Spatial Light Modulator Operation via Python:
Applications to Surface Structure Fabrication

Matthew Van Soelen
Faculty Mentor: Dr. David McGee
The College of New Jersey

ABSTRACT

Utilizing Python in conjunction with a Spatial Light Modulator allows for a wider variety of practical experiments in optical lithography. The goal of this research was to develop tools in python which could be used to further explore the capabilities of surface grating creation using a Spatial Light Modulator as a dynamic polarizer.

INTRODUCTION

- Exposing photosensitive azopolymer films to linearly polarized light induces a photomechanical response causing azopolymer molecules to become trapped perpendicular to the polarization. This drives a surface deformation of the film.
- Polarizers, varying intensity, and exposure time can be used to change the intensity and orientation of surface grating produced on the film
- A Spatial Light Modulator (SLM) contains an array of liquid crystals (LC) which change their orientation based on applied voltage.
- The SLM maps video input to voltages for each LC, changing their orientation. The SLM therefore can be used as a dynamic polarizer.
- Operating the SLM via Python allows for grating pattern generation and automated experiments.

Grating Generation

Mapping grayscale values to Polarization angle:

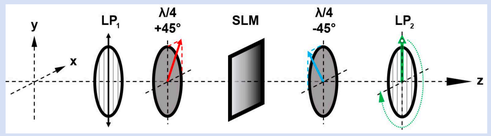

Fig 1. Simplified Experiment Setup

- The SLM has a display resolution of 1920x1152, and each pixel can have a unique gray scale value.
- Each grayscale value (0-255) corresponds to a polarization angle.
- As a laser beam passes through the SLM seen in Fig 1. the beams polarization is altered based on the pattern currently programmed into the SLM, acting as a dynamic polarizer.
- Periodic gray scale patterns such as a Saw Tooth (Fig 2.) or Sinusoidal functions produce analogous surface gratings.

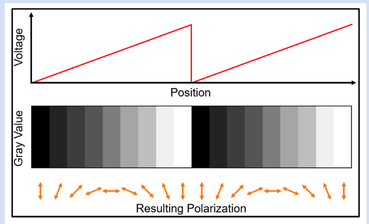

Fig 2. Gray Value to Polarization Angle Relation

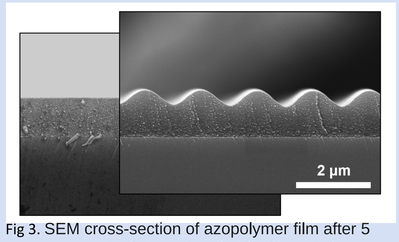

Fig 3. SEM cross-section of azopolymer film after 5 sec exposure to SLM pattern in Fig. 2. Note how polarization distribution generates sinusoidal surface relief grating

Programming

Python Tasks:

- Pattern generation – an 1920x1152 array of values mapping each column against a periodic sawtooth function based on user parameters. The array is then converted to PNG form using Python Image Processing Library.
- GUIs and displaying patterns to the SLM were accomplished using Python Tkinter Interface.
- Communication with SLM is treated as a second monitor connected via HDMI, extending the primary monitor.

RESULTS

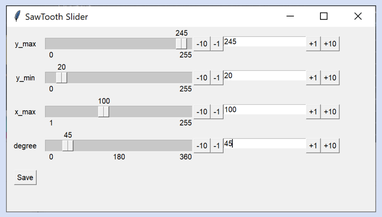

Fig. 4 SawTooth Slider Program

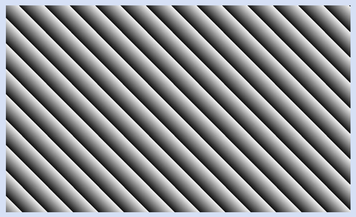

Fig. 5 Corresponding saw tooth pattern for Fig. 4

1. Using "SawTooth Slider" saw tooth patterns can be generated live, by changing parameters using sliders, buttons or text fields.
2. Each saw tooth is defined by a linear slope found using:
   - $y_{max} - y_{min} = m(x_{max} - x_0), where\ x_0 = 0$
   - $Column\ Gray\ Value = f(x) = m(x\ \%\ x_{max}) + y_{min}$
3. Arrays are then converted to PNG and rotated by the provided angle and cropped to the dimensions of the SLM (1920x1152).

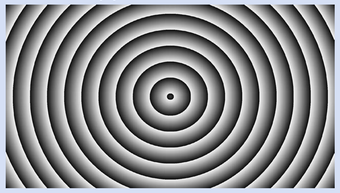

Fig. 6 Circular Saw Tooth Pattern

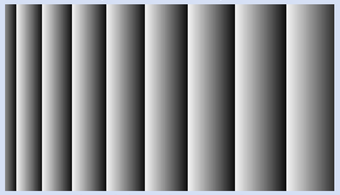

Fig. 7 Chirped Saw Tooth Pattern

Other Patterns:

- Circular saw tooth patterns(Fig. 6) can be generated in a similar manner, rather than columns of a gray value. The gray values are overlapping circles of decreasing radius.
- Patterns with continuously variable periods can also be produced, such as the Chirped saw tooth in Fig. 7

Fig. 8 Circular Saw Tooth grating on film corresponding to pattern in Fig. 6(left), AFM grating scan (right)

```
def create_grating():

    self.slope = (self.y_max.get()-self.y_min.get())/self.x_max.get()

    for i in range(array_width):
        color = self.slope * (i % self.x_max.get()) + self.y_min.get()
        self.data[:, i] = color

self.img = Image.fromarray(self.data)
self.img = self.img.convert('L')
self.img = self.img.rotate(self.degree.get())
self.img = center_crop(self.img, array_width, array_height ,width, height)
self.photo = ImageTk.PhotoImage(self.img)
```

Fig. 9 Pattern Generation Function

Automating SLM Surface Grating Production:

Extended and adapted Luke Kurlanski's Hologram Creation Program to incorporate the SLM, merging multiple images and grating patterns.

1. Added the ability to create gratings using the previous methods or as a PNG. These gratings can be linked to a specific image or gray value.
2. Developed algorithm to merge four images (Fig. 11), resulting in a hologram which displays each image at a given viewing angle.
3. Implemented synchronously displaying grating patterns for corresponding images or gray value, while experiment is running.
4. Ensured experiments could be saved and loaded properly.

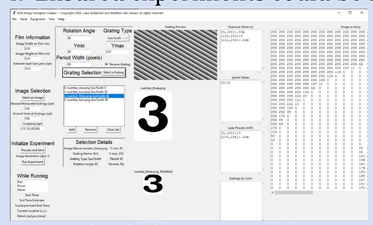

Fig. 10 Adapted Hologram Creation Program

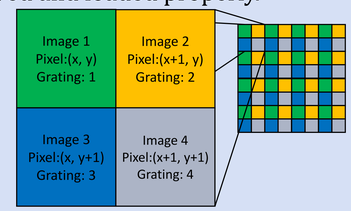

Fig. 11 Super Pixel (left), Resulting Hologram(right)

Merging Images is accomplished by following the pattern shown in Fig. 11. Each hologram pixel (hixel) will be made of one of the four original images and a corresponding grating.

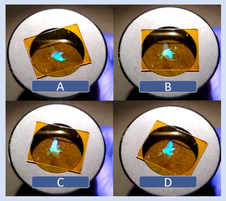

Fig. 12 Hologram constructed by merging 4 images of a crane taking flight. Each image becomes more present as the single film is rotated to their respective viewing angle, A at 15°, B at 0°, C at 15°, and D at 30°. Viewing angles are specified by the grating associated with each image.

FUTURE WORK

- Develop program to incorporate multiple images of a moving object to simulate motion (multiplexing)
- Develop algorithm to produce not just period but arbitrary surface structure.
- Encoding hidden images within a surface grating.

REFERENCES

Tkinter Documentation:
https://docs.python.org/3/library/tk.html

Python Image Processing:
https://pillow.readthedocs.io/en/stable/

Acknowledgments

This project was Supported by the National Science Foundation, and the TCNJ MUSE Program.
Special thanks to Dr. David McGee, Jonas Strobelt, and Luke Kurlandski for their guidance and feedback.

Presented at MUSE, Summer 2020