

# **Developing a Digital-to-Print Fabrication Pipeline for Multi-Color Photochromic 3D Printing**

by

**Sabina W. Chen**

Bachelor of Science in Electrical Engineering and Computer Science  
Massachusetts Institute of Technology, 2019

Submitted to the Department of Electrical Engineering and Computer Science  
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science  
at the

**MASSACHUSETTS INSTITUTE OF TECHNOLOGY**

June 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author: \_\_\_\_\_  
Department of Electrical Engineering and Computer Science  
May 20, 2021

Certified By: \_\_\_\_\_  
Stefanie Mueller  
X-Window Consortium Career Development Assistant Professor  
Thesis Supervisor

Accepted By: \_\_\_\_\_  
Katrina LaCurtis  
Chair, Master of Engineering Thesis Committee

# **Developing a Digital-to-Print Fabrication Pipeline for Multi-Color Photochromic 3D Printing**

by

**Sabina W. Chen**

Submitted to the Department of Electrical Engineering and Computer Science  
on May 20, 2021, in partial fulfillment of the  
requirements for the degree of  
Master of Engineering in Electrical Engineering and Computer Science

## **Abstract**

Stereolithography (SLA) and digital light processing (DLP) 3D printing are two common resin-based 3D printing processes. These printing processes work by projecting a light source onto specific areas of resin, forming thin layers of plastic that eventually stack up to create solid objects. Because only one resin type can be used at a time, one limitation of SLA and DLP 3D printing is that they typically only produce single-color prints. In this project, we present a novel approach that enables multi-color resin printing using photochromic dyes. By combining DLP with photochromic materials, our end-to-end 3D printing fabrication pipeline can create multi-colored objects using only one type of resin-based material.

Our approach involves designing a system that supports the integration of photochromic materials into a UV curable resin. By developing a resin that contains a mixture of photochromic inks that can change color when exposed to different wavelengths of light, we can programmatically change the color of the resin depending on the type of RGB light projected. To support this new resin, we modified an existing resin-based 3D printer to incorporate both a UV and visible light projection system. This enables us to control both the curing and coloring of an object separately. By saturating the dyes prior to printing, and then projecting combinations of RGB light onto each layer after it has been cured, we can color objects directly during the printing process.

In this thesis, we provide the implementation details and design decisions that went into building this integrated 3D printing infrastructure. We discuss the user interface, printer hardware, software implementation, and photochromic resin formulation. We also provide operational instructions and explanations for key design decisions of our system. Finally, we evaluate the capabilities of our photochromic resin and printer system, and propose topics for future work.

Thesis Supervisor: Stefanie Mueller

Title: X-Window Consortium Career Development Assistant Professor



# Acknowledgements

First and foremost, I want to sincerely thank my thesis advisor, Professor Stefanie Mueller. Thank you for the opportunity to work on such unique and interesting research, and for providing me with invaluable guidance and support through my graduate career. I have learned so much in such a short amount of time, and it has been a genuine pleasure to work with you.

I am also thankful to my academic advisor, Professor Gim P. Hom, for providing me the freedom to explore various academic pursuits throughout my time at MIT. The experiences that have arisen from these opportunities have guided much of my research and have greatly shaped my career interests moving forward.

Thank you to Isabel Qamar, my research collaborator and supervisor, who has become a great friend and mentor through this project. She has been incredibly patient in working with and testing all the hardware developed during this project, and has provided me with many research and life lessons.

Thank you to Michael Wessely, my research supervisor, who has been a great source of technical knowledge and advice. His software and hardware expertise have guided much of my design decisions and research directions in this project. Thank you so much for advising me across all the different technical challenges, and for being an overall invaluable mentor during my MEng.

I would also like to express gratitude to everyone in the HCI Engineering Group for creating such a welcoming and creative environment. Everyone has been so nice, helpful, and genuinely wonderful to work alongside! A huge thank you to everyone who has worked on the PhotoChromic 3D Printing project with me, in particular: Isabel Qamar, Faraz Faruqi, Dimitri Tskhovrebadze, Kevin Tang, Paolo Boni, Michael Wessely, and Stefanie Mueller.

Lastly, I am extremely grateful to my friends and family. Thank you all for the unconditional love and support. College has been challenging, but I am extremely thankful to have had all of you by my side through it all.

## **Publications**

My MEng thesis contributed to the hardware, software, and materials development of an ongoing research project conducted under MIT CSAIL'S HCI Engineering Group. The paper is in preparation for ACM CHI 2022.

# Table of Contents

<b>1. Introduction</b>	<b>8</b>
1.1. Overview	8
1.2. Contribution	9
1.3. Language and Figure Note	9
<b>2. Related Work</b>	<b>10</b>
2.1. Multi-Color Resin-Based Printing Systems	10
2.2. Multi-Color Programmable Matter	10
2.3. Multi-Color Printing using Photochromic Dyes	11
<b>3. Implementation</b>	<b>12</b>
3.1. Hardware	12
3.1.1. 3D Printer Setup	12
3.1.2. Mirror	13
3.1.3. Camera	17
3.1.4. Electronics	17
3.1.4.1. Wiring	17
3.1.4.2. Actuators	19
3.1.4.3. Sensors	21
3.1.4.4. Power	22
3.2. Software	22
3.2.1. Blender	22
3.2.1.1. Preparing the Model	22
3.2.1.2. Exporting Images	25
3.2.1.3. Exporting Settings	26
3.2.1.4. Printer Control	26
3.2.1.5. Prioritizing “Stop”	27
3.2.2. Processing	28
3.2.2.1. State Machine	28
3.2.2.2. Desaturation Algorithm	32
3.2.2.3. Communicating between Processing and the Projectors	32
3.2.2.3. Communicating between Processing and Blender	32
3.2.2.4. Communicating between Processing and Arduino	32
3.2.3. Arduino	33
3.2.3.1. Commands	33
3.2.4. Web Server	34
3.2.4.1. Database	34
3.2.4.2. API	35

3.2.4.3. Website	35
3.2.4.4. Updating Printer State	36
<b>4. Calibration</b>	<b>38</b>
4.1. Projector Alignment	38
4.2. Leveling the Build Plate	39
<b>5. Evaluation</b>	<b>40</b>
5.1. Developing the Photochromic Resin	40
5.2. Resin Curing Times	40
<b>6. Future Work</b>	<b>42</b>
6.1. Dye Desaturation Times	42
6.2. Evaluating the Effect of Visible Light through Thickness	43
6.3. Determining the Color Gamut	45
6.4. Conducting Larger Prints	45
6.5. Decreasing Printing Time	46
<b>7. Conclusion</b>	<b>47</b>
<b>8. Bibliography</b>	<b>48</b>

## List of Figures

Figure 1: Conceptual design of the multi-color 3D printing system	12
Figure 2: Assembled hardware	12
Figure 3: 3D models of the mirror components	14
Figure 4: Technical drawing for hall-effect endstop	15
Figure 5: Technical drawing for mirror case w/ magnet	15
Figure 6: Individual mirror components	16
Figure 7: Assembled mirror components	16
Figure 8: Wiring schematic	18
Figure 9: Wiring diagram	18
Figure 10: Protoboard with all electronics connected	19
Figure 11: Steps-to-MM conversion for the Z-axis stepper motor	20
Figure 12: Blender user interface	23
Figure 13: Sliced model	23
Figure 14: Support generation	24
Figure 15: Color preview	24
Figure 16: Textured 3D pyramid model	25
Figure 17: Texture projection views	25
Figure 18: Layer image generation	25
Figure 19: Printer Settings	26
Figure 20: Printer control buttons	27
Figure 21: Processing system control	28
Figure 22: Processing state machine	29
Figure 23: Processing state descriptions	30
Figure 24: Database variables	34
Figure 25: REST API	35
Figure 26: Website GUI for remote operation	35
Figure 27: Updating the database	36
Figure 28: UV projector calibration	38
Figure 29: Visible light projector calibration	38

Figure 30: Leveling the build plate	39
Figure 31: Photochromic resin	40
Figure 32: Cured prints	41
Figure 33: Projected desaturation bars	42
Figure 34: Printed desaturation bars	43
Figure 35: Conceptual drawing for effect of visible light through thickness	43
Figure 36: Processing state machine for experiment 2	44
Figure 37: Example colored prints	45
Figure 38: Large print example	45

# 1. Introduction

## 1.1. Overview

Resin-based printing typically involves printing objects from a single material, and therefore a single color. Thus, to create objects of multiple colors, users will often need to carry out multiple inter- or post- fabrication steps. This process can be time-consuming, especially for makers in the HCI community that may require objects to be rapidly prototyped or integrated into existing systems. To facilitate this issue, we developed a resin-based 3D printing method that enables objects to be colored directly during the printing process.

Our proposed method enables users to print multi-color objects using only one type of resin-based material by programmatically altering the material property of our resin. This thereby reduces the number of fabrication steps necessary to create multi-colored objects, and enables multi-colored textures to be applied to both the interior and exterior of objects.

Our printing process combines digital light processing (DLP) with photochromic materials. Inspired by the method described in *Photo-Chromeleon* [1], we mix cyan, magenta, and yellow photochromic dyes into a UV curable resin. These photopolymers can switch between colored (saturated) and transparent (desaturated) states when exposed to specific wavelengths of light. By saturating the dyes prior to printing, and then projecting combinations of RGB light onto each layer to selectively desaturate each of the dyes in the resin, we can color objects as they are being printed.

Our project aims to make a complete digital-to-print fabrication pipeline of the multi-color 3D printing process. Specifically, our contributions include:

- A new method for multi-material 3D printing for single material DLP 3D Printers
- A new hardware setup
- An user interface for applying textures to models
- An algorithm for coloring the object during printing
- A new photochromic resin formulation

In this thesis, we provide the implementation details and design decisions that went into building this integrated printing infrastructure, as well as an overview of the results from the experiments conducted. We also provide a brief discussion on the limitations of our current design, as well as recommendations for future work moving forward.

## 1.2. Contribution

This project was a collaborative effort between me and Isabel Qamar, under the supervision of Isabel Qamar, Michael Wessely, and Stefanie Mueller. Isabel spearheaded the high-level design, research, and planning for the project. She also built the majority of the mechanical components for the printer and conducted the final materials experiments and evaluations.

Working off of Isabel's design ideas, I implemented the electronics, embedded systems, and software technologies required for this project. A few of my responsibilities included setting up the electronics, designing the control algorithms, implementing the web server and API required for remote operation, building the mechanical components for the mirror, brainstorming operational and experimental procedures, and developing the overall system infrastructure.

Contributions were also made by Paolo Boni, who designed the first iteration of the printer, Kevin Tang, who updated the Arduino communication protocol, Dimitri Tskhovrebadze, who developed the Blender add-on for the user interface, and Faraz Faruqi, who conducted the data analysis of our experimental results.

## 1.3. Language and Figure Note

The pronoun “we” will be used throughout the thesis, as the project was a team effort. However, unless specified otherwise, all the work on the electronics, embedded systems, and software infrastructure are from myself. The term “system” will be used interchangeably to refer to this project. Photographs were taken jointly by the project team.



## 2. Related Work

### 2.1. Multi-Color Resin-Based Printing Systems

Fused Deposition Modeling (FDM) and stereolithography (SLA) are two of the most common processes for 3D printing. FDM printers work by extruding thermoplastic filaments through a heated nozzle, melting the material, and then applying the melted plastic onto a build platform layer-by-layer until the part is complete. In contrast, SLA printers utilize a light source (UV Laser for Laser-SLA, Digital Light Projector for DLP-SLA, and LCD for Masked-SLA) in order to cure liquid resin into hardened plastic in a process called vat photopolymerization.

SLA printers are advantageous for professional development in a number of ways. Compared to other 3D printing methods, SLA printers consistently produce finely detailed parts with high dimensional accuracy and a surface finish. Furthermore, because resin is compatible with a wide range of photopolymers, its mechanical properties can be tailored to the specific functionalities required. This combination of versatility and functionality has enabled material manufacturers to develop innovative resin formulations for a wide range of desired mechanical properties.

For example, Formlabs offer a *Color Kit* [2] for their desktop SLA printers. This color kit contains a “Color Base” resin cartridge and five bottles of color pigments (cyan, magenta, yellow, black and white) that can be mixed together according to predetermined recipes in order to achieve a desired color. This colored resin can then be used for a single one-color print.

To extend beyond a single material, *LayerCode* [3], transforms a stereolithography printer to support two-color prints. This is done by mounting two resin trays, each to hold a different colored resin, then using a 180° rotatable build plate to switch between the two resin tanks during print. While this concept could be extended to enable additional colors by increasing the number of resin tanks, it is limited to the number of tanks that can be mounted.

### 2.2. Multi-Color Programmable Matter

Advancements have also been made to achieve multiple colors using a single material. One way to do so is to use color-changing materials (ie. materials that can change their color during or after fabrication). This can be done using electrochromic (applying voltage), thermochromic (changing temperature), or photochromic (applying light) technology. For resin-based 3D printing, photochromic color-changing materials are particularly suitable, as this printing process already uses a light source, which can then be extended to accommodate new technology.

Previous work has been done in exploring the usability of photochromic technology. Specifically, Hirayama et al. [4] explored how photochromic inks could be mixed together into a single solution, combining cyan and yellow dyes to achieve green. *Photo-Chromeleon* [1] developed a

coating system that could achieve a larger color gamut by mixing cyan, magenta and yellow dyes into a single photochromic solution. By projecting different wavelengths of visible light from a standard office projector, individual dyes can be programmatically desaturated, thereby controlling the resulting color of the photochromic solution.

## 2.3. Multi-Color Printing using Photochromic Dyes

Inspired by recent work on color-changing materials (e.g. *Photo-Chromeleon* [1], *Photochromic Carpet* [5], *Photochromic Canvas* [6], *ColorMod* [7]), our project addresses the challenge of printing in multiple colors by proposing modification to the material itself through the addition of photochromic materials to the resin. Our 3D printing method enables objects to be colored directly during the printing process, thereby reducing the number of fabrication steps necessary to print multi-colored 3D models. The system is built from an off-the-shelf SLA printer, with the addition of two separate projectors - one for curing and one for coloring. This enables our system to require only one fabrication technique and one material type to achieve our desired multi-colored print.

The relevance of multi-color 3D printing can be found across the Human-Computer Interaction (HCI) community. It allows makers and designers to create objects with important visual highlights. Examples include providing users with information on the internal configuration of breadboards [8], in the physical visualization of data [9], for developing 3D printed terrain maps [10], or for directly embedding information into physical objects [3]. Thus, our system simplifies the process for users within the HCI community by reducing the number of fabrication steps necessary to create multi-colored objects. Our fabrication procedure also permits multi-colored textures to be applied to both the interior and exterior of objects, thereby enabling users to embed information anywhere on the printed object.

The rest of this thesis details the system architecture, implementation, operating procedures, and experimental results of our system.

## 3. Implementation

### 3.1. Hardware

#### 3.1.1. 3D Printer Setup

Our multi-color 3D printing system consists of an off-the-shelf SLA printer, a UV light source for curing the resin, an office RGB projector for desaturating the photochromic dyes, and a mirror to project the appropriate light source onto the resin-filled base plate mounted above. The mirror is placed directly below the SLA printer, and can be flipped to either side using a stepper motor. The concept drawing for our design is depicted in Figure 1.

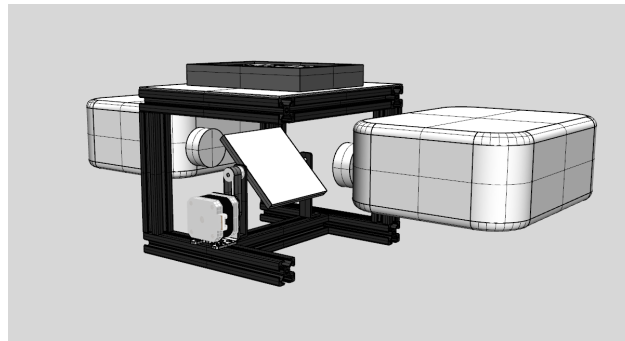


Figure 1. Conceptual design of the multi-color 3D printing system

For our assembled system, a black opaque acrylic casing was built to house all the components and to prevent undesirable desaturating of dyes from occurring during printing. Two cut outs were made at each end of the casing to mount the cooling fans for heat dissipation. We also spray painted the upper casing of the printer, which houses the build plate and resin tank, in black. The final hardware setup is shown in Figure 2.



Figure 2. Assembled hardware

The system utilizes the ANYCUBIC Photon S LCD-based SLA 3D Printer. We chose to use this printer because its built-in technology is easily modifiable. By default, the printer comes with a 115x65x165mm volume resin tank, a 115x165mm area build plate that acts as a flat surface for printed objects to adhere to, a 405nm UV LED light source combined with an LCD masking screen that controls the curing area of each layer during prints, a NEMA17 stepper motor that controls the Z-axis movement of the build plate, and an optical limit switch (Omron EE-SX674-WR 1M) that calibrates the position of the build plate at the start of every print.

After some initial experimentation, we realized that the built-in printer driver did not provide us the low-level control we needed for our system. Therefore, we replaced the default printer driver with our own external computer (Intel NUC 7, Core i7) and microcontroller (Arduino Mega 2560 Rev3). All the electronics were then rewired to be directly controlled by either the NUC or the Arduino.

We also mounted an additional visible light projector system (AAXA Technologies M6 1200 Lumens LED Projector), which operates to selectively desaturate the photochromic color channels within the resin. As in *Photo-Chromeleon* [1], the default green LED from the visible light projector creates a light output over a broad wavelength range, which causes both the yellow and cyan dyes to desaturate in addition to the magenta dye, thereby making it more difficult to selectively desaturate each dye. Thus, we added a filter in front of the green LED (Semrock Brightline® FF02-529/24-25) to limit the wavelength range, so that the green LED only desaturates the magenta dye. Filters did not need to be added to the red or yellow LEDs.

When switching on the visible light source, we also found that the LCD masking screen underneath the resin tank did not allow the visible light from the projector to pass through sufficiently to desaturate the dyes in the resin. We therefore replaced the LCD screen with a tempered glass screen (FYSTEC) and replaced the default light source with an external higher-powered UV projector (InVision Ikarus Full-HD DLP6500 light engine 385nm), set to an intensity of 12A.

The SLA printer, UV projector, and visible light projector can all be controlled through Processing. The software implementation for these components is discussed in (Section 3.2.2: Processing).

### 3.1.2. Mirror

To switch between the UV light source and the visible light source, we mounted two mirrors (Optical Mirror, Glass First Surface Mirror, reflective efficiency of 96%) back-to-back, between the two light sources. The mirrors are controlled by a stepper motor which switches the beam of light that is directed into the resin tank mounted above. A rare earth magnet mounted to the bottom of the mirrors and two hall-effect sensors located at 45° on either side of the mirrors, determines the angular position of the mirrors as they rotate.

Additional mechanical parts were developed for this mirror setup, in particular: end stops for the hall-effect sensors, housing for the mirrors and magnets, outer casings for the hall-effect sensors, and a mount for the stepper motor. The 3D CAD models were developed in FreeCAD, and are shown below in Figure 3.

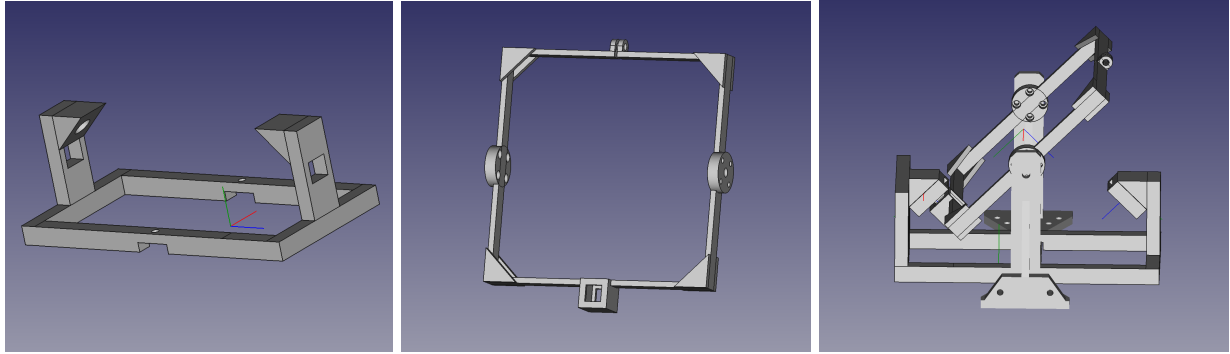


Figure 3. 3D models of the mirror components  
Hall-Effect Endstop (left), Mirror Case w/ Magnet (middle), Mirror Assembly (right)

Because the physical parameters of the components were constantly changing during the development of the 3D printer system, it became very difficult to keep track of the dimensions of each model, so we made technical drawings to document the model designs. The technical drawings were also developed in FreeCAD. The technical drawings for the hall-effect endstop and mirror case are shown in Figure 4-5, respectively.

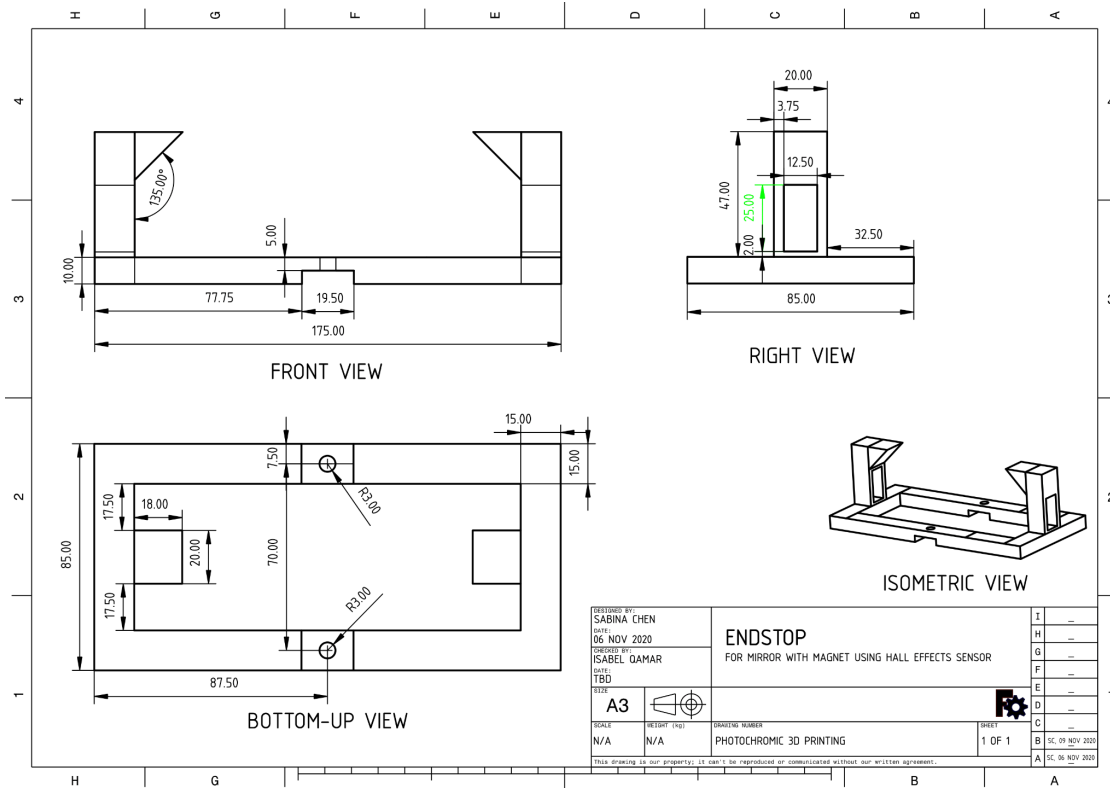


Figure 4. Technical drawing for hall-effect endstop

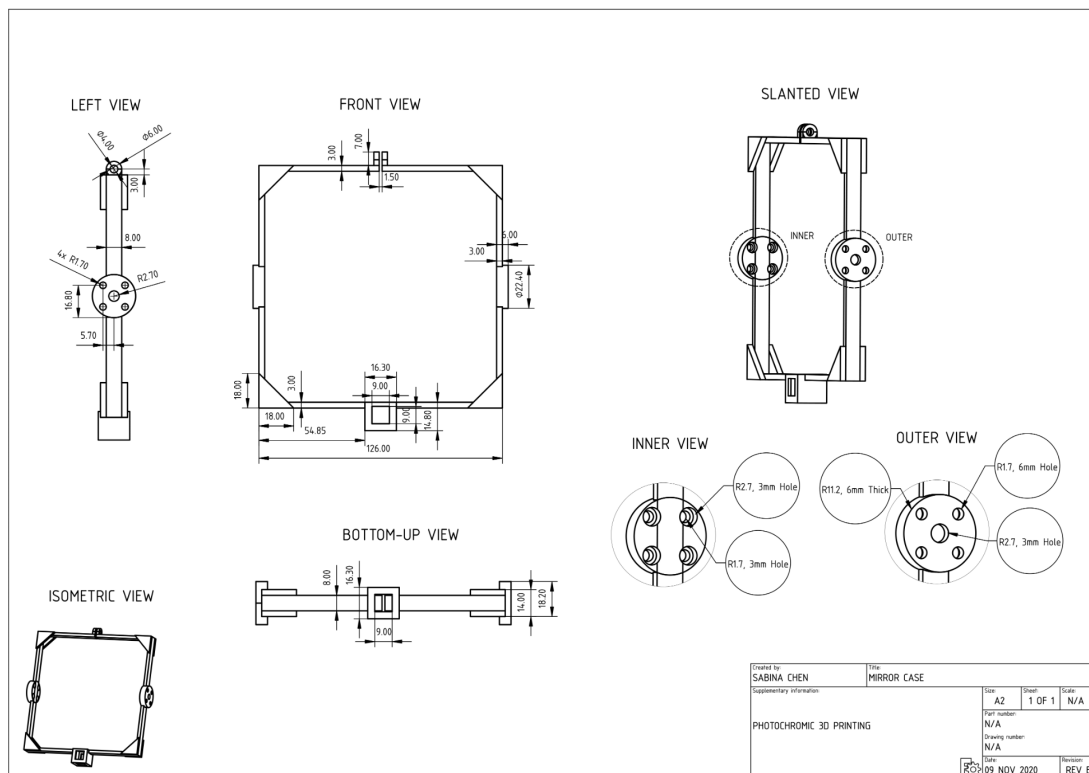


Figure 5. Technical drawing for mirror case w/ magnet

The models were then printed via the Ultimaker 3. The final printed mirror components are shown Figure 6 below.

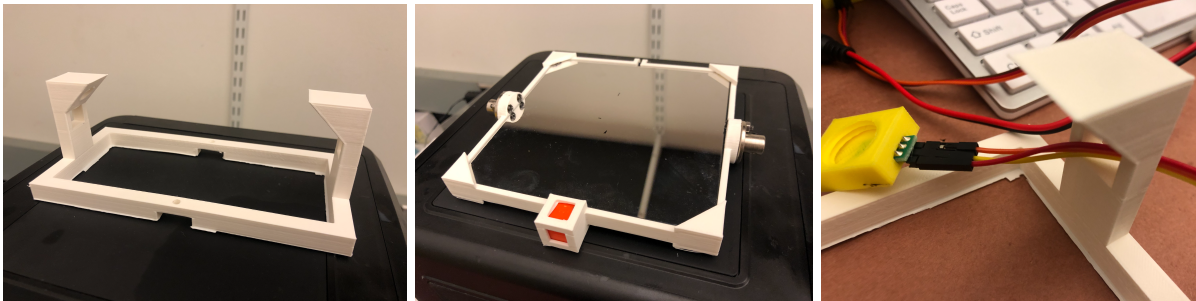


Figure 6. Individual mirror components

From left to right: endstop, mirror and magnet mount, hall-effect case

We then spray painted each printed part in black and then placed the assembled system directly below the horizontal center of the resin tank. The fully assembled and mounted mirror system is shown in Figure 7.

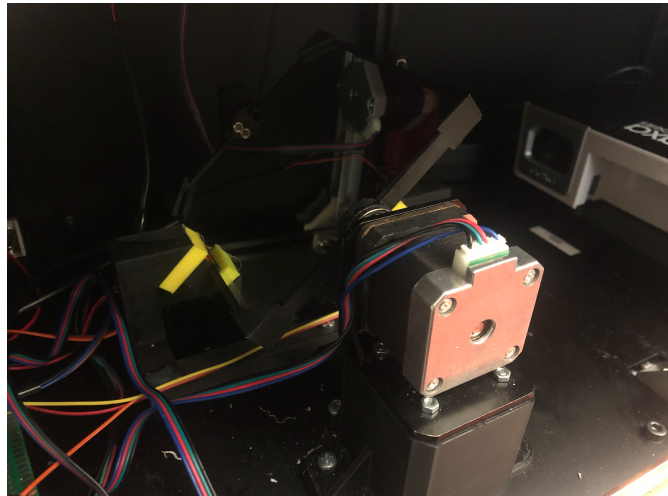


Figure 7. Assembled mirror components

### 3.1.3. Camera

To monitor the printer during long prints, we placed a camera (KAMTRON 1536P) directly in front of the printer. The camera provides a 24/7 live feed of the printer. This live feed can be viewed through a downloadable mobile application (MIPC). The viewing angle of the camera itself can also be rotated and adjusted remotely. In case of an emergency, the power to the printer can be stopped remotely via a live web server. This web server is described in more detail in (Section 3.2.4: Web Server).

### 3.1.4. Electronics

The Arduino (Arduino Mega 2560 Rev3) acts as the main driver board to communicate between the digital electronics and the computer. To facilitate wiring consistency and cleanliness, we developed a protoboard to act as a personalized Arduino shield. Header pins were soldered onto the back of the protoboard so that it can easily be plugged into the Arduino. All actuators and sensors are wired through this protoboard, instead of through the Arduino itself.

#### 3.1.4.1. Wiring

Built into the protoboard are two motor drivers (A4988 Stepper Motor Driver) and one wifi module (ESP8266 ESP-01). The protoboard is powered by a 12V power supply and provides connections to the following actuators and sensors:

- 1 optical sensor
- 2 stepper motors
- 2 hall-effect sensors
- 2 IoT relays

The wiring schematic and wiring diagram are shown in Figures 8-9, respectively.



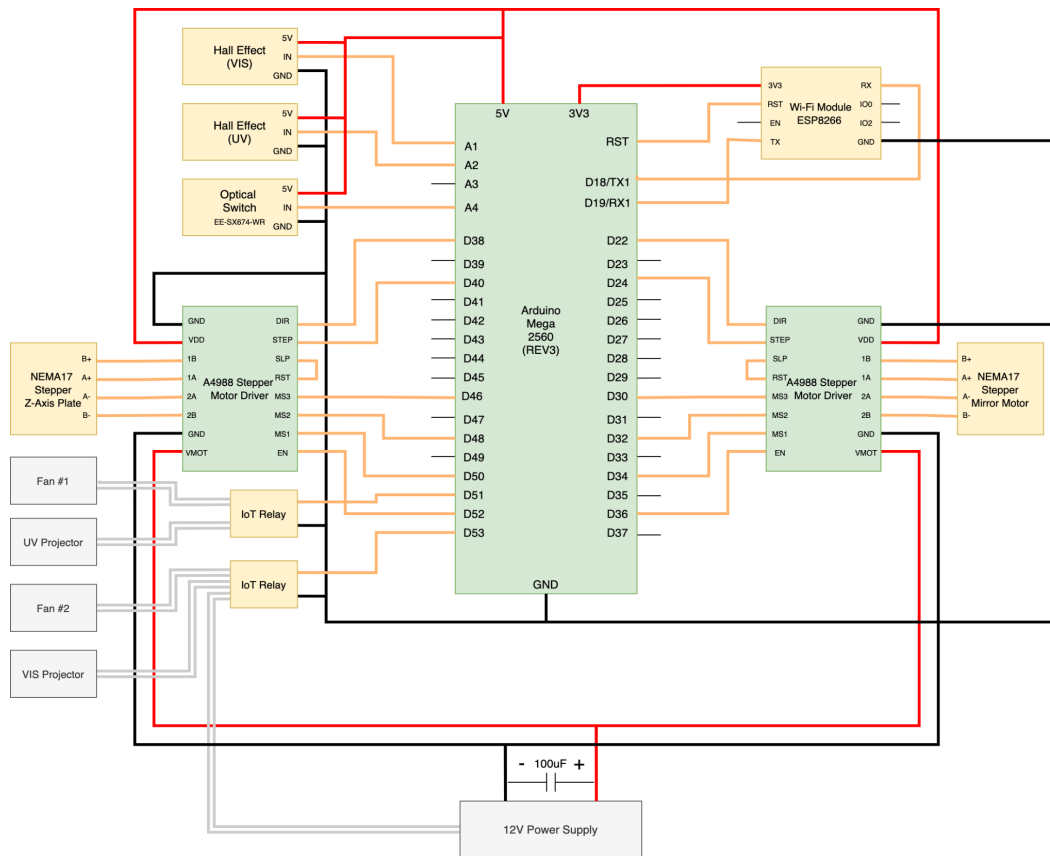


Figure 8. Wiring schematic

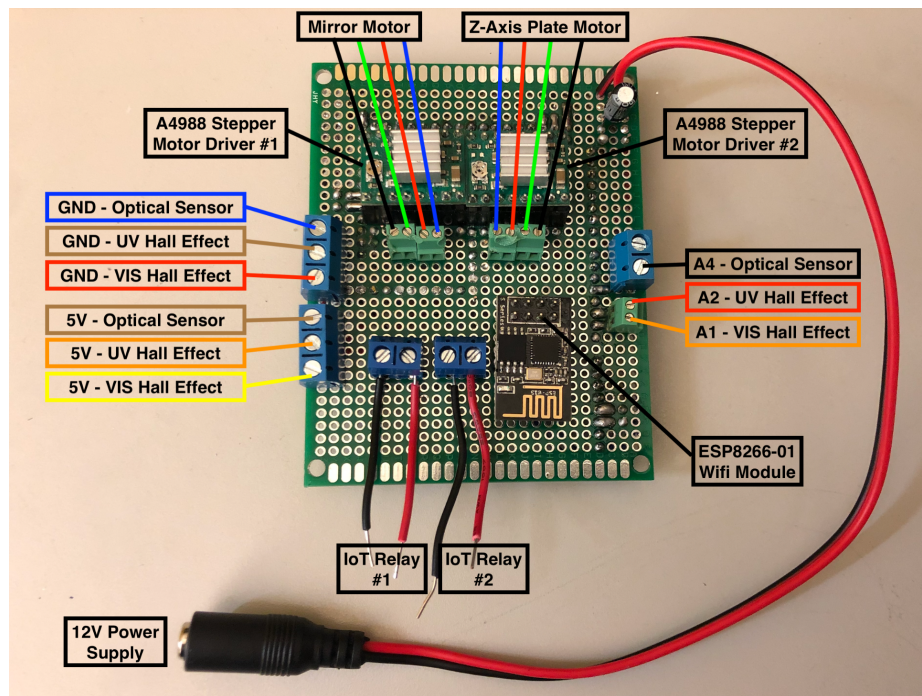


Figure 9. Wiring diagram

An example of the protoboard with all of its electronics plugged in can be found in Figure 10.

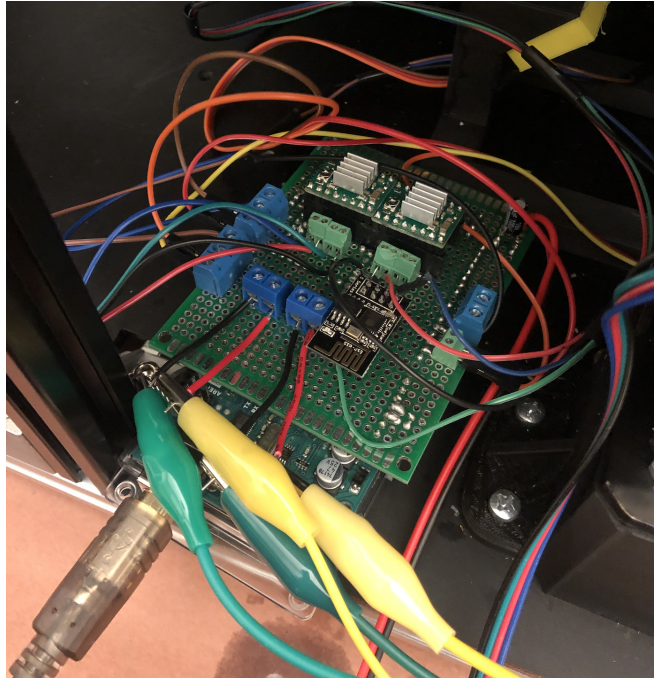


Figure 10. Protoboard with all electronics connected

#### 3.1.4.2. Actuators

Two stepper motors (NEMA17) act as actuators for the system. One motor controls mirror rotation, and the other motor controls the Z-axis movement of the base plate. Using the *AccelStepper* library and two motor drivers (A4988 Stepper Motor Driver), motor speed and distance commands can be sent to control the motors via the Arduino.

##### *Motor Driver*

A motor driver acts as an interface between each stepper motor and the Arduino. While motors require a high amount of current to operate (12V+), the Arduino only works on low current (5V). Thus, motor drivers act as intermediaries that take in low-current control signals from the Arduino, and convert these signals into higher-current outputs to drive the motors.

##### *Stepper Motor*

Stepper motors are DC motors that move in discrete steps. They have multiple coils that, when energized, will rotate the motor, one step at a time. Due to the nature of energizing these coils, stepper motors have a tendency to overheat when powered on for long periods of time. Therefore, we added heatsinks to each stepper motor to facilitate heat dissipation. Additionally, the motors were also programmatically configured to automatically power off when not directly in use (ie. during printer calibration).

While we have considered just turning the motors on and off during the main printing process to reduce overall heat dissipation, because the steppers will “jump” in position when energized, we were unable to implement this option without risking precision issues, and therefore decided to rely solely on mechanical components (ie. heatsinks and fans) to cool the motors.

### *AccelStepper Library*

Arduino uses the *AccelStepper* library to communicate with the motors via the motor drivers. This library provides an abstraction for us to set the speed and step size of the motors directly. The max speed is capped at 1000 steps/sec, as speeds over this setting are deemed unreliable according to the *AccelStepper* documentation. The NEMA17 stepper motor has a 1.8° step angle, meaning 200 steps = 1 full revolution. Steps can be divided into “microsteps”, thereby allowing us greater rotation precision (ie. ½ microstep = 2 microsteps per 1 full step). The higher the microstep, the greater the precision, but the slower the rotation speed.

For the Z-axis motor, the printer’s built-in T8 Lead Screw (2mm pitch) means the base plate will raise or lower exactly 2mm per full revolution from the stepper motor. Thus, to calculate motor speed and distance travelled, we use this information to convert steps/sec to mm/sec, depending on the selected microstep resolution. Users can set the desired speed (in mm/sec) of the Z-axis motor through the Blender UI described in 3.2.1. Blender. The associated speed calculations are shown in Figure 11 below.

<b>Resolution Settings</b>
<ul style="list-style-type: none"><li>• 1 microstep --&gt; [0,0,0] --&gt; 1 mm = 100 steps</li><li>• 1/2 microstep --&gt; [1,0,0] --&gt; 1 mm = 200 steps</li><li>• 1/4 microstep --&gt; [0,1,0] --&gt; 1 mm = 400 steps</li><li>• 1/8 microstep --&gt; [1,1,0] --&gt; 1 mm = 800 steps</li><li>• 1/16 microstep --&gt; [1,1,1] --&gt; 1 mm = 1600 steps</li></ul>
<b>Min Speed</b>
<ul style="list-style-type: none"><li>• 1 microstep --&gt; 1 step/sec = 0.01 mm/sec</li><li>• 1/2 microstep --&gt; 1 step/sec = 0.05 mm/sec</li><li>• 1/4 microstep --&gt; 1 step/sec = 0.0025 mm/sec</li><li>• 1/8 microstep --&gt; 1 step/sec = 0.00125 mm/sec</li><li>• 1/16 microstep --&gt; 1 step/sec = 0.000625 mm/sec</li></ul>
<b>Max Speeds</b>
When setting the motor speed, it is best to not set it to anything higher than 1000 steps per second. Below are the listed max speeds for each microstep resolution:
<ul style="list-style-type: none"><li>• 1 microstep --&gt; 1000 steps/sec = 10 mm/sec</li><li>• 1/2 microstep --&gt; 1000 steps/sec = 5 mm/sec</li><li>• 1/4 microstep --&gt; 1000 steps/sec = 2.5 mm/sec</li><li>• 1/8 microstep --&gt; 1000 steps/sec = 1.25 mm/sec</li><li>• 1/16 microstep --&gt; 1000 steps/sec = 0.625 mm/sec</li></ul>

Figure 11. Steps-to-MM conversion for the Z-axis stepper motor

For the mirror motor, we found that  $\frac{1}{8}$  microstep resolution offered the best balance between precision and speed. This step resolution is set at initialization and remains unchanged throughout the rest of the printing process.

### 3.1.4.3. Sensors

The printer system utilizes three types of sensors: one optical limit switch (Omron EE-SX674-WR 1M), two hall-effect sensors (DRV5055A4), and one wifi module (ESP8266 ESP-01).

#### *Optical Limit Switch*

The optical limit switch determines whether the base plate has lowered to “home” position. Homing is required at the start of every print and to level the base plate during printer calibration, as described in (Section 4.2: Leveling the Build Plate). The optical limit switch is shaped as a T-shaped slot with an incident light that exists between the two terminals. When this incident light is disrupted, the optical switch changes its output voltage. By reading this voltage output via the Arduino, we can determine whether the base plate has lowered to “home” by thresholding this analog signal.

#### *Hall-Effect Sensors*

Hall-effect sensors measure the magnetic field of its surrounding environment. They are often used to detect the proximity, speed, or displacement of a mechanical system. We chose to use hall-effect sensors due to its precision in conducting non-contact measurements.

The angular position of the mirror when directing the UV light and the visible light into the resin tank, is controlled by a magnet mounted on the bottom of the mirror and two hall-effect sensors. As the mirror and magnet (rotated via the stepper motor) get closer to the hall-effect sensors located on either side, each sensor’s output voltage changes. By reading this voltage output via the Arduino, and thresholding this analog signal, we can determine when to stop the mirror.

To ensure the reliability of the hall-effect sensors in positioning the mirror at  $45^\circ$  during each rotation, we rotated the mirror between the UV light and the visible light 200 times. The angular position of the mirror after the first and 200th rotation was then measured. No error was found.

#### *WiFi Module*

The ESP8266-01 is a SOC (System On a Chip) module that allows microcontrollers to access the WiFi network. Since our Arduino does not have built-in internet capabilities of its own, we connect this WiFi module externally via Arduino’s TX/RX pins. This enables the Arduino to connect with the remote web server, described in (Section 3.2.4: Web Server).

Note that because the Arduino interfaces with two devices (WiFi module and Processing), it needs two separate UART communication channels. Using the Uno will require software serial to enable virtual RX/TX pins (since it only has one UART channel), whereas the Mega has multiple pin pairs of RX/TX that can be directly used. We chose to use the Arduino Mega.

#### 3.1.4.4. Power

To control the availability of power to the printer system, we route all high-voltage devices to two IoT power relays (Adafruit Controllable Four Outlet Power Relay Module v2). When the relay is off, all power to the main printer components is cut off. These relays act as emergency stop switches that can be programmatically controlled via the Arduino.

## 3.2. Software

### 3.2.1. Blender

For the user interface, we developed a Blender add-on that provides a way for users to control different steps of the printing process using only one software application. The main functionalities of this user interface include:

- importing and editing 3D models for printing (ie. adding texture, generating supports, scaling)
- slicing the model to generate layer-by-layer UV and RGB projection images
- directly controlling the 3D printer itself.

Dimitri Tskhovrebadze developed the Blender add-on, Isabel Qamar guided the high-level design, and I implemented the backend software infrastructure for Blender, Processing, and Arduino communication. The following sections provide an overview of the resulting user interface developed for our project.

#### 3.2.1.1. Preparing the Model

The user interface consists of “Chromo Editor” and “Chromo Support” panels located to the left of the main workspace. On pressing “RESET SCENE”, an example 3D model of a black and blue checkerboard-textured surfboard is loaded into the workspace. The ground plane and the corner plane (with white and grey checkerboard textures) represent the available 3D printing dimensions. For performance purposes, the unit scale is 1000:1, which means 1000mm (in Blender) = 1mm (in real world). Models can be imported, scaled, and deleted as needed. The default workspace is displayed in Figure 12.

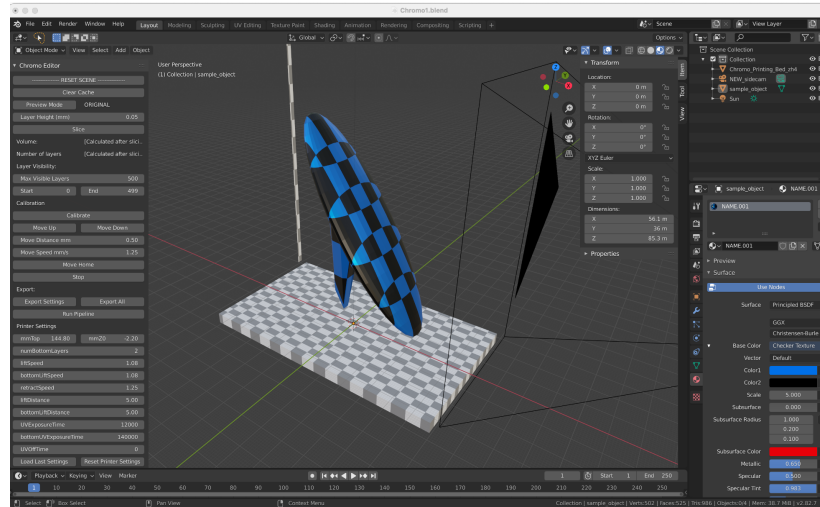


Figure 12. Blender user interface

Users can slice the model by adjusting the “Layer Height” (mm) and then selecting “Slice” under “Chromo Editor”. This generates a preview of the sliced model. Users can inspect individual layers by varying the “Layer Visibility” settings. Additionally, on “Slice”, the number of layers and expected volume of the printed model are calculated and displayed. The number of layers can be used by the user to predict expected printing time, and the volume can be used to determine how much resin to use. An example of the sliced surfboard is depicted in Figure 13.

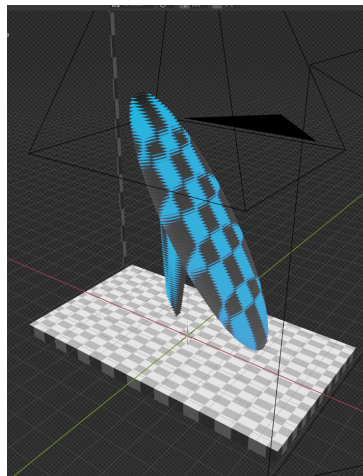


Figure 13. Sliced model

For models with large overhangs, supports can also be generated by selecting “Generate Support” under the “Chromo Support” section. Supports can be modified (ie. density, thickness, angle) or deleted as necessary. The red supports are shown in Figure 14.



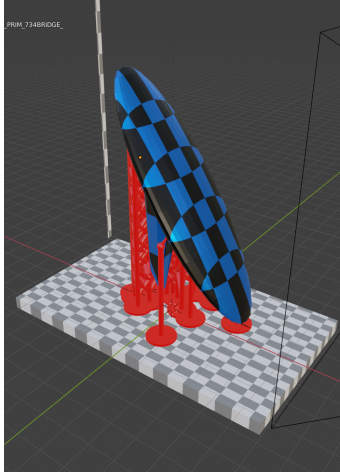


Figure 14. Support generation

Because the color gamut creatable using the available photochromic dyes is limited, the user can use “Preview Mode” to preview how the model will actually look after it is printed. When set to ORIGINAL, the object is displayed in true colors. When set to FINAL, the object colors remap to show what it will look like after printing. An example of the color preview applied to the surfboard can be seen in Figure 15.

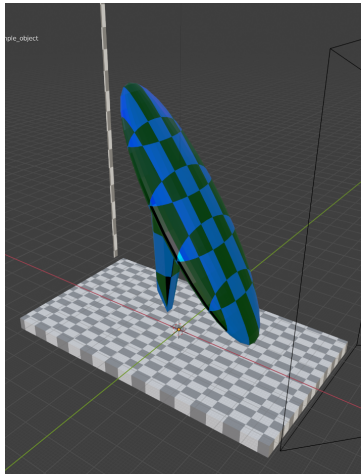


Figure 15. Color preview

By default, textures are applied top-down through each layer. This means that color can only be propagated vertically, meaning all layers have the same texture applied top-down. To enable objects to also be colored horizontally, users have the option to apply textures from the side view, thereby allowing every layer generated to have a different texture. This is useful for users who want to print multi-color objects that vary along the Y-axis. Specifically, this can be used for the proposed experiment discussed in (Section 6.2: Evaluating the Effect of Visible Light through Thickness). To change this projection view, a separate `side-mapper.py` script must be run

after exporting the images. An example of a textured model with a comparison of the resulting layer images produced (depending on the projection view) is shown in Figures 16-17.

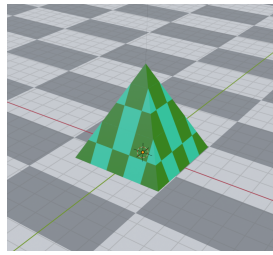


Figure 16. Textured 3D pyramid model

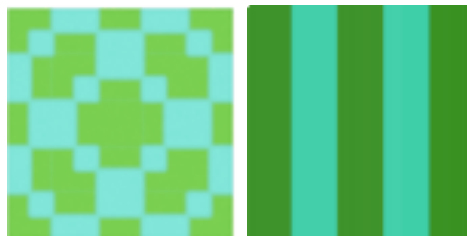


Figure 17. Texture projection views  
Top-Down (left) vs Side Projection (right)

#### 3.2.1.2. Exporting Images

After users are done editing their models, they can generate the layer-by-layer projection images by selecting “Export All” located under the “Export” section of the “Chromo Editor”. Paired black-and-white images (for the UV projector) and colored images (for the visible light projector) are generated for each layer. All files are exported to the Processing data folder that was specified by the user during the installation of the Blender add-on. Example .png images generated for the pyramid model (from Figure 16 above) is shown in Figure 18 below.

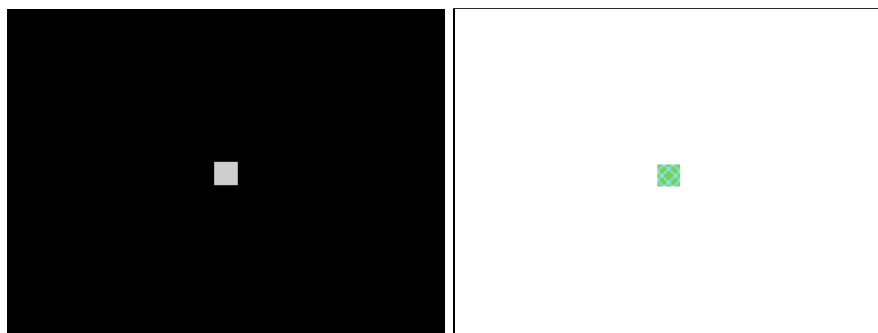


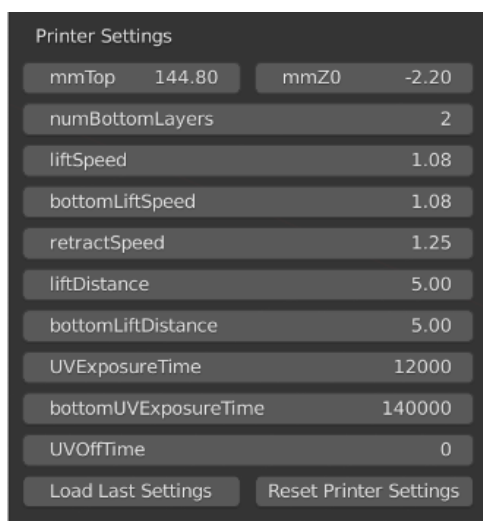
Figure 18. Layer image generation  
UV (left) and RGB (right) projectors



### 3.2.1.3. Exporting Settings

Users can also specify the settings used during the actual print process. We modeled our print settings after commonly found settings in professional slicers such as Ultimaker Cura and ANYCUBIC's Photon Workshop.

“Export Settings” generates a `printSettings.json` file that is later used by Processing at the start of the print. “Export Settings” exports *only* this settings file, while “Export All” exports *both* this settings file AND the sliced UV/RGB .png images. The separate “Export Settings” button was created for users who only want to update the printer settings, without having to wait for all the layer images to be sliced and generated again. All units are in either mm or sec. The default printer settings are shown in Figure 19.



The image shows a dark-themed window titled "Printer Settings". It contains a list of settings, each with a label and a value in a text input field. At the bottom, there are two buttons: "Load Last Settings" and "Reset Printer Settings".

Setting	Value
mmTop	144.80
mmZ0	-2.20
numBottomLayers	2
liftSpeed	1.08
bottomLiftSpeed	1.08
retractSpeed	1.25
liftDistance	5.00
bottomLiftDistance	5.00
UVExposureTime	12000
bottomUVExposureTime	140000
UVOffTime	0

Figure 19. Printer Settings

### 3.2.1.4. Printer Control

Users can also control the 3D printer directly via the Blender user interface. Specifically, the 3D printer has four modes: Menu, Calibrate, Move Home, and Run Pipeline. Whenever a mode changes, the new mode is rewritten to `mode.json` by Blender. This `mode.json` file is then read by Processing to update its internal printer state. The current mode can be changed by selecting “Stop” (Menu mode), “Calibrate” (Calibrate mode), “Move Home” (Move Home mode) or “Run Pipeline” (Run Pipeline mode). The printer controls available are shown in Figure 20.

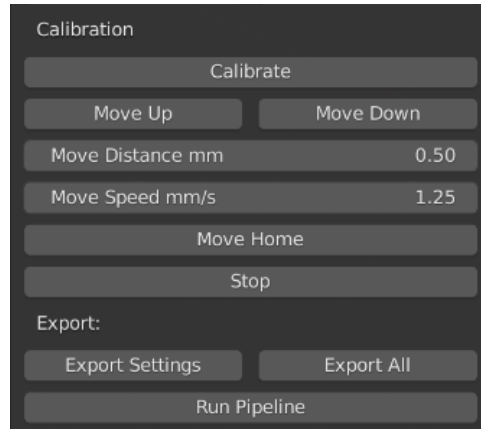


Figure 20. Printer control buttons

### *Menu Mode*

By default, the printer is in Menu mode. In this mode, the printer is awaiting commands from the user. The user can use this time to edit the model or update/generate settings and images. To return to Menu mode at any time, select “Stop” under the “Calibration” section.

### *Calibrate Mode*

To change the printer to Calibrate mode, select “Calibrate” under the “Calibration” section. In this mode, users can move the build plate up and down to a desired distance (mm) at the specified speed (mm/sec). The `move.json` file is updated each time the user presses a button under the “Calibration” section. Processing then reads this `move.json` file so that it can send the appropriate control commands to Arduino.

### *Move Home Mode*

To tell the printer to move the base plate to home, select “Move Home” under the “Calibration” section. In this mode, the printer will lower the base plate until the optical limit switch is triggered. This is useful for resetting the printer to a known position.

### *Printing Mode*

To start the printing process, change the printer to Run Pipeline mode by selecting “Run Pipeline” under the “Export” section. This tells Processing to start the print using the settings and files generated from the previous steps. The printing process can be stopped at any time by selecting “Stop”.

#### 3.2.1.5. Prioritizing “Stop”

For safety purposes, the “Stop” button has top priority over all other printer commands. “Stop” automatically overrides any processes currently being run on the printer. When “Stop” is

selected, `mode.json` is immediately rewritten, which updates Processing, and consequently tells Arduino to stop all actuation by cutting power to the motors. This priority queue was designed so that users can halt undesirable actions performed by the 3D printer at any time, regardless of action performed.

### 3.2.2. Processing

Processing acts as the “central management unit” of the system. It is responsible for keeping track of both the internal printer state and for communicating between Blender, Arduino, the UV projector, and the visible light projector. Processing reads user inputs from Blender, and then proceeds to send the appropriate commands to the Arduino, which in turn controls the actuators and sensors of the 3D printer. Processing is also responsible for projecting layer images to each projector during prints. An overview of Processing’s control endpoints is shown in Figure 21.

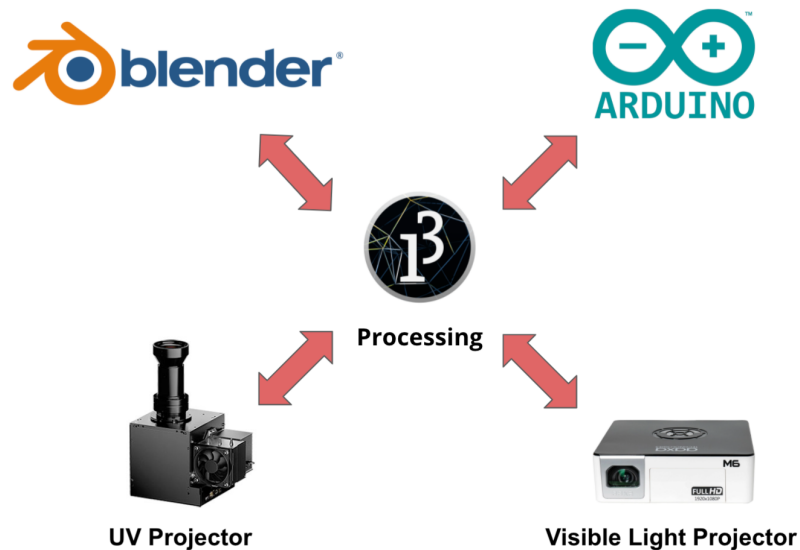


Figure 21. Processing system control

#### 3.2.2.1. State Machine

Processing has four modes of operation: Menu, Move Calibrate, Move Home, and Run Pipeline. These modes correspond directly with Blender’s four modes, described in (Section 3.2.1.4: Printer Control). Processing determines which mode to execute by reading the `mode.json` file, which is updated by Blender based on user input. Different state transitions are executed depending on the mode selected. The complete Processing state machine is depicted in Figure 22. Corresponding state descriptions from the source code is shown in Figure 23.

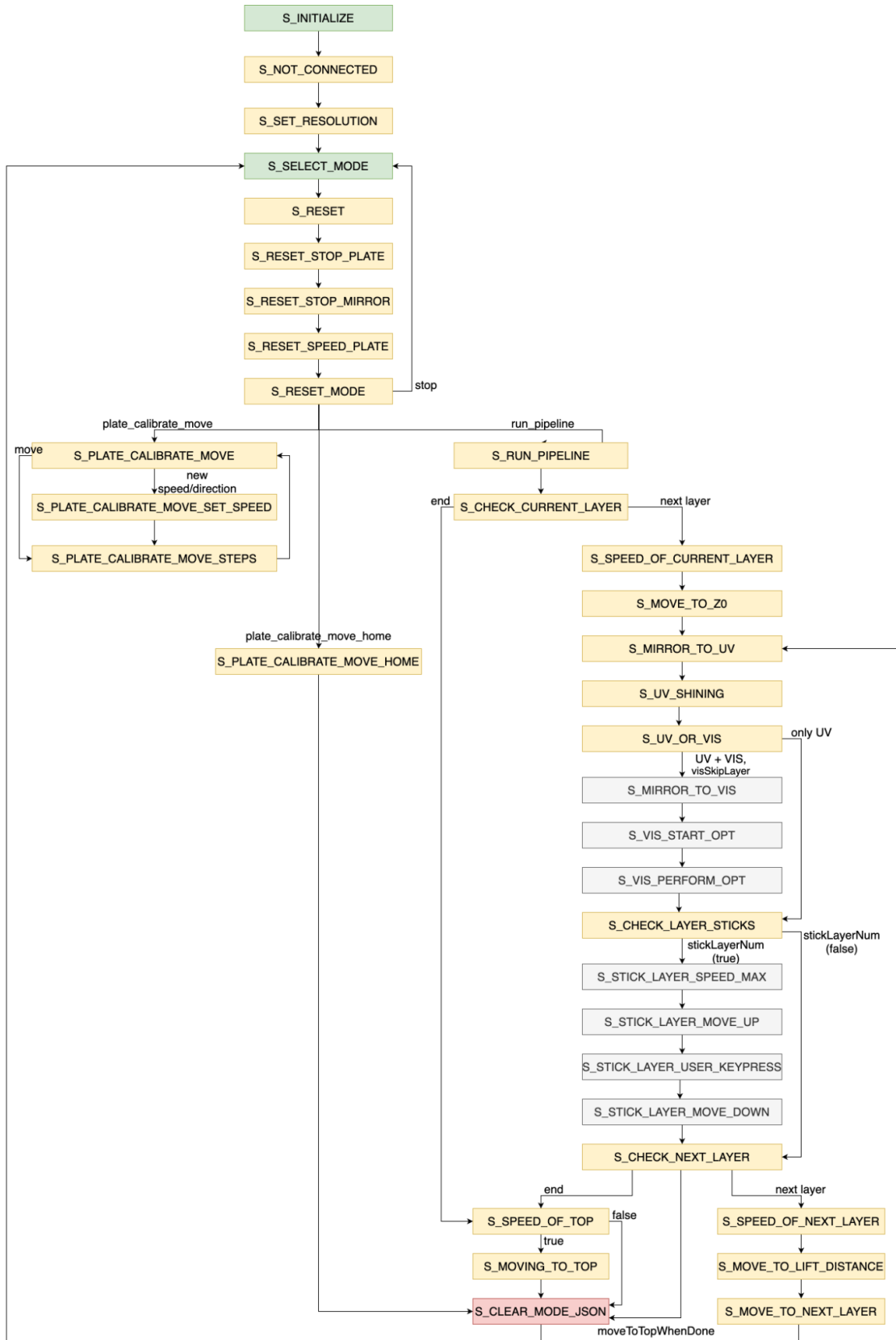


Figure 22. Processing state machine

```

// connection states
final String S_INITIALIZE = "[State] S_INITIALIZE: getting resolution settings (get numLayers and printSettings)";
final String S_NOT_CONNECTED = "[State] S_NOT_CONNECTED: trying to connect to arduino";
final String S_SET_RESOLUTION = "[State] S_SET_RESOLUTION: default homingSpeed; send plate speed resolution";

// reset states
final String S_RESET = "[State] S_RESET: resetting system; send plate stop";
final String S_RESET_STOP_PLATE = "[State] S_RESET_STOP_PLATE: stopping plate, send mirror stop";
final String S_RESET_STOP_MIRROR = "[State] S_RESET_STOP_MIRROR: stopping mirror";
final String S_RESET_SPEED_PLATE = "[State] S_RESET_SPEED_PLATE: resetting up/down speeds to MAX";
final String S_RESET_MODE = "[State] S_RESET_MODE: updating mode; start state machine with selected mode";

// user input states
final String S_SELECT_MODE = "[State] S_SELECT_MODE: waiting for user input in mode.json; send plate to home";
final String S_PLATE_CALIBRATE_MOVE = "[State] S_PLATE_CALIBRATE_MOVE: read move.json; send new calibrate speed (if applicable)";
final String S_PLATE_CALIBRATE_MOVE_SET_SPEED = "[State] S_PLATE_CALIBRATE_MOVE_SET_SPEED: changing speed; send plate up/down";
final String S_PLATE_CALIBRATE_MOVE_STEPS = "[State] S_PLATE_CALIBRATE_MOVE_STEPS: moving plate";
final String S_PLATE_CALIBRATE_MOVE_HOME = "[State] S_PLATE_CALIBRATE_MOVE_HOME: moving plate to home";
final String S_CLEAR_MODE_JSON = "[State] S_CLEAR_MODE_JSON: set mode.json to empty string";

// run_pipeline states
final String S_RUN_PIPELINE = "[State] S_RUN_PIPELINE: moving plate to home; send retractSpeed";
final String S_CHECK_CURRENT_LAYER = "[State] S_CHECK_CURRENT_LAYER: changing speed; (set liftDistance/uvExposureTime; send upSpeed) OR (send topOutSpeed)";
final String S_SPEED_OF_CURRENT_LAYER = "[State] S_SPEED_OF_CURRENT_LAYER: changing speed; send plate (up OR down) to mmz0";
final String S_MOVE_TO_Z0 = "[State] S_MOVE_TO_Z0: moving plate to mmz0; set initial currentDistance, send mirror to UV";

final String S_MIRROR_TO_UV = "[State] S_MIRROR_TO_UV: moving mirror to uv; set UVImage to layernum, turn on UV";
final String S_UV_SHINING = "[State] S_UV_SHINING: shining UV light; set UVImage to black, turn off UV";
final String S_UV_OR_VIS = "[State] S_UV_OR_VIS: (go to next uv layer) OR (send mirror to VIS)";

final String S_CHECK_LAYER_STICKS = "[State] S_CHECK_LAYER_STICKS: check if sticklayer => (send up/down speed to MAX) or (go to check next layer)";
final String S_STICK_LAYER_SPEED_MAX = "[State] S_STICK_LAYER_SPEED_MAX: resetting up/down speeds to MAX; move plate up to stickLayerMM";
final String S_STICK_LAYER_MOVE_UP = "[State] S_STICK_LAYER_MOVE_UP: moving plate up";
final String S_STICK_LAYER_USER_KEYPRESS = "[State] S_STICK_LAYER_USER_KEYPRESS: waiting for user keypress 'n'; move plate down stickLayerMM";
final String S_STICK_LAYER_MOVE_DOWN = "[State] S_STICK_LAYER_MOVE_DOWN: moving plate down";

final String S_CHECK_NEXT_LAYER = "[State] S_CHECK_NEXT_LAYER: increment layer number; (set liftDistance/uvExposureTime; send upSpeed) OR (send topOutSpeed)";
final String S_SPEED_OF_NEXT_LAYER = "[State] S_SPEED_OF_NEXT_LAYER: changing speed; send plate to liftDistance";
final String S_MOVE_TO_LIFT_DISTANCE = "[State] S_MOVE_TO_LIFT_DISTANCE: moving plate to liftDistance; send plate to layerHeight";
final String S_MOVE_TO_NEXT_LAYER = "[State] S_MOVE_TO_NEXT_LAYER: moving plate to layerHeight; update currentDistance, send mirror to UV";

final String S_SPEED_OF_TOP = "[State] S_SPEED_OF_TOP: changing speed; send plate to top";
final String S_MOVING_TO_TOP = "[State] S_MOVING_TO_TOP: moving plate to top; go to main menu";

final String S_MIRROR_TO_VIS = "[State] S_MIRROR_TO_VIS: moving mirror to VIS; send start opt";
final String S_VIS_START_OPT = "[State] S_VIS_START_OPT: waiting for start opt return; perform opt";
final String S_VIS_PERFORM_OPT = "[State] S_VIS_PERFORM_OPT: shining animation; go to next layer";

```

Figure 23. Processing state descriptions

## Initialization

On initialization, Processing establishes connection with the Arduino (`S_INITIALIZE`, `S_NOT_CONNECTED`). After connection has been established, Processing resets the motors to a known state (`S_SET_RESOLUTION`).

## Menu Mode

After initialization, Processing will then proceed to idle until a mode is selected (`S_SELECT_MODE`). This is our Menu mode. No new action is taken by Processing until the user selects a new mode in Blender and the `mode.json` file is updated.

## Resetting Between Modes

When transitioning between modes, Processing conducts a reset procedure, in which all actuation is halted and speed settings are cleared (`S_RESET`, `S_RESET_STOP_PLATE`, `S_RESET_MIRROR`, `S_RESET_SPEED_PLATE`, `S_RESET_MODE`). This reset procedure brings the motor settings back to a known state, and is necessary to ensure that actuation commands are not unintentionally propagated between modes. Additionally, when a mode is successfully executed, the `move.json` file is cleared to communicate completion (`S_CLEAR_MODE_JSON`).

### *Move Calibrate*

When Processing reads `plate_calibrate_move` from `mode.json`, it transitions to Move Calibrate mode (`S_PLATE_CALIBRATE_MOVE`). In this mode, users can either update the speed (`S_PLATE_CALIBRATE_MOVE_SET_SPEED`) or trigger the build plate to move a specified distance (`S_PLATE_CALIBRATE_MOVE_STEPS`).

### *Move Home*

When Processing reads `plate_calibrate_move_home` from `mode.json`, it transitions to Move Home mode (`S_PLATE_CALIBRATE_MOVE_HOME`), in which the build plate is lowered to home position. After the build plate has homed, the mode returns to Menu mode.

### *Run Pipeline*

When Processing reads `run_pipeline` from `mode.json`, it transitions to Run Pipeline mode (`S_RUN_PIPELINE`). In this mode, the main printing process is executed.

Our printing process works similar to existing color inkjet printing processes such as the da Vinci printer. The printer first creates the geometry of the layer and then applies the color. After a layer has been colored, the build plate raises, and the next layer is printed and colored. This process repeats until the entire object is built and colored. The specific state transitions are detailed below.

Before starting the print, layer images, settings, and desaturation times must have already been generated by Blender, detailed in (Section 3.2.1: Blender). On starting Run Pipeline mode, Processing generates its own build instructions for the printer based on the information input from Blender (ie. the layer height, the bottom layer exposure time, the layer exposure time, and the retraction speed). To start the print, the build plate first moves to home position (`S_MOVE_TO_Z0`) to calibrate its position. The mirror then rotates to direct the UV light into the resin tank (`S_MIRROR_TO_UV`) and then projects the first black-and-white image (`S_UV_SHINING`) to cure the first layer of the object. The mirror then rotates to direct the RGB light into the resin tank (`S_MIRROR_TO_VIS`), and shines a light pattern (`S_PERFORM_OPT`) determined by the algorithm specified in (Section 3.2.2.2: Desaturation Algorithm) to color the object. This process repeats for each layer, with settings continuously checked and updated throughout the rest of the print (`S_CHECK_CURRENT_LAYER`, `S_CHECK_NEXT_LAYER`). When the print has finished, the build plate moves to the top of the printer (`S_SPEED_OF_TOP`, `S_MOVING_TO_TOP`), so that the print can be removed.

One failure mode is when the printed object does not adhere to the build plate at the beginning of the print. To ensure that the print successfully adheres to the build plate (`S_CHECK_LAYER_STICKS`), we have added a feature that allows the user to check the print after the 3rd layer has been printed, in which the build plate will move to the top of the printer

and wait for user keypress confirmation (`S_STICK_LAYER_MOVE_UP`, `S_STICK_LAYER_USER_KEYPRESS`) before continuing.

#### 3.2.2.2. Desaturation Algorithm

Colored images generated by Blender are imported into Processing, which then converts each colored layer image into a series of RGB projection images for the visible light projector. To generate these projection images, Processing runs a gradient descent optimization algorithm with memoization, based on that of *Photo-Chromeleon* [1], to calculate the desaturation times required for each of the projector's R, G, B color channels. These RGB desaturation times are then saved into a hashmap to be referenced and applied later during visible light projection steps. The desaturation algorithm is run only once at the start of every print. To develop this algorithm, we had to conduct experiments to determine the desaturation times of each dye when exposed to individual RGB light input from the projector. This experiment is described in (Section 6.1: Dye Desaturation Times).

#### 3.2.2.3. Communicating between Processing and the Projectors

Both the UV projector and visible light projector are powered by its own external power source and are controlled via HDMI input. The HDMI input value can be referenced as display numbers in Processing (ie. Display 1 = UV projector, Display 2 = visible light projector).

To project images to the desired projectors in Processing, we can simply draw the image to the correct display number via Processing's built-in `fullScreen()` function. The images projected to each display can also be individually modified (ie. rotate, scale, translate). This is useful for projector calibration, which is described in (Section 4.1: Projector Alignment).

#### 3.2.2.3. Communicating between Processing and Blender

Processing and Blender communicate by reading and writing files. To prevent read-and-write conflicts between the two applications, `processing_ready` and `blender_ready` files are generated whenever either application is accessing a file. For example, when Processing is reading or writing to a file, it generates an empty file named `processing_ready` to prevent Blender from opening files within the same directory. The same procedure occurs vice versa whenever Blender must also read or write to a file.

#### 3.2.2.4. Communicating between Processing and Arduino

Arduino and Processing communicate with each other via serial port communication. Specifically, Processing reads from Arduino's serial port. Before sending data, Arduino and Processing must establish connection via a digital "handshake".

Since serial ports can only be read by one device at a time, Processing also acts as Arduino's "serial monitor". Meaning that any print-statements sent by Arduino to the serial port, is also consequently read and printed out by Processing.

Debug statements can also be sent, but are not acted upon as actionable state triggers. To differentiate between debug statements and commands, a prefix is placed before each debug statement. Specifications are as follows:

- `[Processing]` debug for Processing
- `[State]` debug for Processing's current state
- `[Arduino]` debug for Arduino
- `[ESP8266]` debug for ESP8266-01
- If a message is not preceded by a bracket description, it is a command sent from either Processing or Arduino

Processing sends actuation commands to Arduino via the serial port. These commands are then executed by Arduino to control the 3D printer. When Arduino completes an action (ie. motor finishes moving), it will send a "done" command to Processing to signify that the command has been successfully executed. This triggers a state change in Processing's internal state machine, effectively moving the printing process forward.

### 3.2.3. Arduino

Arduino is responsible for communicating with the sensors and actuators. The microcontroller controls two stepper motors (via drivers), two hall-effect sensors, two IoT relays, one optical limit switch, and one wifi module.

#### 3.2.3.1. Commands

We developed a basic library for sending commands to Arduino. Commands can be sent through *either* the Serial Monitor or Processing (but not both at once), and are received via Arduino's serial port. The Serial Monitor is used mostly for debugging purposes, while the integrated system and user interface relies on Processing. The commands available are listed below. Note that input arguments are capitalized and bolded.

To establish connection with Processing:

- `processing_ready`

To change the current mode:

- `mode_changed, MODE`

To move the build plate (z-axis stepper motor):



- `plate_move_up`
- `plate_move_down`
- `plate_home`
- `plate_stop`
- `plate_move_up_specific_steps, STEPS`
- `plate_move_down_specific_steps, STEPS`

To change the build plate speed settings (z-axis stepper motor, steps/sec):

- `plate_motor_change_speed_up, SPEED`
- `plate_motor_change_speed_down, SPEED`
- `plate_motor_change_speed_both, SPEED`

To rotate the mirror (mirror stepper motor):

- `mirror_move_uv`
- `mirror_move_vis`
- `mirror_stop`

Note that stepper motor commands take in `steps` as input, not mm. Therefore, mm-to-step conversion must be done in Processing before motor commands can be sent to Arduino.

### 3.2.4. Web Server

To enable remote observation and operation of the 3D printer, specifically during long print jobs and for emergency stop procedures, we developed a web server, hosted on Heroku, that includes both a frontend user interface and a backend API. This web server acts as an IoT intermediary that stores, updates, and propagates 3D printer state information to relevant device endpoints.

The web server URL can be found here: <https://photochromic-3d-printing.herokuapp.com/>

#### 3.2.4.1. Database

The web server uses ClearDB, which is a cloud service for running MySQL applications on Heroku. This database acts as persistent storage to keep track of real-time 3D printer states. The type of information stored in the ClearDB database is shown in Figure 24.

```
"printerKey": "HCIE",
"power": 0,
"mode": "n/a",
"printNumLayers_total": 100,
"printNumLayers_current": 50,
"printMinutes_total": 300,
"printMinutes_left": 150,
"timeUpdated_arduino": "2021-01-03T08:20:12.000Z",
"timeUpdated_processing": "2021-01-03T08:28:41.000Z",
"emergencyStop": 0,
"emergencyStopTime": "2021-01-03T08:20:19.000Z"
```

Figure 24. Database variables

### 3.2.4.2. API

The backend API is scripted in Node and follows REST API guidelines. To update the database, device clients can send the following API requests to the server, shown in Figure 25.

URL	HTTP Verb	Action
/api/printer	GET	get printer status
/api/printer/power	POST	set power ON
/api/printer/power	DELETE	set power OFF
/api/printer/stop	POST	set emergencyStop to true; set emergencyStopTime
/api/printer/stop	DELETE	set emergencyStop to false; set power OFF
/api/printer/arduino	PUT	set arduino time; set power ON
/api/printer/processing	PUT	set processing status
/api/printer/processing/mode	PUT	set processing mode

Figure 25. REST API

### 3.2.4.3. Website

The frontend user interface is designed via React. The website is a simple GUI with an emergency stop button, and a display of current printer state information. The type of state information displayed is similar to that provided in commercial 3D printers (ie. total layers, current layer, amount of time left, etc).

The button is red (active) when the 3D printer is powered on. The button is greyed out (inactive) when the 3D printer is powered off. A pop-up display asking for user confirmation is required when the POWER OFF button is selected in order to prevent accidental misclicks. All times are displayed in EST. Screenshots of the website are shown in Figure 26.

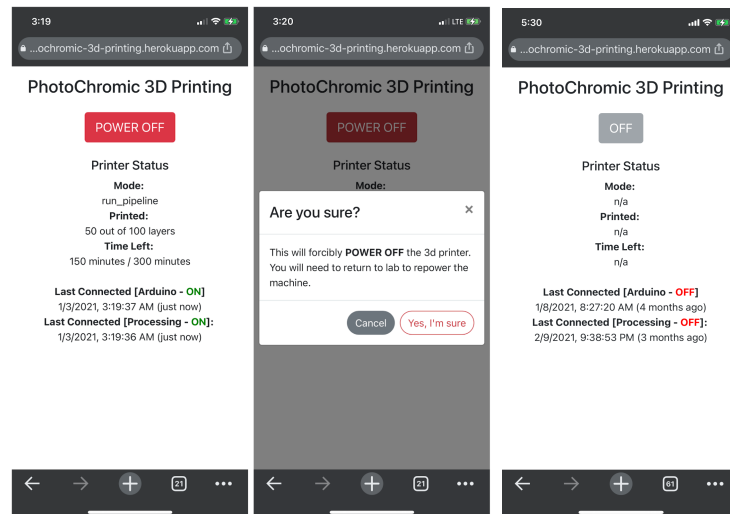


Figure 26. Website GUI for remote operation  
Printer ON (left), Emergency stop pop-up confirmation (middle), Printer OFF (right)

#### 3.2.4.4. Updating Printer State

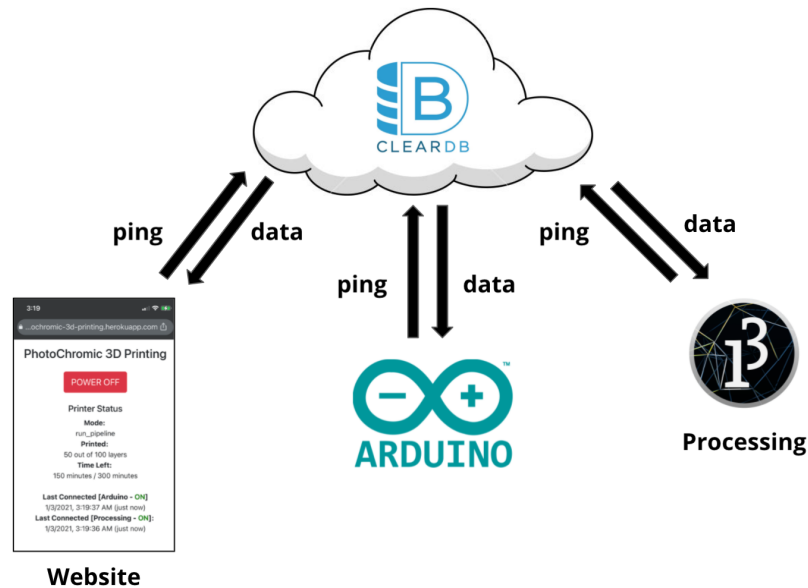


Figure 27. Updating the database

The web server updates the printer state based on information gathered from three separate applications: the web server itself, Arduino, and Processing. State information is updated via API requests, which require internet connection from each of the three applications.

The web server uses Arduino's last connection time to determine whether the 3D printer is currently ON or OFF. Because Arduino controls the 3D printer, if the Arduino is powered on, we assume the 3D printer is also powered on. If the web server has not received a ping from Arduino within the last 30 seconds, the 3D printer is deemed to be OFF, and the UI is updated accordingly on the website. When a user selects the POWER OFF button on the GUI, an emergency stop request is set through the API, which triggers the shutdown procedure for both Arduino and Processing.

When powered on, Arduino pings the web server via its wifi module every 5 seconds, which updates the "last connected" time of the Arduino in the database. Simultaneously, Arduino also requests state information every 5 seconds to determine if an emergency stop command has been triggered. If an emergency stop is triggered, Arduino immediately signals the power relay to shut down, which cuts power to all electrical components of the 3D printer system. This effectively prevents any processes from continuing to run on the printer. For safety purposes, the printer cannot be powered on remotely. Thus, users must physically return to the lab to the printer.

When the main Processing sketch is run, Processing pings the web server every 5 seconds, which updates the "last connected" state of Processing in the database. Simultaneously, Processing also requests state information every 5 seconds to determine if an emergency stop command has been

triggered. If an emergency stop is triggered, Processing first sends a shutdown command to Arduino (in case Arduino is delayed or hasn't received the update yet), and then proceeds to exit out of the sketch itself. This effectively halts Processing's internal state machine, described in (Section 3.2.2.1: State Machine), so that it will no longer send commands to Arduino.

Note that from the time that the user selects the POWER OFF button, there is a minimum 30 second delay before the user will receive confirmation that the printer has been powered off. This is because the web server only deems the 3D printer to be OFF if it has not received a ping from the Arduino within the last 30 seconds. This built-in delay was to ensure that the number of requests combined from all our devices is below Heroku's API rate request limit of 4500 requests per hour. To notify the user of this delay, the power off button is greyed out (inactive) on the website, and the text is replaced with "Loading..." until 30 seconds has passed, in which it will return to the greyed-out OFF button state.

## 4. Calibration

Before starting a print, the UV projector, visible light projector, and base plate must be calibrated to ensure that the 3D printer executes properly. The following sections detail these calibration procedures.

### 4.1. Projector Alignment

The UV projector and visible light projector must be calibrated to ensure that the projected images align during prints. The size mapping from Blender to the real-world is also calibrated at this time, to ensure that the printer will actually print objects at the expected size specified in Blender. This calibration procedure should be done anytime a hardware component (ie. a projector, mirror, or printer) is physically moved.

To calibrate the visible light projector, first place a piece of paper with a centered 40mm square on top of the tempered glass screen. Rotate the mirror 45° so that the light from the visible light projector is directed to the tempered glass screen mounted above. Then, slice and export a 40mm<sup>3</sup> cube in Blender. Project any black-and-white layer image onto the tempered glass screen. Use `Projector.pde` to rotate, scale, and translate the projected image as needed so that it fits onto our drawn 40mm square. Repeat this procedure with the UV projector. Figures 28-29 show examples of this calibration process using 5mm and 40mm squares.

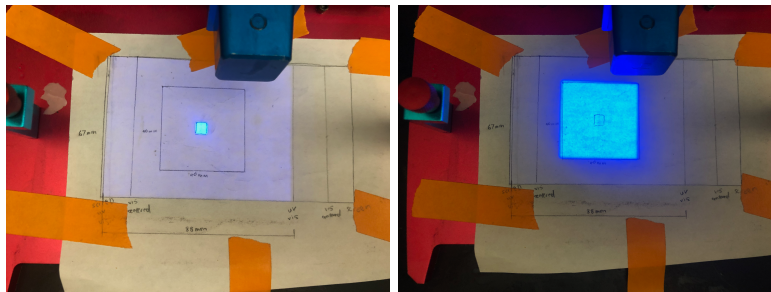


Figure 28. UV projector calibration  
5mm square (left) vs 40mm square (right)

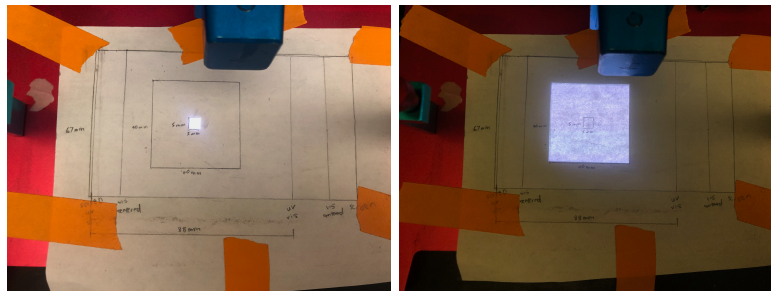


Figure 29. Visible light projector calibration  
5mm square (left) vs 40mm square (right)

## 4.2. Leveling the Build Plate

$z_0$  represents the Z-axis position of the build plate at the first layer.  $z_0$  ensures that the first few layers of the print will adhere properly to the build plate. If not calibrated correctly, rather than adhering to the build plate, the print will remain stuck inside the resin tank. To calibrate this  $z_0$  position, first remove the resin tank, then manually lower the base plate until it is approximately the thickness of a piece of paper above the tempered glass screen. Figure 30 shows an example of this leveling procedure.

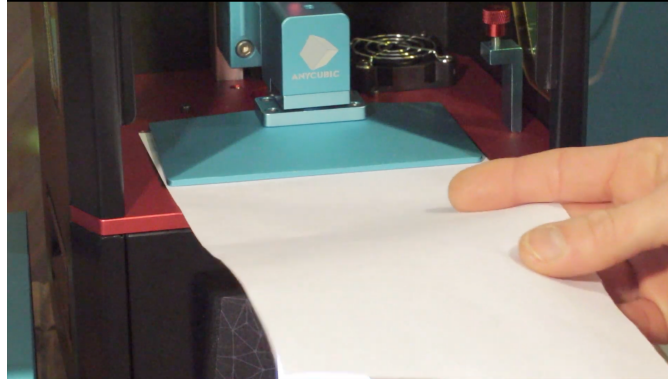


Figure 30. Leveling the build plate [11]

To manually lower the build plate, select “Calibrate” then “Move Down” under the “Calibration” section in Blender. Place a piece of paper between the build plate and tempered glass screen, then lower the build plate until moving the paper causes friction. At this point, select “Move Home” under “Calibration”. The distance  $z_0$  is away from Home (in mm) will be printed in the Processing console. Input this new value into “mmZ0” under “Printer Settings” in Blender. Then export these settings by selecting “Export Settings” under “Export”.

This  $z_0$  position does not need to be calibrated at the start of every print, but should be done throughout the use of the printer, or when prints start failing to adhere to the base plate.

## 5. Evaluation

Our multi-color photochromic resin consists of mixing resin with a ratio of cyan, magenta, and yellow photochromic ink. This ratio was determined via experimentation, with the goal of achieving the highest color saturation, while also ensuring minimal effect of the photochromic inks on the printing properties of the resin. The results of our formulation from our experiments are described in the following sections. The materials experiments discussed in this section were conducted by Isabel Qamar.

### 5.1. Developing the Photochromic Resin

We sourced White Resin from Formlabs as the base material. The photochromic dyes were sourced from Yamada Chemical Co [6]: cyan (DAE-0001), magenta (DAE-0012) and yellow (DAE-0068). The dyes are mixed into Ethyl Acetate (VWR International) to create the photochromic ink, and then mixed the resulting ink into the resin. We maintain the C:M:Y ink ratio of 1:1:3, based on that of *Photo-Chromeleon* [12], when mixed into the resin. An example of our photochromic resin is depicted in Figure 31 below.

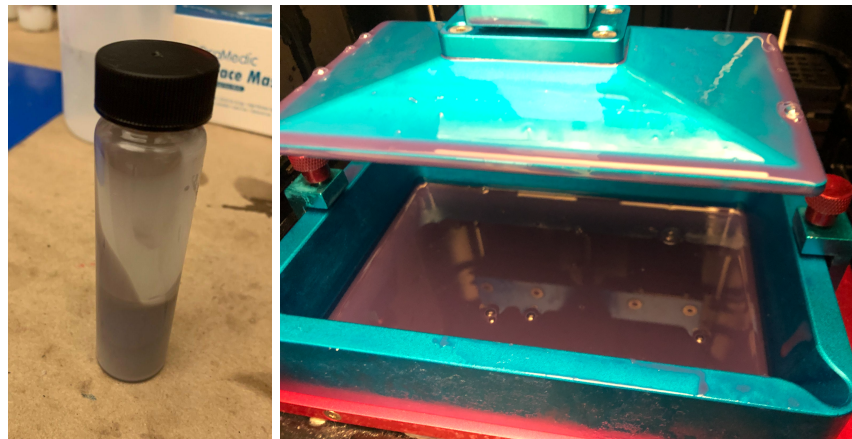


Figure 31. Photochromic resin  
Photochromic resin stored a vial (left) vs. poured in the resin tank (right)

### 5.2. Resin Curing Times

The optimal wavelength for curing the Formlabs White Resin is 405nm. To determine the exposure times necessary for each layer, we printed 40mm squares at 0.05mm layer thickness, and experimented with the bottom layer and normal layer exposure times until solid prints were created. The exposure times that worked best for our 3D printer system are as follows:



- The first and second layers of the resin should be exposed to UV light for 140 seconds to ensure that the print adheres to the build plate.
- All subsequent layers should be exposed to UV light for 10 seconds, which is sufficient to cure the resin.

When a print is completed, the print is placed in a light-sealed container of isopropanol, and washed for 30 minutes to remove any excess uncured resin from the outer surfaces of the object. The print is then dried. The resulting prints using the exposure times and layer settings specified above are shown in Figure 32 below.

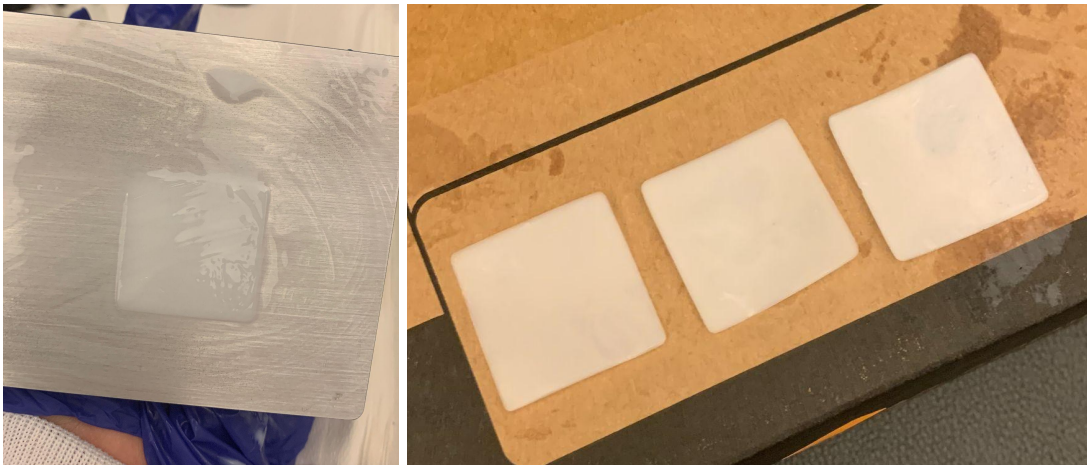


Figure 32. Cured prints  
Adhered to base plate (left), washed and dried (right)



## 6. Future Work

Below are proposed topics for future work. Because this research is still ongoing, our team has already begun development on a few of these action items. Thus, the following sections provide both an overview of each proposed topic as well as its current progress.

### 6.1. Dye Desaturation Times

The overall saturation of our photochromic resin can be programmatically controlled by projecting specific wavelengths of light onto this resin-dye mixture. To achieve a desired print color, we conducted experiments to determine the individual desaturation times of each dye. The results of these experiments will enable us to algorithmically calculate (1) what wavelengths to project onto our printed object, and (2) how long to project these wavelengths for, during our final printing process. We detail the procedures for these experiments below.

To determine the desaturation times of each dye on a single layer when exposed to RGB wavelengths from the visible light projector, we separately created magenta, cyan, and yellow photochromic resin mixtures according to the procedure outlined in (Section 5.1 Developing the Photochromic Resin), and cured the resin into 67x84mm rectangles at a layer height of 0.05mm, using the UV projector.

Once printed, we then shone all three RGB wavelengths onto the printed layer to quantify the effect of each of the projector's LEDs on each resin-dye mixture. The saturation level of each projected RGB color bar linearly decreased from top to bottom over time until it reached the edge of the projection area. The resulting desaturation times for each dye and color channel were then input into our optimization algorithm from (Section 3.2.2.2 Desaturation Algorithm) to calculate the desaturation times required for each of the projector's RGB color channels on each image pixel during the actual print. Figure 33 shows the desaturation bars projected.

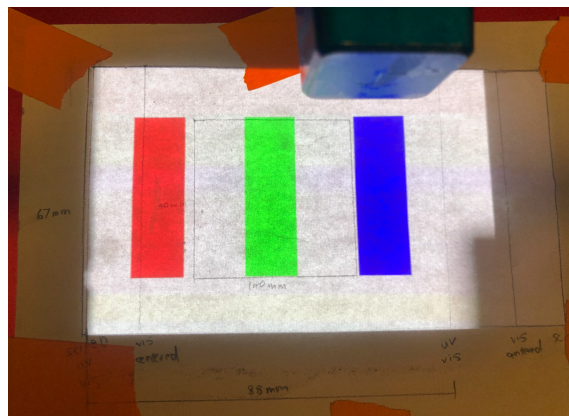


Figure 33. Projected desaturation bars

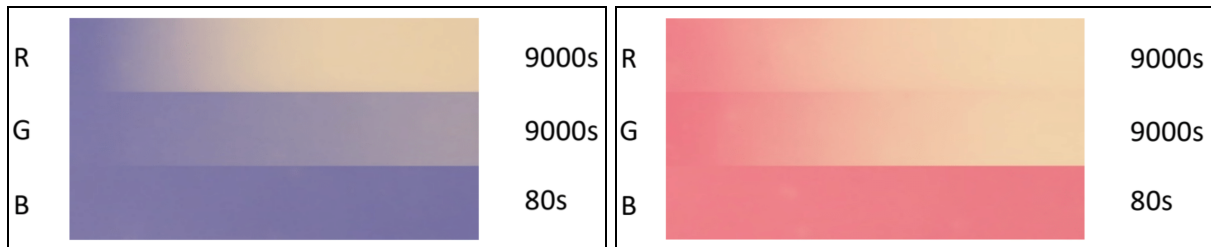


Figure 34. Printed desaturation bars  
cyan desaturation bars (left), magenta desaturation bars (right)

The resulting desaturation bars for the cyan and magenta prints, depicted above in Figure 34, show the relative saturation decrease over time per color channel. Due to the quick desaturation time of the yellow dyes, along with our project's time constraints, we were unable to get the printed desaturation bars for the yellow prints before the submission of this thesis, thus this is one item of future work moving forward.

## 6.2. Evaluating the Effect of Visible Light through Thickness

Since each layer of printed resin has some degree of transparency, we must evaluate how shining RGB light on a single layer affects the previously printed and colored layers. This data can then be accounted for during the coloring process of the previous layers, and incorporated into our final desaturation algorithm. Figure 35 provides conceptual drawing of this evaluation.

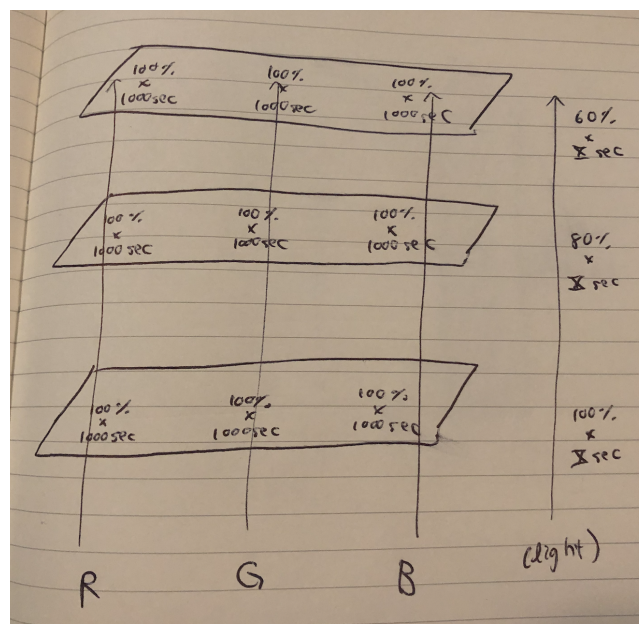


Figure 35. Conceptual drawing for effect of visible light through thickness

To characterize the vertical desaturation of color through the photochromic resin, we designed an experiment in which the printer would print three cube samples in cyan, magenta and yellow

photochromic resin individually. Once a cube has finished printing, the resin tank would be emptied, and the cube would be lowered so that the last printed layer was in contact with the bottom of the empty tank. RGB lights (in separate experiments) would be shown for 30 seconds on each cube from the bottom using the visible light projector, for a total of 9 separate experiments. The base plate would then be raised, and a picture of the color gradient would be taken on the side of the projector that shows the vertical desaturation of color through the photochromic resin. This data can then be analyzed and included into our final optimization algorithm, similar to (Section 6.1: Dye Desaturation Times).

A separate state machine that provides the controls for this evaluation was developed for this experiment, shown in Figure 36 below.

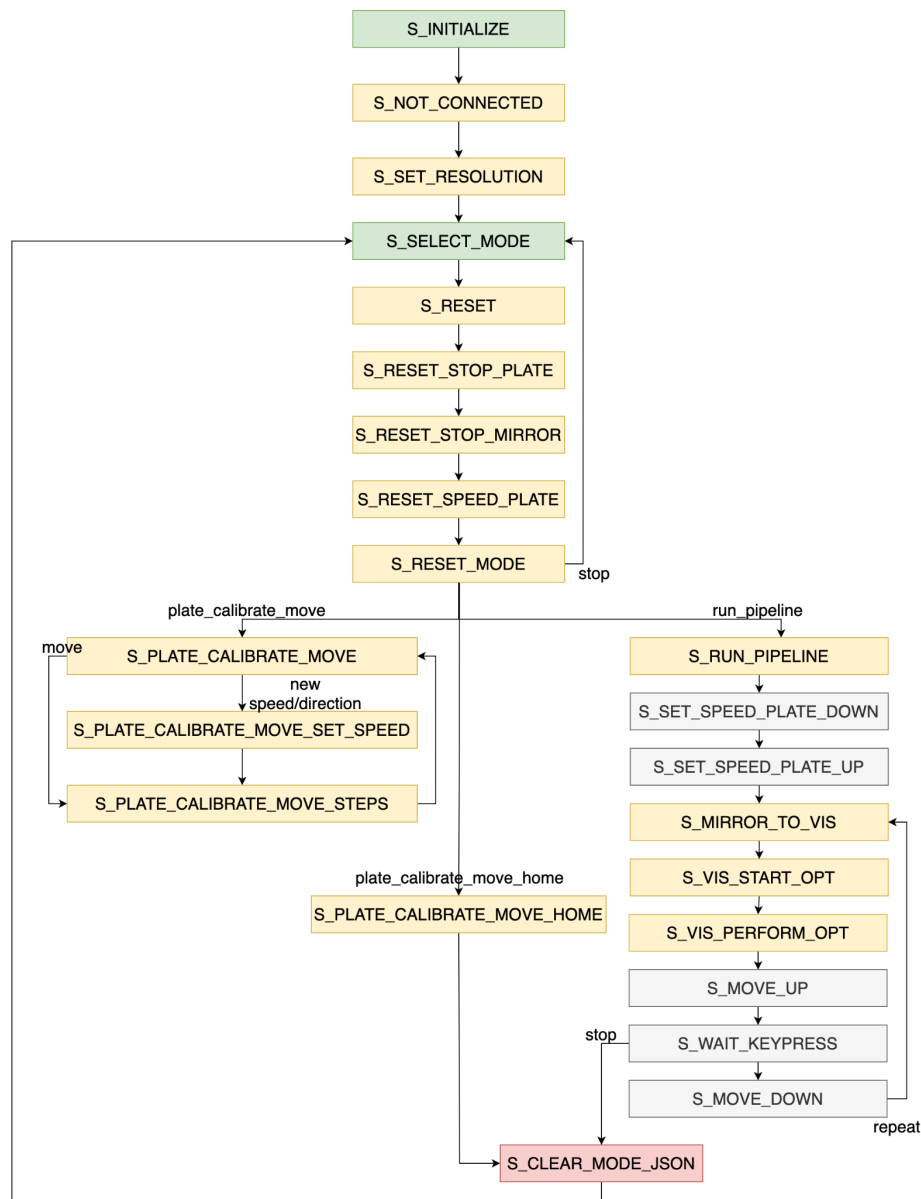


Figure 36. Processing state machine for experiment 2

### 6.3. Determining the Color Gamut

Due to the physical constraints of our dyes, the range of colors printable via our 3D printer system is constrained. This creates a discrepancy between desired colors (digital) and expected colors (physical). Thus, it is important to evaluate the color gamut of our resin, so that users can predict what colors their printed objects will ultimately be. We can evaluate the color gamut of our multi-color photochromic resin similar to the experiments in *Photo-Chromeleon* [1].

### 6.4. Conducting Larger Prints

The majority of the objects that we printed for this project were <20 layers thick (ie. <1mm). The reason is that many of our initial experiments (ie. resin curing times, dye desaturation times) required only thin-layer prints. Figure 37 shows a few examples of the colored prints we produced from our experiments during this time.

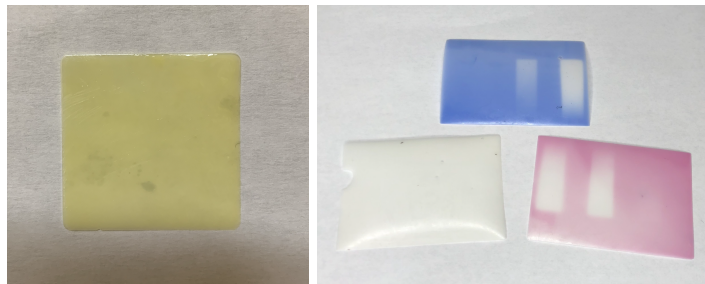


Figure 37. Example colored prints  
Yellow print (left), RGB prints after 24 hours of ambient light exposure (right)

While these colored prints are a valid proof-of-concept for our 3D printer system, larger objects should be printed to test the efficacy and usability of the system. Previously, we have attempted to conduct one larger print of a 5mm pyramid. This print ultimately failed due to the mirror motor overheating, thereby causing the mirror to shift near the end of the print, and causing the tip of the printed pyramid to become offset, depicted in Figure 38.

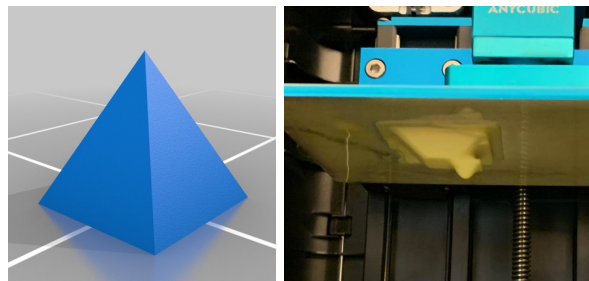


Figure 38. Large print example  
desired 3d model to print (left), actual print (right)

Moving forward, one unknown is whether the electronics would survive longer-lasting, multi-day prints. This is important to evaluate in order to determine the allowable size of future prints (ie. for application examples). The main reason that the 3D printer currently fails for longer prints is due to the electronics overheating (ie. stepper motor). To mitigate this issue, we have added heat sinks to the motors and included cooling fans for heat dissipation. Additionally, we have benchmarked the time required to print each layer via Processing, which can help us determine which parts of the print are taking the longest, and thereby develop ways to optimize these areas. Since stepper motors dissipate more heat the longer they are powered on, one idea for decreasing overall heat dissipation is to determine how the overall print time can be decreased. Suggested methods for how to decrease this print time are discussed in (Section 6.5 Decreasing Printing Time).

## 6.5. Decreasing Printing Time

The time taken to build and color an object is dependent upon several factors: the run time of the software, the time taken to cure a layer, the time taken to apply the texture onto the layer, and the total number of layers of the object.

From a hardware perspective, one large contributor to the long total print time is the current communication protocol. Three separate applications (Blender, Arduino, Processing) communicate with each other in our current 3D printer system. Through benchmarking the Processing sketch, we found that the communication times, specifically between Arduino and Processing via the Serial Port, average ~3-4 seconds per command. While this may not seem long individually, this time adds up for longer prints. However, because we are using off-the-shelf software, this delay in communication time is to be expected. One way to improve communication times is to decrease the overall number of software applications we are using, such as moving all logic into one low-level embedded system (ie. one device driver). Other methods to decrease printing time include: finding optimizations in the current communication protocol, removing extraneous steps in the printing process (ie. coloring every other layer, as opposed to every layer), determining and decreasing the time required to fully cure and color each layer, etc.

Overall, one large step we have already taken to decrease total print time was to replace the default Photon S UV projector with the higher-powered Formlabs UV projector (InVision Ikarus Full-HD DLP6500 light engine 385nm). This has hugely decreased the curing time required for each layer (ie. from 30 seconds per layer via the default Photon S UV projector, to 10 seconds per layer via the Formlabs UV projector). The addition of this modification along with the suggested optimizations detailed above for decreasing overall print times can further optimize the fabrication experience of our multi-color 3D printing system.

## 7. Conclusion

For this project, we implemented an end-to-end digital-to-print fabrication infrastructure for multi-color photochromic 3D printing. We also developed a new photochromic resin formulation that can be cured and colored under one fabrication procedure. To create this resin, we incorporate photochromic dyes into a UV-curable resin. By implementing an additional visible light projector into the printer, along with a higher-powered UV projector and a rotatable mirror, we can both cure and color each layer of an object as it is being printed.

In this thesis, we provided the implementation details and design decisions that went into building this integrated printing infrastructure. We discussed the results from the experiments conducted, and detailed the operational procedures for creating our photochromic resin formulation and for controlling the 3D printer system. Lastly, we provided a brief discussion on the limitations of our current design and recommended topics for future work moving forward.

## 8. Bibliography

- [1] Yuhua Jin, Isabel Qamar, Michael Wessely, Aradhana Adhikari, Katarina Bulovic, Parinya Punpongsanon, and Stefanie Mueller. 2019. Photo-Chromeleon: Re-Programmable Multi-Color Textures Using Photochromic Dyes. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 701–712. DOI: <http://dx.doi.org/10.1145/3332165.3347905>
- [2] Formlabs. Formlabs Announces Color Kit: 3D Print in Color on the Form 2. <https://formlabs.com/company/press/formlabs-announces-color-kit-3d-print-color-form-2>
- [3] Henrique Teles Maia, Dingzeyu Li, Yuan Yang, and Changxi Zheng. 2019. LayerCode: Optical Barcodes for 3D Printed Shapes. *ACM Trans. Graph.* 38, 4, Article 112 (July 2019), 14 pages. <https://doi.org/10.1145/3306346.3322960>
- [4] Ryuji Hirayama, Atsushi Shiraki, Makoto Naruse, Shinichiro Nakamura, Hirotaka Nakayama, Takashi Kakue, Tomoyoshi Shimobaba, and Tomoyoshi Ito. 2016. Optical Addressing of Multi-Colour Photochromic Material Mixture for Volumetric Display. *Scientific Reports* 6, 1 (2016), 31543. <https://doi.org/10.1038/srep31543>
- [5] Daniel Saakes, Takahiro Tsujii, Kohei Nishimura, Tomoko Hashida, and Takeshi Naemura. 2013. Photochromic Carpet: Playful Floor Canvas with Color-Changing Footprints. In *Proceedings of the 10th International Conference on Advances in Computer Entertainment - Volume 8253* (Boekelo, The Netherlands) (*ACE 2013*). Springer-Verlag, Berlin, Heidelberg, 622–625.
- [6] Tomoko Hashida, Yasuaki Kakehi, and Takeshi Naemura. 2010. Photochromic Canvas Drawing with Patterned Light. In *ACM SIGGRAPH 2010 Posters* (Los Angeles, California) (*SIGGRAPH '10*). Association for Computing Machinery, New York, NY, USA, Article 26, 1 pages. <https://doi.org/10.1145/1836845.1836873>
- [7] Parinya Punpongsanon, Xin Wen, David S. Kim, and Stefanie Mueller. 2018. ColorMod: Recoloring 3D Printed Objects Using Photochromic Inks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) (*CHI '18*). Association for Computing Machinery, New York, NY, USA, Article 213, 12 pages. <https://doi.org/10.1145/3173574.3173787>
- [8] Junyi Zhu, Lotta-Gili Blumberg, Yunyi Zhu, Martin Nisser, Ethan Levi Carlson, Xin Wen, Kevin Shum, Jessica Ayeley Quaye, and Stefanie Mueller. 2020. CurveBoards: Integrating Breadboards into Physical Objects to Prototype Function in the Context of Form. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing*

*Systems* (Honolulu, HI, USA) (*CHI '20*). Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376617>

- [9] Yvonne Jansen, Pierre Dragicevic, Petra Isenberg, Jason Alexander, Abhijit Karnik, Johan Kildal, Sriram Subramanian, and Kasper Hornbæk. 2015. Opportunities and Challenges for Data Physicalization. *In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) (*CHI '15*). Association for Computing Machinery, New York, NY, USA, 3227–3236. <https://doi.org/10.1145/2702123.2702180>
- [10] Jianzhe Gu, David E. Breen, Jenny Hu, Lifeng Zhu, Ye Tao, Tyson Van de Zande, Guanyun Wang, Yongjie Jessica Zhang, and Lining Yao. 2019. Geodesy: Self-Rising 2.5D Tiles by Printing along 2D Geodesic Closed Path. *In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) (*CHI '19*). Association for Computing Machinery, New York, NY, USA, 1–10. <https://doi.org/10.1145/3290605.3300267>
- [11] Anycubic photon paper levelling method. (2018, December 22). [Video]. YouTube. <https://www.youtube.com/watch?v=gDARzoy>