Portfolio A

# Overview

## Problem

3D printers print in layers; this means that it is not possible to print any part of a model which has nothing below it, as this part of the model would collapse during the print. Resulting in that section and potentially the rest of the model failing to be printed correctly.

Current 3D printing software has functionality to detect these unsupported sections and add supports. In their most simple form, these supports are simply vertical lines which directly attach the bottom of the print to the unsupported section. In reality, these supports are usually pipe-like structures which branch from not only the bottom of the print but also from other parts of the print and even other supports, ensuring every section is supported.

3D print models are developed in multiple stages. The first stage is the 3D design stage – usually with a graphical piece of software, such as *Autodesk Fusion 360*. Once a 3D design is produced, it is then sliced – separated into horizonal planes called slices, which each slice being a grid of 1s and 0s. A 1 represents that there should be plastic at that location, and a 0 that there shouldn’t.

The problem the industry currently faces is that the supports described above can only be implemented before the print has been sliced, and not after. Whilst this is often not a problem, there are situations where this can cause great delay in the printing process.

Currently there are some solutions to this problem but mostly for Filament Deposition Modelling (FDP) printers – these printers lay filament onto the print platform and build upon that in layers. Our client wishes for us to develop a solution for use with Digital Light Processing (DLP) printers. These printers work by exposing an entire layer of resin to light which cures the resin. This layer is then pulled to the top of the resin bed and the next layer is cured with the light. There are no real automated solutions for creating supports for this type of printers.

## Project

Our project is to develop a program which will read in a post-slice 3D print model and check whether the model has all required supports.

Depending on our progress, we would also like to implement features to add in these supports, and to optimise the supports in the following ways:

* use the least possible material
* be as easy to remove as possible
* merge/bridge supports where possible

The program must accept a basic file format, which will effectively be a .zip file containing many .png files, with each representing one layer of the model.

We would also like to be able to read in a second, more complicated file type, but this will depend on our progress.

## Client

Our client is Oliver Matthews, a Software Engineer at Hewlett Packard Enterprise. Oliver doesn’t work with 3D Printers in his job but is a keen hobbyist who regularly contributes to 3D Printing software development communities online.

# Requirements

From our projects aims and overview we have identified the following key stakeholders

* Consumers: who own a personal DLP 3D printer for their own use at home
* Commercial Printing Businesses: who print 3D models for a fee and send them out for a fee to consumers if they do not own their own 3D printer
* Developers: who currently produce software related to the DLP printing workflow, such as tools for adding supports manually and model slicers
* Manufacturers: who design, produce and support their own DLP 3D printers

While there are a variety of stakeholders for our project, we will not directly interact with many of them, and in fact the use case for our software is likely to be very similar for many of them. We have provided a series of user stories for consumers who will directly use our system. First are a set of which are directly relevant to our MVP.

* The program should identify unsupported islands in the sliced file so that the model can be fixed before a failed print occurs
* The program should automatically add basic supports to the unsupported islands directly to the sliced file so that the need to return to 3D modelling software and slicer is eliminated resulting in the printing workflow being simplified

On top of this there are a variety of user stories which we will aim to consider beyond our MVP, these are:

* The program should create sloping supports which avoid the print material below them and contact the model at an appropriate angle such that the model is both effectively supported with minimal scarring
* The program should have a GUI which allows the user to add their files and potentially visualise changes to the model, so that the user can easily see modifications before they print
* The program should add appropriate supports which account for the suction applied to the model during DLP printing by varying quantity or thickness, so that the guesswork in adding supports to 3D models is reduced
* The program should intelligently group supports to pillars so that the wasted material is reduced, and model clean-up becomes simplified
* The program should add bracing struts between nearby supports so that the supports are reinforced during the printing process
* The program should use antialiasing when adding supports to soften them in key points so that they break off more easily during the post print removal process

The core goal of the project is for the software to analyse sliced 3D files and identify unsupported islands in the model. There are a few different potential flows broken down below.

BASIC FLOW (user imports sliced file which is fully supported)

1. User opens the program
2. User imports their sliced file
3. Program analyses each layer of the file
4. Program informs the user that no additional supports are required
5. The program doesn’t modify the sliced file
6. The user goes onto print the file

ALTERNATIVE FLOW (user imports sliced file which is not supported)

1. User opens the program
2. User imports their sliced file
3. Program analyses each layer of the file
4. Program informs the user of the unsupported islands
5. Program suggests adding supports in necessary locations
6. User confirms the changes
7. The program modifies the sliced file with the new supports
8. The user goes onto print the file

EXCEPTIONAL FLOW (User imports an incorrect file type)

1. User opens the program
2. User loads a file type which is not recognised or has the wrong format
3. Program alerts the user the file cannot be read properly
4. Program offers the user to try importing a file again
5. User either quits or enters a correct file and follows another flow

Based on these user stories and our goals for developing a program to implement them we have determined a set of requirements for the project, including those for the MVP (which are in bold) and further extensions to it:

1. GENERAL
   1. **The program accepts and opens sliced files** 
      1. **SL1 files (zip file of greyscale PNGs)**
         1. Be able to handle anti-aliased greyscale images
      2. Photon files (run length encoded binary)
   2. **The program detects unsupported islands in the file**
      1. **differentiate between islands and parts of the model which are diagonally supported**
   3. **The program adds supports in necessary locations which:**
      1. are sloping to avoid the model’s print material below them
      2. are at an appropriate angle
         1. To support the model
         2. To be removed easily
      3. account for the suction forces they will need to withstand
      4. Are grouped onto the same pillars for easy removal
      5. Are cross braced to other supports for strength
      6. Are anti-aliased for easier removal
   4. **The program outputs a printable file**
      1. **In the Sl1 file format**
      2. In the photon file format
   5. **The program runs on windows as a local executable**
      1. **The program uses a reasonable amount of space in memory**
      2. The program supports Linux and Mac as well
2. User Interface
   1. **Can be run in a terminal window**
      1. **Clear instructions for giving file directory**
      2. **Option to overwrite existing file or create a new one at the end of modification**
   2. Can be run in a graphical interface
      1. Has directory browser for selecting file
      2. Can visualise the sliced file
         1. Can show the support modifications the program intends to add
         2. Allows the user to select certain supports to be included/ removed from the modifications
      3. Configuration fields available to control
         1. Support thickness
         2. Max support grouping size
         3. Whether cross bracing is added to supports or not
      4. Clear buttons for overwriting file or saving as a new one

|  |  |  |
| --- | --- | --- |
| No. | Name | Description |
| **1.** | **Program Requirements** |  |
| 1.1 | Windows application | Ideally our program will be cross platform and be able to run on MacOs, Windows and Linux. Our client needs the program for his Windows computer. |
| **2.** | **Input/Output** |  |
| 2.1 | File type | Must be able to read in a SL1 file and output the supported version in the same format |
| 2.2 | File Validation | Other input types should be rejected |
| 2.3 | Input File from Disk | The file should be read in from the user’s filesystem |
| 2.4 | Output File Location | The file should be saved to a user specified directory |
| 2.5 | Output File Name | The file name should remain the same with ‘SUPPORTED’ appended to the end of it as to not overwrite the input file |
| **3** | **Island Detection** |  |
| 3.1 | Unsupported Sections | The program must detect unsupported sections of the design |
| 3.2 | Overhangs | The program must work out if an overhang in the design will need supporting |
| **4.** | **Support Creation** |  |
| 4.1 | Pillars | The program should create cylindrical pillars to support the unsupported sections |
| 4.2 | Rounded Ends | The supports should have rounded conical tips so that they can be removed from the model easily after printing |
| 4.3 | Minimise Material | The supports should use as small amount as material as possible |
| 4.4 | Impact Model | The supports must not alter the model as to make it look/act differently |
| **5.** | **Resource Usage** |  |
| 5.1 | 180 second runtime | The program should run in less than 180 seconds as that was determined to be an acceptable time by our client |
| 5.2 | Memory Usage | The program should never use more than 5gb of memory throughout its processing |
| 5.3 | Output File Size | The output file should not be significantly (>50%) bigger than the input file |
| **6.** | **Interface** |  |
| 6.1 | Text Based | The program must as a minimum have a command line interface |
| 6.2 | GUI | A graphical use interface would be preferrable – time permitting. |
| 6.3 | Simplicity | The program must abstract away details and be easy to use |
| 6.4 | Intuitive | The program should make it clear how to use it without the needs for a manual |

# Personal Data, Privacy, Security and Ethics Management

Our application as its minimum viable product will not have a need to store/manage personal data about the users. In the future there could be a need for accounts which could expand functionality for example a backup system for the created files; in which case we would need the appropriate permissions from the user to manage their data. Users may be concerned about our program storing their 3D designs and therefore having the potential to steal their designs. In order to avoid any issues regarding this - our program will process their file and provide an output file without saving any of their data to the application storage. Our program will process their model locally without having to send it to a server and so that avoids any issues regarding their designs being seen by us.

Passwords would need to be stored in an encrypted manner. The passwords should be hashed ideally using SHA-256. Ideally the hashed passwords would be created using a dynamic salting method so that they are more secure.

We need to ensure that our program has the correct permissions for the system. The program should only have access to the files inputted into it and the program should be able to read files that have not explicitly been chosen by the user. There is no requirement for the program to have network access and so that limits concerns for the user regarding the handling of their files. Our program will need read/write permission to read in the input files and write the new created models with the supports to the user’s computer.

Since we won’t be handling user data there are no real ethical issues. The main aspect we need to consider is keeping the user’s models private as to not steal their intellectual property.

# Architecture

Our program will be a **Java application** that runs **offline** entirely in the **user’s** **machine** relying only on its available computational capacities mainly **CPU and RAM**.

The requirements suggest that our program should be designed to fit into the 3D printing flow exactly between **the slicer** and **the printer**.

**Image conversion :**

Interacting with the slicer implies that our program’s input is a ZIP collection of PNG files, each of which representing a layer read by our program as a binary matrix to form a tensor.

This choice is preferred over the 24-bit Java native image storing because of memory considerations. In fact, our client clearly mentioned that nowadays printers are using 1080p and 4K screens. Considering the number of layers a typical 3D model contains (~1000 layer) - encoding each pixel using the 24-bit model would require an irrational amount of RAM. As a solution, our program will use a tensor of matrices, where every layer is simply a binary matrix with ‘1’ for existing matter and ‘0’ for no matter.

**Island Detection & Support Creation**

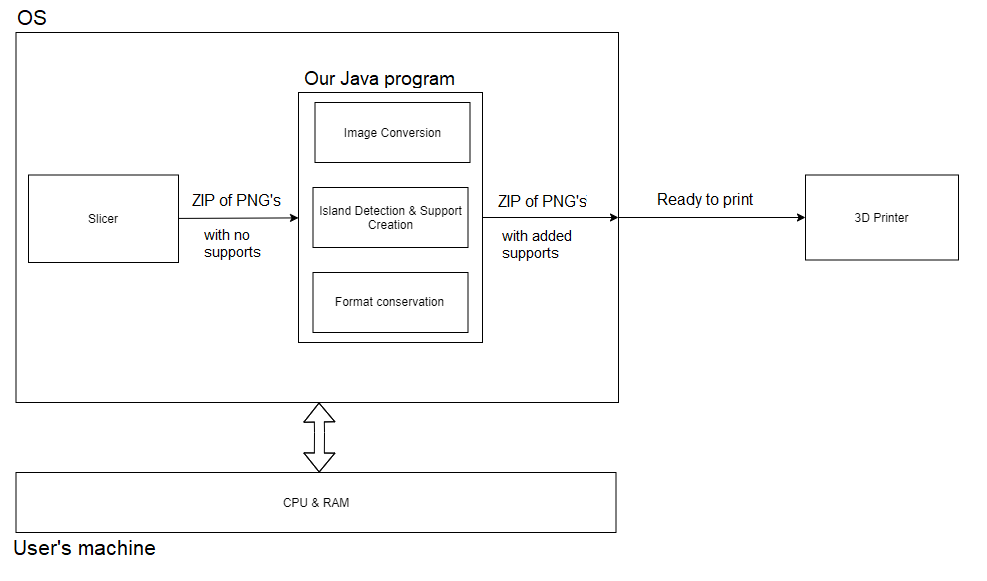
In order to perform Island Detection, our program will iteratively compare matrices of each consecutive layer and place supports - flipping 0’s to 1’s- in the correct places in every layer of the tensor.

The complexity of the model generating supports relies on the type of the comparison:

* Inter-layer comparison (MVP): Takes into consideration layers directly below the island and adds vertical supports.
* Adding Intra-layer comparisons (Features): Takes also into consideration surrounding material within the same layer to generate optimal supports.

**Format conservation:**

Since our program’s output is the direct input to the 3D printer, our program conserves the format required – ZIP of PNG’s- by converting the tensor from matrices to PNG’s and compressing them in the same order as the input before finally writing them in the user’s preferred directory.



# Development Testing

We plan on programming the testing of a feature before implementing it in order to give us a clear understanding of how the feature should work before writing it, but it can be changed at any time to ensure that we don’t gain tunnel vision on one possible implementation and to fit the agile strategy.

Our software will have 4 major areas which need development testing performed on them: Reading in the files as valid matrices, locating the areas that need supports, accurately and efficiently implementing these supports and converting the supported file into a valid format to be printed.

Locating the areas that need supports is an area that has clear correct and incorrect answers, as opposed to implementing the supports which is much more ambiguous, so the testing will be well-defined and somewhat simple to understand.

We will automatically test our island detection by comparing it with the output of Photon File Validator, another program that analyses whether there are unsupported islands in a model; while this normally only outputs data graphically, it is open source, and shouldn’t be difficult to edit to output a simple, non-compressed file to compare to. Hence, we can write a program to automatically compare the output file from our edited version of Photon File Validator with our own island detection program.

There are many models available online to test out your 3D printing software, which can be repurposed for us to test not only the locations that need supports, but also the implementation of the supports in the future. We could also use models we make ourselves in order to test edge cases.

Effectively, our tests would be laid out as such:

* Download model from the internet or make a model
* Run our program on the model, outputting a file that matches the formatting of another support analysis program’s output
* Run tests to compare the files, pointing out any differences that may exist

Edge cases would be those that exhibit particularly strange qualities, such as those that would require an excessive number of supports (imagine many single-pixel points dotted around in 3d space) or those that have a more complex supportive structure that, despite having overhangs, are still structurally stable (imagine a honeycomb-like structure).

# Release Testing

Our project will be tested upon each iterative release according to the following strategy. The purpose of this testing will be two-fold: to ensure that the new features perform as expected, and to ensure that no new changes have broken any other aspect of the program.

## User Story

Our release testing will focus on our primary user story – somebody who is printing on a 3D printer and needs to verify after they have sliced their print that there are no unsupported sections. All user stories will follow from this one, as our program only has one function. Therefore, this is the best focus of any release testing as it should capture the requirements of most/all other user stories.

## Testing new functionality

The first part of this strategy, testing new functionality, is not something that can be generically planned at this early stage. Each release will require different tests to check the new functionality, and therefore these tests can only be designed at the point of release. However, we can be sure that these tests must test a variety of cases including edge cases and invalid inputs.

## Testing fundamental and existing functionality

The second part of this strategy, testing fundamental and existing functionality can be far better planned. At every stage, we will run our program on the five models listed below. These models will hopefully capture all edge cases and the variety of real-world situations that should be tested.

1. The empty-print. This will be a 3d model which prints nothing (i.e. all 0s).
2. The full-print. This will be a 3d model in which every location should be filled (i.e. all 1s).

For the above tests to pass, the program should find no unsupported sections and therefore should add no supports.

1. *#3dBenchy*, from [thingiverse](https://www.thingiverse.com/thing:763622).
2. *All in One 3D Printer Test*, from [thingiverse](https://www.thingiverse.com/thing:2656594).
3. *PolyPearl Tower Torture Test*, from [thingiverse](https://www.thingiverse.com/thing:2064029).

Tests 3, 4 and 5 should test various printing situations and edge cases. Each was designed to test the functionality of 3D printers and printing software, including the support functionality which is important for all three designs.

For these tests to pass, each should return the same number of unsupported sections as a pre-slice support check on that print. If we have implemented functionality to build supports, these tests will pass if those supports correctly support all of the print.

These three tests are all published under a Creative Commons licence, which allows us to use them in our testing. More details on each of the licences can be found on their thingiverse pages.

In theory, all tests could all be automated. There could potentially be some room for disagreement with the expected output, as different printing techniques have different tolerance for unsupported sections. In these cases, our tests will establish some threshold of tolerance and the tests will pass if this threshold is not exceeded.

If we have also implemented efficient supports (according to the optimisations in the project overview), we could write a test which measures this efficiency in some way, again with a threshold to be met for a pass.