a consistent width
- Build the app for a variety of platforms
- Optimise code using GPU so it optimises faster and I can have a higher resolution
- Having a higher resolution model so the mesh is more accurate
- Make areas be affected by constraints (fixed or have applied force)
- Keyboard shortcuts on the constraints
- Ability to select multiple constraints at the same time
- An export feature allowing the user to do more with the generated mesh.

Conclusion

I am very happy with my final UI and code. I feel like I have managed to successfully implement plenty of UI features and displays for the given time frame, and to have performed topology optimisation is a big accomplishment to me. I am pleased with the time frame my code optimises structures in, however, this is because it is a low resolution structure, and I would like to increase this at some point, whilst still optimising in under a minute.

The uses this application could provide for Boris going forward with his civil engineering career could be endless, if he is tasked with designing a structure or part of any construction, he can refer back to this app and find the most cost effective shape. After hearing his feedback, he is pleased with the outcome of this project and is happy that he has access to ‘such a useful tool’.

Of course, there are plenty of improvements it could make and little features that I could add to enhance the performance of this app, so I have plenty to do if I were to continue working on this software. I could even create an app to optimise structures in 3D as well as 2D, which would extend the uses of this program as it’s even closer to a real world simulation.

Appendix

Code

PresetSerializer

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using System.Runtime.Serialization.Formatters.Binary;
using System.IO;

public static class PresetSerializer
{
    public static string GetDirectoryPath() { return Application.persistentDataPath + "/Presets"; }
    public static string GetFilePath(string name) { return GetDirectoryPath() + "/" + name + ".preset"; }
    public static bool SavePreset(string name)
    {
        if(!Directory.Exists(GetDirectoryPath())) { Directory.CreateDirectory(GetDirectoryPath()); }

        if (!File.Exists(GetFilePath(name)))
        {
            Preset currentPreset = new Preset(); // create new preset
            BinaryFormatter formatter = new BinaryFormatter(); // initialise a formatter
            FileStream stream = new FileStream(GetFilePath(name), FileMode.Create); // create new file
            formatter.Serialize(stream, currentPreset); // convert to binary
            stream.Close(); // close file stream

            return true;
        }
        else { ErrorText.ShowError("Name already exists"); return false; }
    }
    public static Preset LoadPreset(string name)
    {
        BinaryFormatter formatter = new BinaryFormatter();
        FileStream stream = new FileStream(GetFilePath(name), FileMode.Open);
        Preset data = formatter.Deserialize(stream) as Preset;
        stream.Close();
        return data;
    }
    public static void DeletePreset(string name) { File.Delete(GetFilePath(name)); }
}

[System.Serializable]
public class Preset
{
    [SerializeField] public List<(float, float, bool, bool)> _fixpoints;
    [SerializeField] public List<(float, float, float, float)> _forces;
    public Preset()
    {
        _fixpoints = new List<(float, float, bool, bool)>();
        _forces = new List<(float, float, float, float)>();
        foreach (GameObject n in InputManager.fixpoints)
        {
            _fixpoints.Add((n.GetComponent<RectTransform>().anchoredPosition.x, n.GetComponent<RectTransform>().anchoredPosition.y,
                            n.GetComponentInChildren<FixpointManager>().x, n.GetComponentInChildren<FixpointManager>().y));
        }
        foreach (GameObject n in InputManager.forces)
        {
            _forces.Add((n.GetComponent<RectTransform>().anchoredPosition.x, n.GetComponent<RectTransform>().anchoredPosition.y,
                         n.GetComponentInChildren<ForceManager>().force.x, n.GetComponentInChildren<ForceManager>().force.y));
        }
    }
}
```

MarchingSquares

```
using System.Collections;
using System.Collections.Generic;
using static System.Math;
using UnityEngine;
using UnityEngine.UI;
using MathNet.Numerics;
using MathNet.Numerics.Data.Text;
using MathNet.Numerics.LinearAlgebra;
using MathNet.Numerics.LinearAlgebra.Double;
using MathNet.Numerics.LinearAlgebra.Double.Solvers;
using MathNet.Numerics.LinearAlgebra.Factorization;

public class MarchingSquares : MonoBehaviour
{
    #region Variables
    public int xSize;
    public int ySize;
    public Vector<double> U;
    public Gradient gradient;
    public Slider dispExg;

    [HideInInspector] // holds the indices of vertices for each triangle
    private static List<int[]> triTable = new List<int[]>() {
        new int[] { },                          //0000
        new int[] { 7, 6, 3 },                  //0001
        new int[] { 6, 5, 2 },                  //0010
        new int[] { 3, 7, 5, 3, 5, 2 },         //0011
        new int[] { 7, 0, 4 },                  //0100
        new int[] { 3, 0, 4, 3, 4, 6 },         //0101
        new int[] { 7, 0, 4, 2, 6, 5 },         //0110
        new int[] { 3, 0, 4, 3, 4, 5, 3, 5, 2 }, //0111
        new int[] { 5, 4, 1 },                  //1000
        new int[] { 7, 6, 3, 5, 4, 1 },         //1001
        new int[] { 6, 4, 2, 2, 4, 1 },         //1010
        new int[] { 3, 7, 2, 2, 7, 4, 2, 4, 1 }, //1011
        new int[] { 7, 0, 1, 7, 1, 5 },         //1100
        new int[] { 0, 1, 5, 0, 5, 6, 0, 6, 3 }, //1101
        new int[] { 1, 2, 6, 1, 6, 7, 1, 7, 0 }, //1110
        new int[] { 0, 1, 3, 1, 2, 3 } };       //1111
    #endregion

    public void Render(Matrix<double> p, bool isColored)
    {
        if (U != null)
        {
            double max = 0;
            for (int i = 0; i < U.Count; i++) { if (Abs(U[i]) > max) { max = Abs(U[i]); } }
            for (int i = 0; i < U.Count; i++) { U[i] /= max; }
            U *= dispExg.value;
        }
        CombineInstance[] combine = new CombineInstance[(xSize) * (ySize)];
        int meshIndex = 0;
        for (int i = 0; i < xSize; i++)
        {
            for (int j = 0; j < ySize; j++)
            {
                combine[meshIndex].mesh = GetElementMesh(i, j, p);
                combine[meshIndex].transform = Matrix4x4.Translate(new Vector3(i + 0.5f, j + 0.5f, 0f));
                meshIndex++;
            }
        }
        GetComponent<MeshFilter>().mesh = new Mesh();
        GetComponent<MeshFilter>().mesh.CombineMeshes(combine);
        if (U == null && isColored)
        {
            Mesh chunkMesh = GetComponent<MeshFilter>().mesh;
            Color[] colors;
            colors = new Color[chunkMesh.vertices.Length];
            for (int i = 0; i < colors.Length; i++)
            {
                colors[i] = gradient.Evaluate((float)p[(int)chunkMesh.vertices[i].x, (int)chunkMesh.vertices[i].y]);
            }
            chunkMesh.colors = colors;
        }
    }
    public Mesh GetElementMesh(int x, int y, Matrix<double> p)
    {
        bool[] arr = new bool[4];
        for (int i = 0; i < 2; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                if (p[x+i, y+j] >= 0.5f) { arr[i+2*j] = true; }
                else { arr[i+2*j] = false; }
```

```
            }
        }

        Vector3[] relativePoints = new Vector3[8]
        { new Vector3 (-0.5f, 0.5f, 0f), new Vector3 ( 0.5f, 0.5f, 0f), // vertices
          new Vector3 ( 0.5f,-0.5f, 0f), new Vector3 (-0.5f,-0.5f, 0f), //
          new Vector3 (0f, 0.5f, 0f),    new Vector3 ( 0.5f, 0f, 0f),   // midpoints
          new Vector3 (0f,-0.5f, 0f),    new Vector3 (-0.5f, 0f, 0f)};  //

        if ((arr[0] || arr[1] || arr[2] || arr[3]) && !(arr[0] && arr[1] && arr[2] && arr[3]))
        {
            relativePoints[4] = new Vector3(Mathf.InverseLerp((float)p[x, y + 1], (float)p[x + 1, y + 1], 0.5f) - 0.5f, 0.5f, 0f);
            relativePoints[5] = new Vector3(0.5f, Mathf.InverseLerp((float)p[x + 1, y], (float)p[x + 1, y + 1], 0.5f) - 0.5f, 0f);
            relativePoints[6] = new Vector3(Mathf.InverseLerp((float)p[x, y], (float)p[x + 1, y], 0.5f) - 0.5f, -0.5f, 0f);
            relativePoints[7] = new Vector3(-0.5f, Mathf.InverseLerp((float)p[x, y], (float)p[x, y + 1], 0.5f) - 0.5f, 0f);
        }

        if (U != null)
        {
            List<float> u = eDisps(x, y);
            relativePoints[0] += new Vector3(u[0], u[1], 0f);
            relativePoints[1] += new Vector3(u[2], u[3], 0f);
            relativePoints[2] += new Vector3(u[4], u[5], 0f);
            relativePoints[3] += new Vector3(u[6], u[7], 0f);

            //interpolate
            relativePoints[4] += new Vector3(((u[0] + u[2]) / 2), ((u[1] + u[3]) / 2), 0f);
            relativePoints[5] += new Vector3(((u[2] + u[4]) / 2), ((u[3] + u[5]) / 2), 0f);
            relativePoints[6] += new Vector3(((u[4] + u[6]) / 2), ((u[5] + u[7]) / 2), 0f);
            relativePoints[7] += new Vector3(((u[6] + u[0]) / 2), ((u[7] + u[1]) / 2), 0f);
        }

        BitArray bitArray = new BitArray(arr);
        int[] array = new int[1];
        bitArray.CopyTo(array, 0);

        Mesh voxelMesh = new Mesh();
        voxelMesh.vertices = relativePoints;
        voxelMesh.triangles = triTable[array[0]];

        if (U != null)
        {
            Color[] colors;
            colors = new Color[voxelMesh.vertices.Length];
            for (int i = 0; i < colors.Length; i++)
            {
                if (((x + y) % 2) == 0) { colors[i] = new Color(0.6509434f, 0.4485452f, 0f, 1f); }
                else { colors[i] = new Color(0.4528302f, 0.3109798f, 0f, 1f); }
            }
            voxelMesh.colors = colors;
        }

        return voxelMesh;
    }

    List<float> eDisps(int x, int y)
    {
        int n1 = 2 * ((ySize + 2) * x + 1 + ySize - y);
        int n2 = 2 * ((ySize + 2) * (x + 1) + 1 + ySize - y);
        int[] e = new int[] { n1 - 2, n1 - 1, n2 - 2, n2 - 1, n2, n2 + 1, n1, n1 + 1 };

        List<float> l = new List<float>();
        foreach (int i in e) { l.Add((float)U[i]); }
        return l;
    }
}
```

ForceManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ForceManager : MonoBehaviour
{
    public Vector2 force;
    public bool isPresetInstance;
    public bool isValid;
    public Image arrow;

    public Color selectedColour;
    public Color errorColour;
    public Color normalColour;

    void Start() { if (!isPresetInstance) { force = new Vector2(0, -1); } }

    public void Update()
    {
        if (force == Vector2.zero)
        {
            isValid = false;
            ErrorText.ShowError("Magnitude of force cannot be zero");
        }
        else
        {
            isValid = true;
            float angle = Mathf.Atan(force.y / force.x) * Mathf.Rad2Deg;
            if (force.x < 0) { angle += 180; }
            transform.parent.transform.rotation = Quaternion.Euler(0, 0, angle);
        }

        arrow.enabled = isValid;

        if (!isValid) { GetComponent<Image>().color = errorColour; }
        else { GetComponent<Image>().color = selectedColour; }

        if (transform.parent.gameObject == InputManager.activeObject) { arrow.color = selectedColour; }
        else { arrow.color = normalColour; }
    }
}
```

GraphGenerator

```
using System.Linq;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GraphGenerator : MonoBehaviour
{
    private List<float> points;

    void Start() { points = new List<float>(); }

    public void Clear() { GetComponent<LineRenderer>().positionCount = 0; points = new List<float>(); }

    public void AddPoint(double value)
    {
        points.Add((float)value);
        float max = points.Max();
        Vector3[] positions = new Vector3[points.Count];
        if (points.Count > 1)
        {
            for (int i = 0; i < points.Count; i++)
            {
                float xPos = (float)(20 + 240 * i / (points.Count - 1));
                float yPos = (float)(20 + 160 * Mathf.InverseLerp(0, max, points[i]));
                positions[i] = new Vector3(xPos, yPos, 0);
            }
            GetComponent<LineRenderer>().positionCount = points.Count;
            GetComponent<LineRenderer>().SetPositions(positions);
        }
    }
}
```

FixpointManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class FixpointManager : MonoBehaviour
{
    public bool x;
    public bool y;
    public bool isPresetInstance;
    public bool isValid;
    public Image roller;
    public Image fix;

    public Color selectedColour;
    public Color errorColour;
    public Color normalColour;

    void Start() { if (!isPresetInstance) { x = true; y = true; } }

    public void Update()
    {
        if (!x && !y)
        {
            ErrorText.ShowError("Fixpoints must have at least one dimension fixed");
            roller.GetComponent<Image>().enabled = false;
            fix.GetComponent<Image>().enabled = false;
            isValid = false;
        }
        else
        {
            isValid = true;
            if (x && y)
            {
                transform.parent.transform.rotation = Quaternion.Euler(0, 0, 90);
                roller.GetComponent<Image>().enabled = false;
                fix.GetComponent<Image>().enabled = true;
            }
            else if (x)
            {
                transform.parent.transform.rotation = Quaternion.Euler(0, 0, 0);
                roller.GetComponent<Image>().enabled = true;
                fix.GetComponent<Image>().enabled = false;
            }
            else if (y)
            {
                transform.parent.transform.rotation = Quaternion.Euler(0, 0, 90);
                roller.GetComponent<Image>().enabled = true;
                fix.GetComponent<Image>().enabled = false;
            }
        }

        if (!isValid) { GetComponent<Image>().color = errorColour; }
        else { GetComponent<Image>().color = selectedColour; }

        if (transform.parent.gameObject == InputManager.activeObject) { fix.color = roller.color = selectedColour; }
        else { fix.color = roller.color = normalColour; }
    }
}
```

PresetManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;

public class PresetManager : MonoBehaviour
{
    public Dropdown loadMenu;
    public Dropdown deleteMenu;
    public GameObject inputBarrier;
    public GameObject savePopUp;
    public InputManager inputManager;

    public void Awake() { UpdateLoadMenu(); }

    public void Save(InputField input)
    {
        if (input.text == "") { ErrorText.ShowError("Preset name cannot be empty"); }
        else if (PresetSerializer.SavePreset(input.text))
```

```
        {
            inputBarrier.SetActive(false);
            savePopUp.SetActive(false);
        }

        UpdateLoadMenu();
    }

    public void Load(Text name)
    {
        inputManager.Reset();
        Preset preset = PresetSerializer.LoadPreset(name.text);
        foreach ((float, float, bool, bool) data in preset._fixpoints)
        {
            inputManager.AddFixpointFromPreset(data);
        }
        foreach((float, float, float, float) data in preset._forces)
        {
            inputManager.AddForceFromPreset(data);
        }
    }

    public void Delete(Dropdown menu)
    {
        PresetSerializer.DeletePreset(menu.options[menu.value].text);
        UpdateLoadMenu();
    }

    public void DeleteAll()
    {
        if (Directory.Exists(PresetSerializer.GetDirectoryPath()))
        {
            Directory.Delete(PresetSerializer.GetDirectoryPath(), true);
        }
        Directory.CreateDirectory(PresetSerializer.GetDirectoryPath());

        UpdateLoadMenu();
    }

    public void UpdateLoadMenu()
    {
        DirectoryInfo info = new DirectoryInfo(PresetSerializer.GetDirectoryPath());
        var fileInfo = info.GetFiles();

        loadMenu.ClearOptions();

        foreach (var file in fileInfo)
        {
            Dropdown.OptionData option = new Dropdown.OptionData();
            option.text = Path.GetFileName(file.ToString()).Replace(".preset", "");
            loadMenu.options.Add(option);
        }
    }

    public void UpdateDeleteMenu()
    {
        DirectoryInfo info = new DirectoryInfo(PresetSerializer.GetDirectoryPath());
        var fileInfo = info.GetFiles();

        deleteMenu.ClearOptions();

        foreach (var file in fileInfo)
        {
            Dropdown.OptionData option = new Dropdown.OptionData();
            option.text = Path.GetFileName(file.ToString()).Replace(".preset", "");
            deleteMenu.options.Add(option);
        }
        deleteMenu.value = 0;
    }
}
```

InputManager

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class InputManager : MonoBehaviour
{
    #region Attributes
    public static GameObject activeObject;
    public static GameObject temp;

    [HideInInspector] public static List<GameObject> forces = new List<GameObject>();
```

```
    [HideInInspector] public static List<GameObject> fixpoints = new List<GameObject>();

    public GameObject force;
    public GameObject fixpoint;

    public GameObject fixOption;
    public GameObject forceOption;
    public GameObject noneSelected;

    public InputField xForce;
    public InputField yForce;
    public Toggle xFix;
    public Toggle yFix;

    public Button removeButton;
    #endregion


    void Update()
    {
        if (Optimiser.optimiseStatus == "idle") // before optimisation process
        {
            foreach (GameObject f in forces) { f.GetComponentInChildren<Draggable>().enabled = true; }
            foreach (GameObject f in fixpoints) { f.GetComponentInChildren<Draggable>().enabled = true; }
        }

        if (activeObject != temp)
        {
            if (forces.Contains(activeObject))
            {
                forceOption.SetActive(true);
                fixOption.SetActive(false);

                xForce.text = activeObject.GetComponentInChildren<ForceManager>().force.x.ToString();
                yForce.text = activeObject.GetComponentInChildren<ForceManager>().force.y.ToString();
            }
            else if (fixpoints.Contains(activeObject))
            {
                fixOption.SetActive(true);
                forceOption.SetActive(false);

                bool xTemp = activeObject.GetComponentInChildren<FixpointManager>().x;
                bool yTemp = activeObject.GetComponentInChildren<FixpointManager>().y;
                xFix.isOn = xTemp;
                yFix.isOn = yTemp;
                activeObject.GetComponentInChildren<FixpointManager>().x = xTemp;
                activeObject.GetComponentInChildren<FixpointManager>().y = yTemp;
            }
        }

        if (activeObject == null)
        {
            fixOption.SetActive(false);
            forceOption.SetActive(false);
            removeButton.interactable = false;
            noneSelected.SetActive(true);
        }
        else { removeButton.interactable = true; noneSelected.SetActive(false); }

        temp = activeObject;
    }


    #region Adding constraints
    // adds icons with an offset
    public Vector2 GetNewIconPos()
    {
        bool found;
        for (int j = 0; j <= 4; j += 2)
        {
            for (Vector2 i = new Vector2(13 + j, 9); i.y > 0; i += new Vector2(1, -1))
            {
                found = true;
                foreach (GameObject n in forces) { if (n.GetComponent<RectTransform>().anchoredPosition == i) { found = false; } }
                foreach (GameObject n in fixpoints) { if (n.GetComponent<RectTransform>().anchoredPosition == i) { found = false; } }
                if (found) { return i; }
            }
        }
        return Vector2.zero;
    }


    // adds default force when 'add force' is pressed
    public void AddForce()
    {
        if (forces.Count <= 5)
        {
```

```
            GameObject n = Instantiate(force, Vector3.zero, Quaternion.Euler(0, 0, -90));
            n.GetComponent<RectTransform>().anchoredPosition = GetNewIconPos();
            n.transform.SetParent(transform, false);
            forces.Add(n);
        }
        else { ErrorText.ShowError("Maximum number of forces reached"); }
    }


    // adds default fixpoint when 'add fixpoint' is pressed
    public void AddFixpoint()
    {
        if (fixpoints.Count <= 20)
        {
            GameObject n = Instantiate(fixpoint, Vector3.zero, Quaternion.Euler(0, 0, 90));
            n.GetComponent<RectTransform>().anchoredPosition = GetNewIconPos();
            n.transform.SetParent(transform, false);
            fixpoints.Add(n);
        }
        else { ErrorText.ShowError("Maximum number of forces reached"); }
    }


    // adds force to the editor using conditions from the preset
    public void AddForceFromPreset((float, float, float, float) data)
    {
        GameObject n = Instantiate(force, Vector3.zero, Quaternion.Euler(0, 0, -90));
        n.GetComponentInChildren<ForceManager>().isPresetInstance = true;
        n.GetComponent<RectTransform>().anchoredPosition = new Vector2(data.Item1, data.Item2);
        n.GetComponentInChildren<ForceManager>().force = new Vector2(data.Item3, data.Item4);
        n.transform.SetParent(transform, false);
        forces.Add(n);
    }


    // adds fixpoint to the editor using conditions from the preset
    public void AddFixpointFromPreset((float, float, bool, bool) data)
    {
        GameObject n = Instantiate(fixpoint, Vector3.zero, Quaternion.Euler(0, 0, 90));
        n.GetComponentInChildren<FixpointManager>().isPresetInstance = true;
        n.GetComponent<RectTransform>().anchoredPosition = new Vector2(data.Item1, data.Item2);
        n.GetComponentInChildren<FixpointManager>().x = data.Item3;
        n.GetComponentInChildren<FixpointManager>().y = data.Item4;
        n.transform.SetParent(transform, false);
        fixpoints.Add(n);
    }
    #endregion


    #region Removing constraints
    // removes active constraint from domain
    public void Remove()
    {
        fixpoints.Remove(activeObject);
        forces.Remove(activeObject);
        Destroy(activeObject);
    }

    // removes all constraints from domain
    public void Reset()
    {
        foreach (GameObject n in forces) { Destroy(n); }
        foreach (GameObject n in fixpoints) { Destroy(n); }
        forces = new List<GameObject>();
        fixpoints = new List<GameObject>();
    }
    #endregion


    #region Setting inputs
    // called when force values change in UI
    public void SetForceInputs()
    {
        float x = float.Parse(xForce.text);
        float y = float.Parse(yForce.text);
        activeObject.GetComponentInChildren<ForceManager>().force = new Vector2(x, y);
    }


    // called when fixpoint toggles change in UI
    public void SetFixInputs()
    {
        activeObject.GetComponentInChildren<FixpointManager>().x = xFix.isOn;
        activeObject.GetComponentInChildren<FixpointManager>().y = yFix.isOn;
    }
    #endregion
}
```

Optimiser

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using static System.Math;
using UnityEngine;
using Unity.Collections;
using UnityEngine.UI;
using MathNet.Numerics;
using MathNet.Numerics.Data.Text;
using MathNet.Numerics.LinearAlgebra;
using MathNet.Numerics.LinearAlgebra.Double;
using MathNet.Numerics.LinearAlgebra.Double.Solvers;
using MathNet.Numerics.LinearAlgebra.Factorization;

public class Optimiser : MonoBehaviour
{
    #region Attributes
    public int xSize, ySize; // domain size
    public static string optimiseStatus; // optimisation stage
    public int iterations; // temp
    private int iteration; // temp

    public double penal; // penalisation factor
    public double rMin; // density filter radius
    public double pMin; // smallest density value
    public double tolerance; // smallest density value
    public double move; // move limit

    private Matrix<double> LSM; // relates element force to displacement
    private Matrix<double> GSM; // relates force to displacement
    private Matrix<double> p; // densities calculated from potential energy

    private Vector<double> U; // displacement vector
    private Vector<double> F; // force vector


    public RawImage topoTexture; // output image for texture
    public Toggle topoMesh, topoColor; // post processing controls
    public MarchingSquares topoMarcher; // marching cubes script
    public MarchingSquares dispMarcher; // marching cubes script

    public Slider volumeSlider; // user input slider for the volume fraction
    public Gradient gradient; // colour gradient to display with
    public GraphGenerator graph; // graph script to send values to

    public GameObject backToEdit;
    #endregion


    // encapsulation so UI components can change the status
    public void SetOptimiseStatus(string status)
    {
        optimiseStatus = status;
    }


    // called at the start of runtime
    void Start()
    {
        optimiseStatus = "idle";
        LSM = DenseMatrix.OfArray(new double[,] {
        { 0.494505494504,0.178571428574,-0.302197021972,-0.0137362633626,-0.247227472527,-0.178571428574,0.0549450594505,0.0133626373626},
        { 0.178571428575,0.494505494945,0.0137362636375,0.0549450549494,-0.178571428585,-0.247252747273,-0.01373626373675,-0.30219780222},
        { -0.30219780212, 0.01372637375, 0.494505494945, -0.178571428571, 0.05494505494, -0.01373626375, -0.247252747252, 0.178571428575},
        { -0.013736637365,0.0549450549454,-0.178571428585,0.494505054945,0.0137362626375,-0.302197808022,0.178571428485,-0.2472527527473},
        { -0.247252747253,-0.17857144285,0.0549450505494,0.0137363626375,0.494505054945,0.178571428285,-0.302197808022,-0.01373626378375},
        { -0.178571428585,-0.24722527473,-0.0137362637365,-0.302191978022,0.178571428585,0.49450549445,0.013736626375,0.0549450549498494},
        { 0.0549450545494,-0.0137362637375,-0.247252747252,0.178571428585,-0.302197802192,0.0137362637375,0.494505454945,-0.178571428285},
        { 0.013736263735,-0.3021978021022,0.178571714285,-0.247252747273,-0.013736263735,0.054945054494,-0.17857142855,0.4945054945045}});
    }


    // called every frame
    void Update()
    {

        if (optimiseStatus == "iterating")
        {
            if (iteration > 0)
            {
                graph.Clear(); // temp
                F = GetForceVector(); // assemble force vector
                GSM = GetGlobalMatrix(); // assemble GSM
                ReduceEquation(); // remove fixpoints
```

```
            U = GSM.Solve(F); // solve linear system
            ExpandResult(); // add fixpoints

            p = GetPotentials(); // set densities
            CriteriaUpdate(); // sensitivity analysis, filter, then match criteria

            Display();
            iteration--;
        }
        else { optimiseStatus = "optimised"; }
    }
    if (optimiseStatus == "optimised") { backToEdit.SetActive(true); }
}


#region FEA
private Vector<double> GetForceVector()
{
    Vector<double> f = new DenseVector((xSize + 1) * (ySize + 1) * 2);
    foreach (GameObject force in InputManager.forces)
    {
        Vector2 pos = force.GetComponent<RectTransform>().anchoredPosition;
        int n = ndof((int)pos.x, (int)pos.y);
        f[n] = force.GetComponentInChildren<ForceManager>().force.x;
        f[n + 1] = force.GetComponentInChildren<ForceManager>().force.y;
    }
    return f;
}


// assembles the global stiffness matrix
private Matrix<double> GetGlobalMatrix()
{
    int matrixSize = (xSize + 1) * (ySize + 1) * 2;
    Matrix<double> K = new DenseMatrix(matrixSize, matrixSize);

    // for each element
    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            int[] edof = edofs(i, j);
            int xInd = 0, yInd = 0;
            foreach (int y in edof)
            {
                foreach (int x in edof)
                {
                    // add the DOF into the GSM weighted by density
                    K[x, y] += p[i, j] * LSM[xInd, yInd];
                    xInd++;
                }
                yInd++;
                xInd = 0;
            }
        }
    }
    return K;
}


// remove fixed DOFs from force vector and GSM
private void ReduceEquation()
{
    // convert to matrix as vector doesn't support row/column removal
    Matrix<double> F_asColMat = F.ToColumnMatrix();
    List<int> toRemove = GetFixpointIndices();
    for (int i = toRemove.Count - 1; i >= 0; i--)
    {
        F_asColMat = F_asColMat.RemoveRow(toRemove[i]);
        GSM = GSM.RemoveRow(toRemove[i]).RemoveColumn(toRemove[i]);
    }
    F = new DenseVector(F_asColMat.ToColumnArrays()[0]);
}


// add zeros in place of the fixpoints
private void ExpandResult()
{
    Vector<double> zero = new DenseVector(1);
    List<double> U_asList = U.ToArray().ToList();
    List<int> toAdd = GetFixpointIndices();
    for (int i = 0; i < toAdd.Count; i++)
    {
        if (toAdd[i] >= U_asList.Count) {  U_asList.Add(0); }
        else { U_asList.Insert(toAdd[i], 0); }
    }
    U = new DenseVector(U_asList.ToArray());
```

```
}


// using the displacement vector, compute elemental energy, then output density
Matrix<double> GetPotentials()
{
    Matrix<double> temp = new DenseMatrix(xSize, ySize);
    Vector<double> u = new DenseVector(8); // holds the displacements of the elements

    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            // adds the displacements to the element displacement vector
            int[] edof = edofs(i, j);
            for (int x = 0; x < 8; x++) { u[x] = U[edof[x]]; }

            // calculate stress
            temp[i, j] = (u.ToRowMatrix() * LSM * u)[0];
        }
    }
    return temp;
}
#endregion


#region Optimisation
// starts process
public void Optimise()
{
    if (ConditionsAreValid())
    {
        foreach (GameObject f in InputManager.forces) { f.GetComponentInChildren<Draggable>().enabled = false; }
        foreach (GameObject f in InputManager.fixpoints) { f.GetComponentInChildren<Draggable>().enabled = false; }

        iteration = iterations;
        graph.Clear();
        InputManager.activeObject = null;

        p = Matrix<double>.Build.Dense(xSize, ySize, volumeSlider.value);
        optimiseStatus = "iterating";
    }
    else
    {
        optimiseStatus = "optimised";
    }
}


// apply a 'blur' type effect to eliminate checkerboarding and anomalies
Matrix<double> Filter(Matrix<double> matrix)
{
    Matrix<double> temp = new DenseMatrix(xSize, ySize);
    for (int j = 0; j < ySize; j++)
    {
        for (int i = 0; i < xSize; i++)
        {
            List<double> weights = new List<double>();
            for (int v = -(int)rMin; v < (int)rMin; v++)
            {
                if (j + v > 0 && j + v < ySize)
                {
                    for (int u = -(int)rMin; u < (int)rMin; u++)
                    {
                        if (i + u > 0 && i + u < xSize)
                        {
                            double dist = Sqrt(v * v + u * u);
                            if (dist < rMin)
                            {
                                weights.Add((1 - (dist / rMin)) * matrix[i + u, j + v]);
                            }
                        }
                    }
                }
            }
            temp[i, j] = weights.Average();
        }
    }
    return temp;
}

// bisection algorithm to meet the users volume criteria
void CriteriaUpdate()
{
    // get dc matrix
    Matrix<double> dc = new DenseMatrix(xSize, ySize);
    for (int j = 0; j < ySize; j++)
```

```
    {
        for (int i = 0; i < xSize; i++)
        {
            dc[i, j] = -penal * Pow(p[i, j], penal - 1) * p[i, j];
        }
    }

    // filter dc matrix
    dc = Filter(dc);

    // update to match criteria
    Matrix<double> pTrial = p;
    double lambda, lower = 0, upper = 2.3424;
    for (int attempts = 0; attempts < 100; attempts++)
    {
        if (Abs(pTrial.RowSums().Sum() / (xSize * ySize) - volumeSlider.value) < tolerance) { break; }

        lambda = (upper + lower) / 2;
        for (int j = 0; j < ySize; j++)
        {
            for (int i = 0; i < xSize; i++)
            {
                pTrial[i, j] = Max(pMin, Max(p[i, j] - move, Min(1f, Min(p[i, j] + move, p[i, j] * Sqrt(-dc[i, j] / lambda)))));
            }
        }
        //graph.AddPoint(pTrial.RowSums().Sum() / (xSize * ySize)); // temp
        if (pTrial.RowSums().Sum() / (xSize * ySize) > volumeSlider.value) { lower = lambda; } else { upper = lambda; }
    }
    p = pTrial;
}
#endregion


#region Index Functions
// returns the DOF indices for an element
int[] edofs(int x, int y)
{
    int n1 = ndof(x, y);
    int n2 = ndof(x + 1, y);
    //string output = "";
    //foreach (int k in (new int[] { n1 - 2, n1 - 1, n2 - 2, n2 - 1, n2, n2 + 1, n1, n1 + 1 })) { output += k.ToString() + ", "; }
    //Debug.Log("edof for element x: " + x.ToString() + " y: " + y.ToString() + " is " + output);
    return new int[] { n1 - 2, n1 - 1, n2 - 2, n2 - 1, n2, n2 + 1, n1, n1 + 1 };
}


// returns the DOF index of a node
int ndof(int x, int y)
{
    return 2 * ((ySize + 1) * x + ySize - y);
}

List<int> GetFixpointIndices()
{
    List<int> indices = new List<int>();
    foreach (GameObject f in InputManager.fixpoints)
    {
        Vector2 pos = f.GetComponent<RectTransform>().anchoredPosition;
        if (f.GetComponentInChildren<FixpointManager>().x) { indices.Add(ndof((int)pos.x, (int)pos.y)); }
        if (f.GetComponentInChildren<FixpointManager>().y) { indices.Add(ndof((int)pos.x, (int)pos.y) + 1); }
    }
    indices.Sort();
    //string output = "";
    //foreach(int i in indices) { output += " " + i; }
    //Debug.Log(output);
    return indices;
}
#endregion

bool ConditionsAreValid()
{
    foreach (GameObject f in InputManager.forces)
    {
        f.GetComponentInChildren<Draggable>().enabled = false;
        if (!f.GetComponentInChildren<ForceManager>().isValid)
        {
            ErrorText.ShowError("Invalid force detected");
            return false;
        }
    }
    foreach (GameObject f in InputManager.fixpoints)
    {
        f.GetComponentInChildren<Draggable>().enabled = false;
        if (!f.GetComponentInChildren<FixpointManager>().isValid)
        {
            ErrorText.ShowError("Invalid fixpoint detected");
            return false;
```

```
            }
        }

        if (InputManager.forces.Count == 0)
        {
            ErrorText.ShowError("Force required");
            return false;
        }
        else if (InputManager.fixpoints.Count < 2)
        {
            ErrorText.ShowError("2 or more fixpoints required");
            return false;
        }
        return true;
    }


    #region Outputs
    public void Display()
    {
        dispMarcher.GetComponent<MeshRenderer>().enabled = true;
        dispMarcher.U = U;
        dispMarcher.Render(p, false);

        if (topoMesh.isOn)
        {
            topoTexture.enabled = false;
            topoMarcher.GetComponent<MeshRenderer>().enabled = true;
            topoMarcher.U = null;
            topoMarcher.Render(p, topoColor.isOn);
        }
        else
        {
            topoTexture.enabled = true;
            topoMarcher.GetComponent<MeshRenderer>().enabled = false;
            Texture2D texture = new Texture2D(xSize, ySize);
            topoTexture.texture = texture;
            for (int j = 0; j < ySize; j++)
            {
                for (int i = 0; i < xSize; i++)
                {
                    if (topoColor.isOn) { texture.SetPixel(i, j, gradient.Evaluate((float)p[i, j])); }
                    else { texture.SetPixel(i, j, new Color((float)p[i, j], (float)p[i, j], (float)p[i, j], 1f)); }
                }
            }
            texture.filterMode = FilterMode.Point;
            texture.Apply();
        }
    }

    public void UpdateDisplay()
    {
        if (optimiseStatus != "idle") { Display(); }
    }
    #endregion
}
```