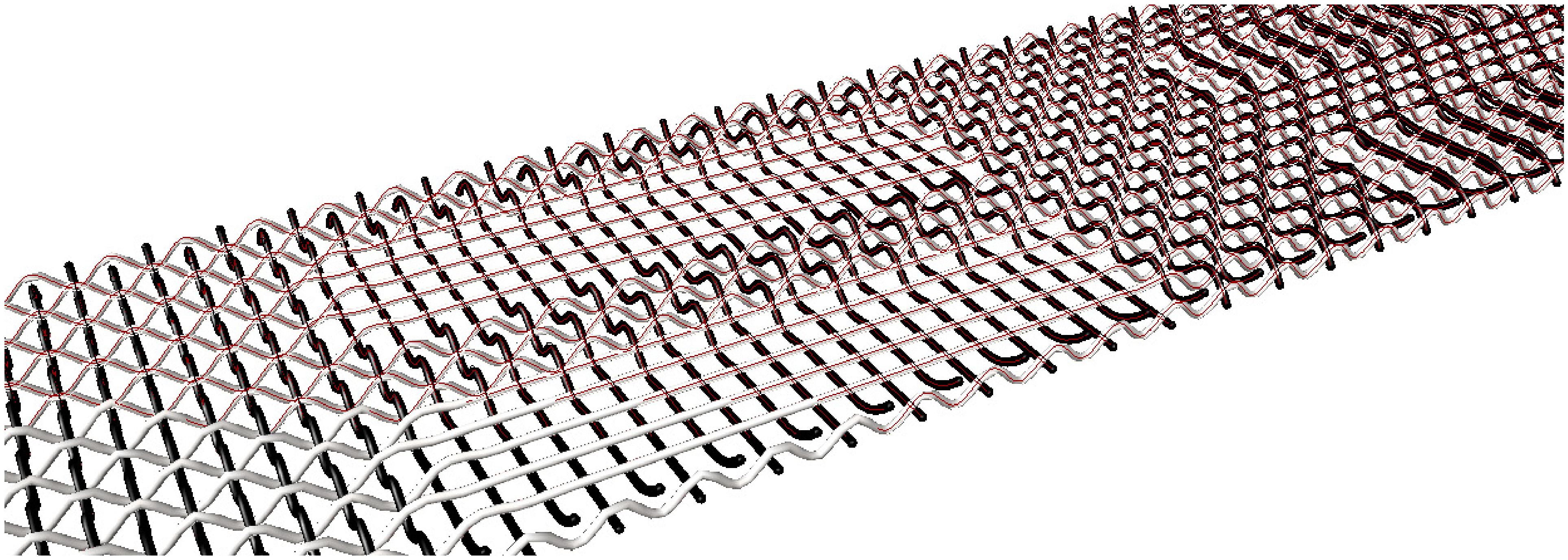
Crafting Material Interfaces

Final project ~ Fall 2011 ~ Dena Molnar

Woven Sensors

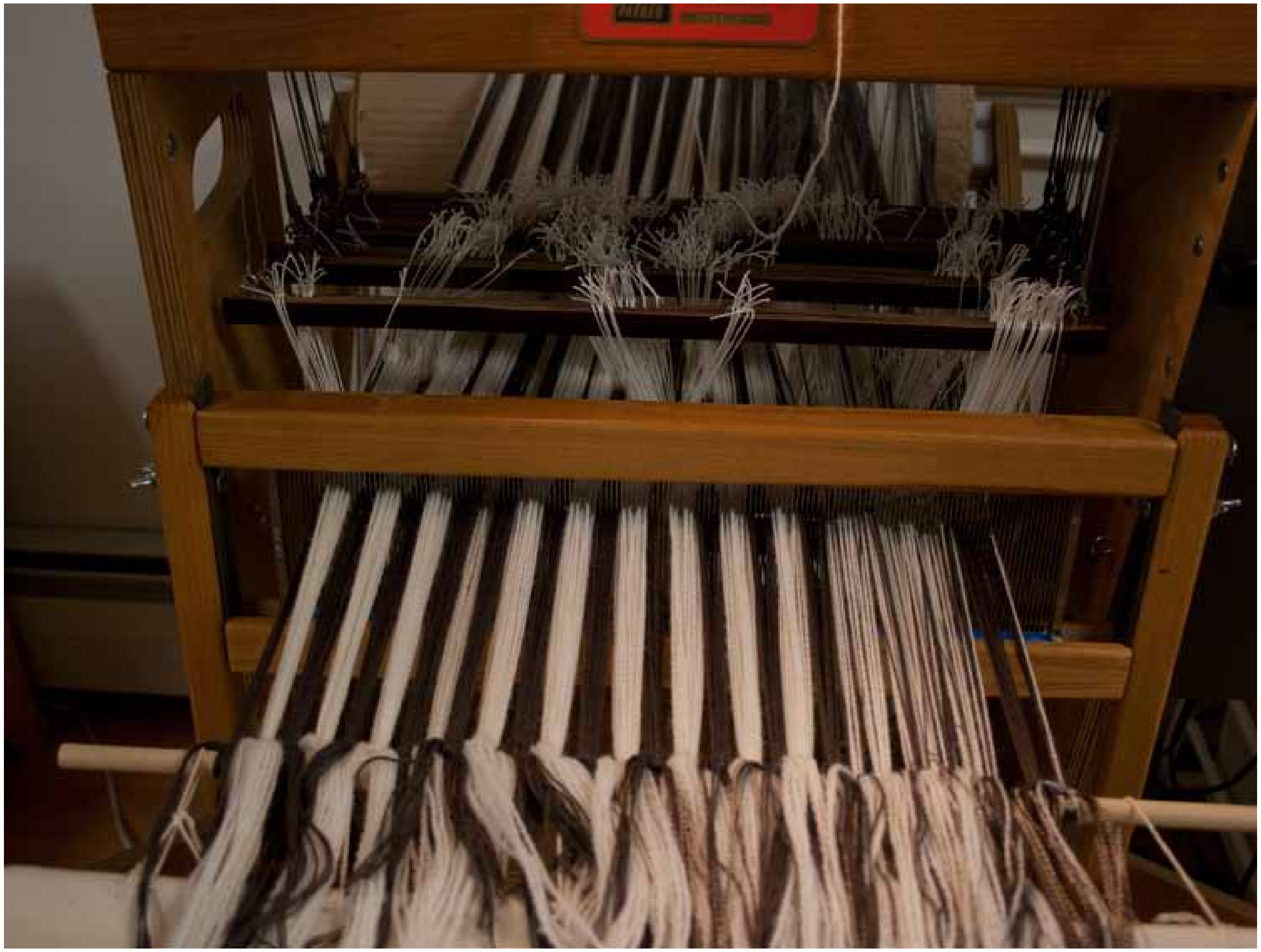


Concept

As part of my final project I was interested in developing and simulating a series of textiles whose sensing capabilities are inherent in the construction of the material. In the woven prototype, I used a construction called deflected double weave, which allows for areas of conductive warp and weft to 'float' over each other instead of weave together. By inserting Velostat between these layers, a series of pressure sensors was created. For the descriptions in Grasshopper, I used Firefly to simulate when pressure is applied to the material. Firefly is a Grasshopper plug in which links modeling and Arduino, allowing for interactive prototyping.

Application and Future Development

I invision that these woven pressure sensors could be constructed on a very small scale, allowing for a material with dense sensing points. Modeling this serves as a way to visually test the functionality of the sensors, while providing an acurate description of a material's construction/surface topology. From this series of explorations, I am interested in further developing descriptions that simulate actuated materials.

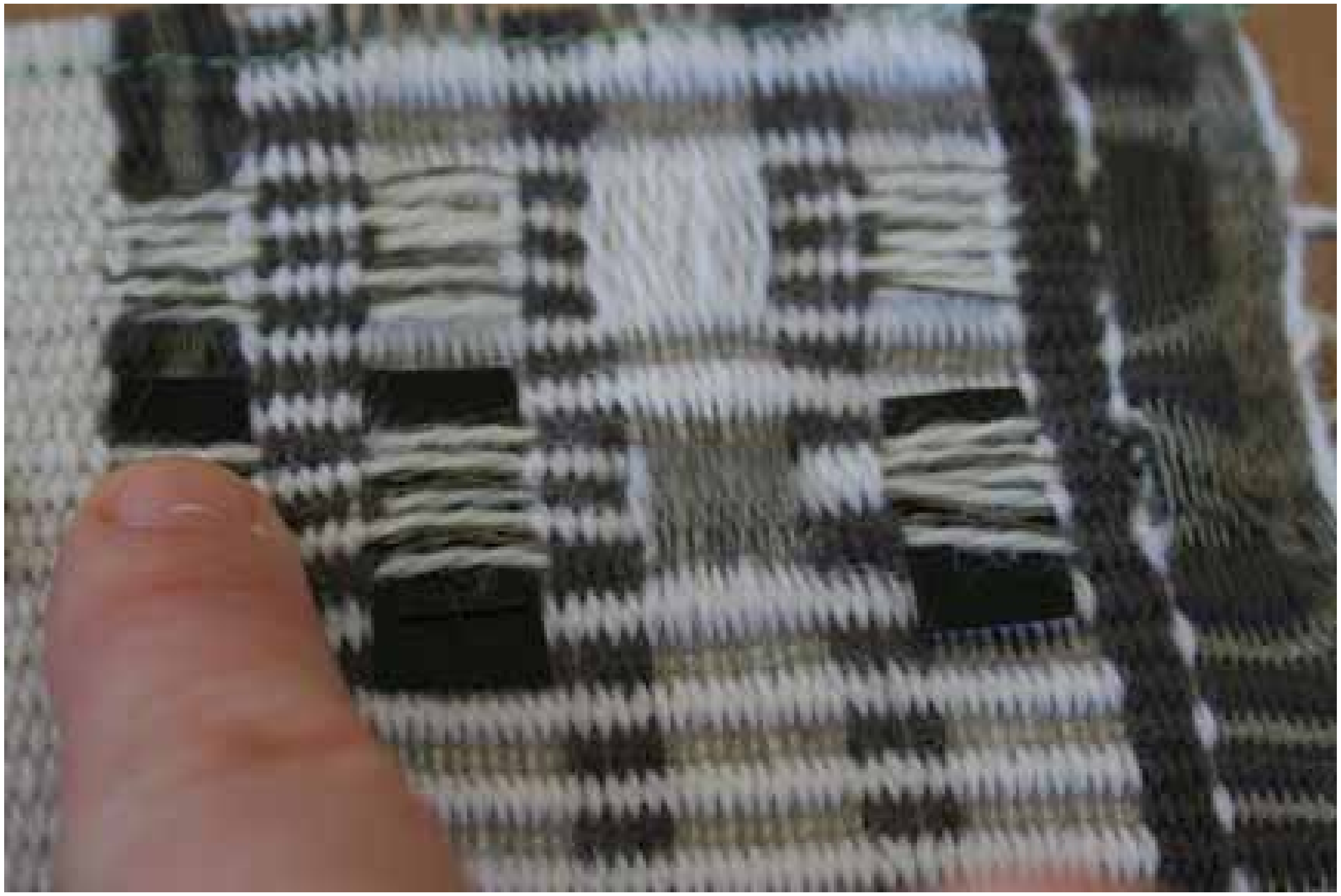


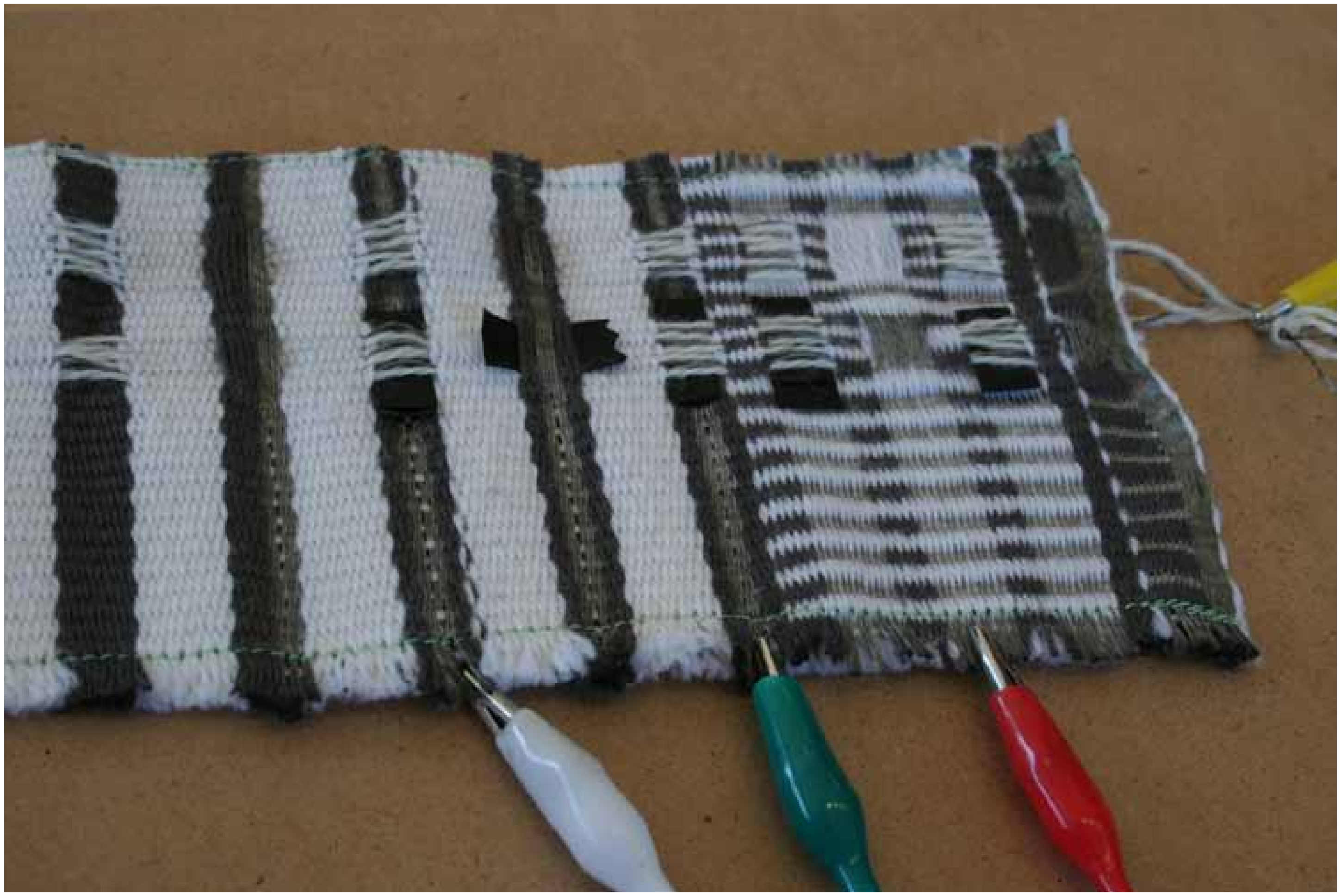
Prototyping Background

To begin this series of Investigations, I have been working in a weaving studio to develop prototypes. I wound a warp of wool, bamboo and steel threads . The steel ends in the warp were used as inputs in the pressure sensor circuit design. This image shows the warp wound onto a beam at the back of the loom, threaded through a sequence of eylets called heddles and through a comb like frame called a reed. The threading maintains the order that the warp ends will weave.

Woven Sensor

This image displays the finished woven protoype. As mentioned earlier, I utilized a construction called deflected double weave, which allows for areas of conductive weft (the horizontal threads) to “float” over areas of conductive warp (vertical threads). For the first set of visualizations, the conductive warp channels were used as inputs and the wefts were connected to ground by a steel warp stripe at far right. I inserted Velostat between the 2 conductive layers to create resistance. When pressed, Velostat decreases in resistance, creating a lower value reading.





Sensor Circuit Connections

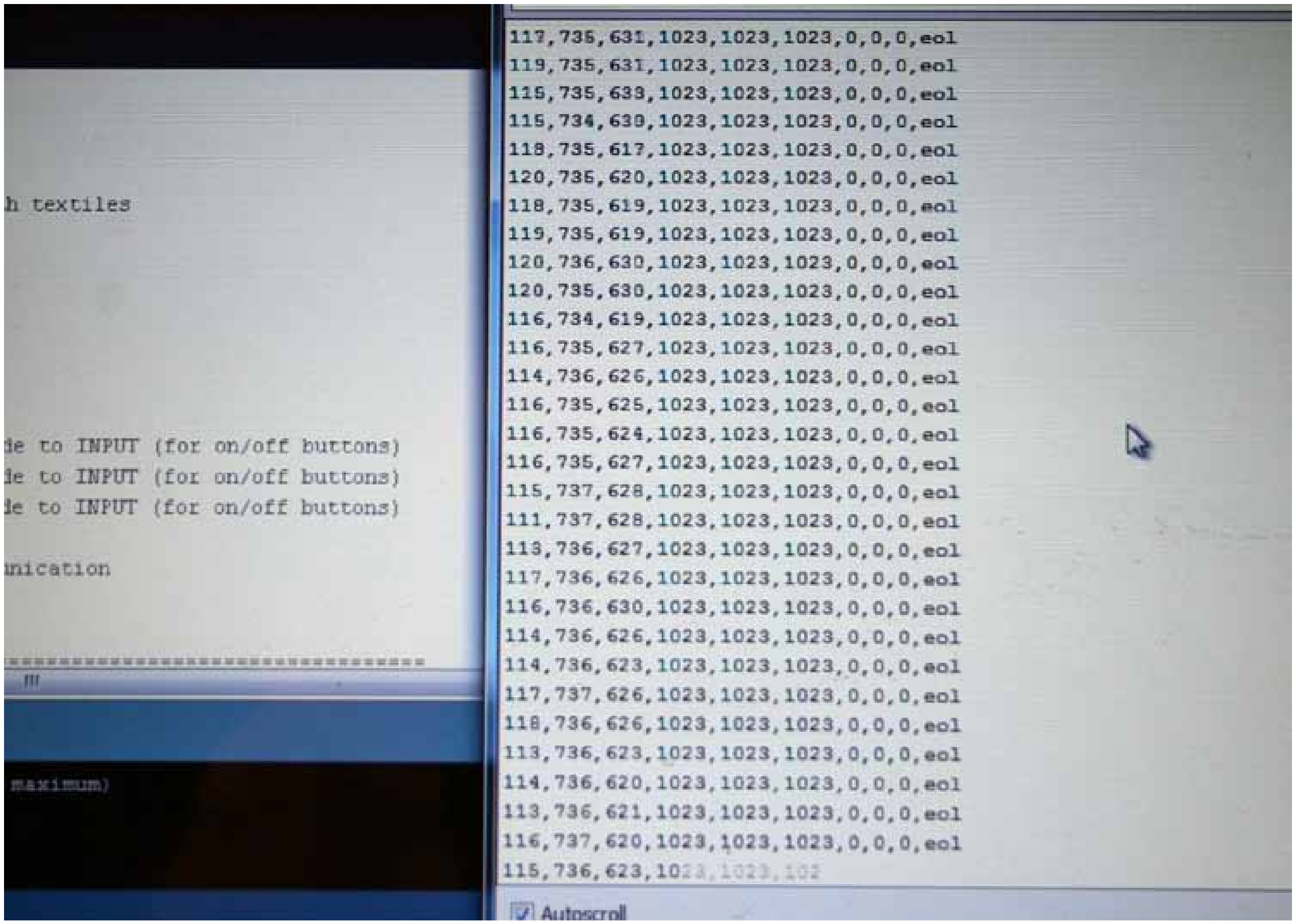
Here you can see that conductive warp channels are connected to 3 alligator clips (White, Green and Red) that are in turn used as Analog input pins on an Arduino. The conductive wefts are connected to ground via Yellow clip.

Arduino and Processing

To test the pressure sensors, I used a simple Arduino code to read the incoming data from the serial port. I then used a Processing sketch to visualize when one of the sensors was being pressed.

Arduino Code

```
void setup()
{
  Serial.begin(9600);
  digitalWrite(A0, HIGH);
  digitalWrite(A1, HIGH);
  digitalWrite(A2, HIGH);
//other pins not used
}

void loop() {
  Serial.print(analogRead(A0));
  Serial.print("\t");
  Serial.print(analogRead(A1));
  Serial.print("\t");
  Serial.print(analogRead(A2));
  Serial.println();
  delay(100);
}
```

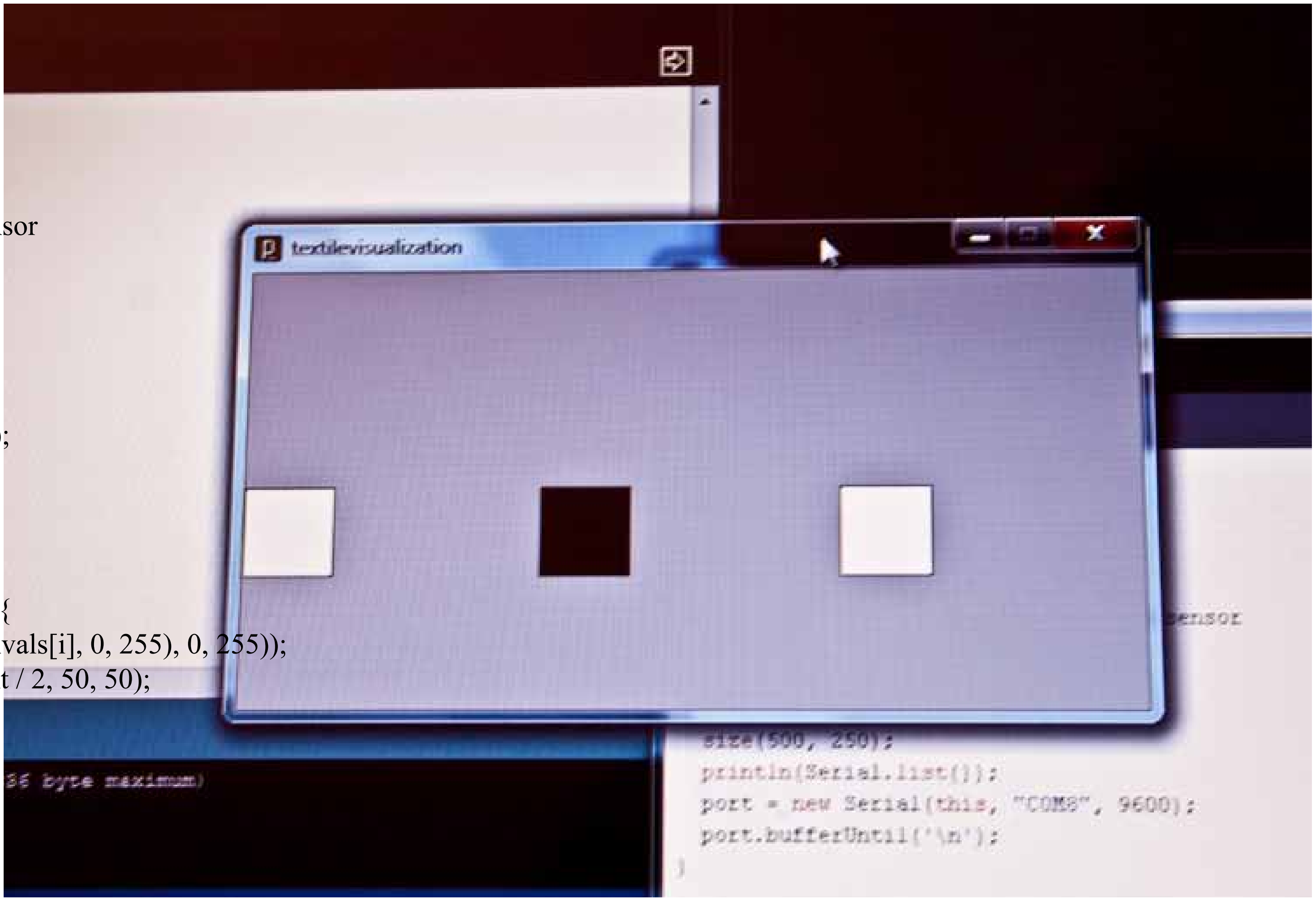


```
h textiles

de to INPUT (for on/off buttons)
de to INPUT (for on/off buttons)
de to INPUT (for on/off buttons)

nication

maximum)
```

```
117,736,631,1023,1023,1023,0,0,0,eol
119,735,631,1023,1023,1023,0,0,0,eol
115,735,633,1023,1023,1023,0,0,0,eol
115,734,630,1023,1023,1023,0,0,0,eol
118,735,617,1023,1023,1023,0,0,0,eol
120,736,620,1023,1023,1023,0,0,0,eol
118,735,619,1023,1023,1023,0,0,0,eol
119,735,619,1023,1023,1023,0,0,0,eol
120,736,630,1023,1023,1023,0,0,0,eol
120,735,630,1023,1023,1023,0,0,0,eol
116,734,619,1023,1023,1023,0,0,0,eol
116,735,627,1023,1023,1023,0,0,0,eol
114,736,626,1023,1023,1023,0,0,0,eol
116,735,625,1023,1023,1023,0,0,0,eol
116,735,624,1023,1023,1023,0,0,0,eol
116,735,627,1023,1023,1023,0,0,0,eol
115,737,628,1023,1023,1023,0,0,0,eol
111,737,628,1023,1023,1023,0,0,0,eol
113,736,627,1023,1023,1023,0,0,0,eol
117,736,626,1023,1023,1023,0,0,0,eol
116,736,630,1023,1023,1023,0,0,0,eol
114,736,626,1023,1023,1023,0,0,0,eol
114,736,623,1023,1023,1023,0,0,0,eol
117,737,626,1023,1023,1023,0,0,0,eol
118,736,626,1023,1023,1023,0,0,0,eol
113,736,623,1023,1023,1023,0,0,0,eol
114,736,620,1023,1023,1023,0,0,0,eol
113,736,621,1023,1023,1023,0,0,0,eol
116,737,620,1023,1023,1023,0,0,0,eol
115,736,623,1023,1023,102
```

Processing Code

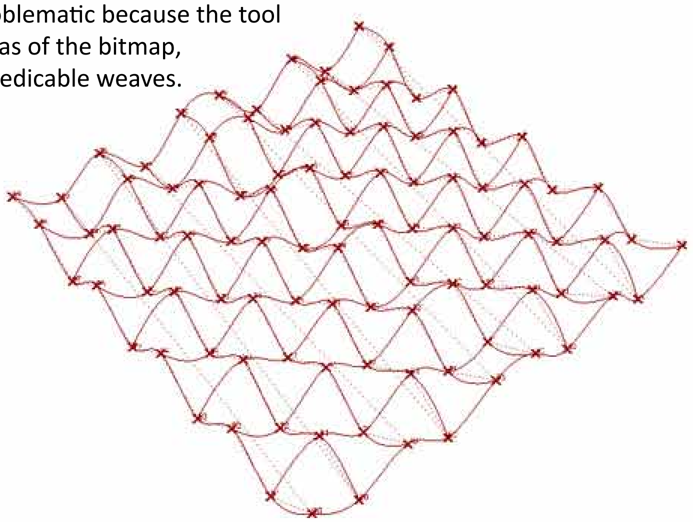
```
import processing.serial.*;
Serial port;
int[] numbers = { 0, 0, 0 };
int[] maxvals = { 255, 255, 255 };
//write coordinates for each pressure sensor

void setup()
{
  size(500, 250);
  println(Serial.list());
  port = new Serial(this, "COM8", 9600);
  port.bufferUntil('\n');
}

void draw() {
  for (int i = 0; i < numbers.length; i++) {
    fill(constrain(map(numbers[i], 0, maxvals[i], 0, 255), 0, 255));
    rect(i * width / numbers.length, height / 2, 50, 50);
  }
}

void serialEvent(Serial p)
{
  String s = p.readStringUntil('\n');
  numbers = int(s.trim().split("\t"));
}
```

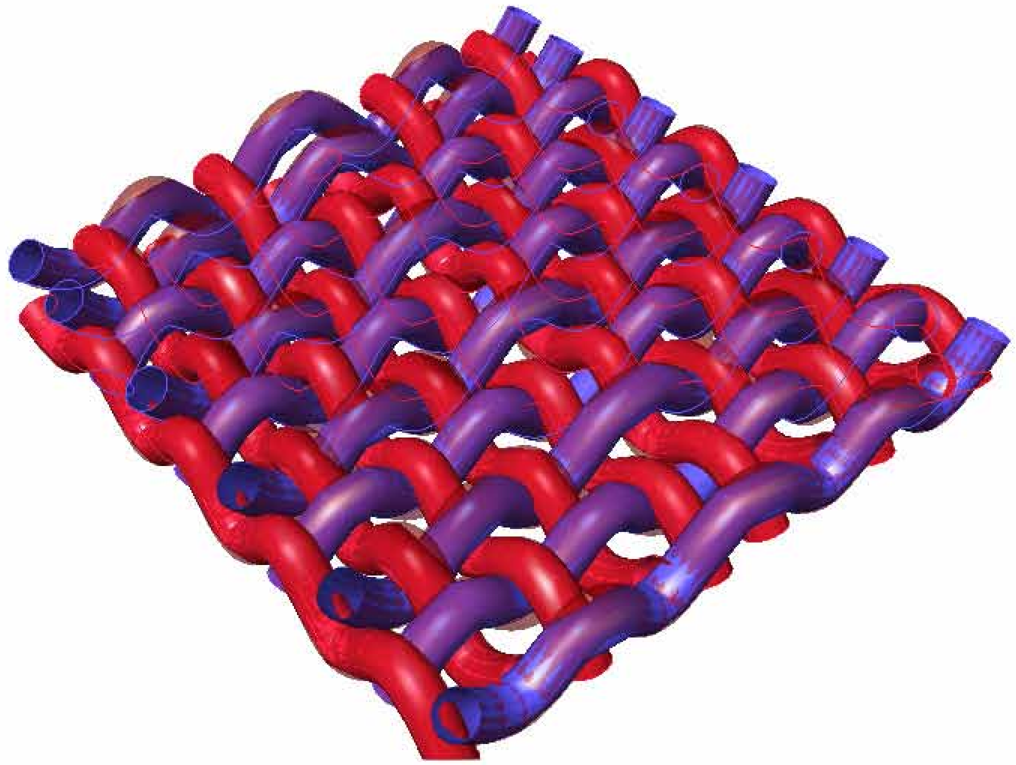


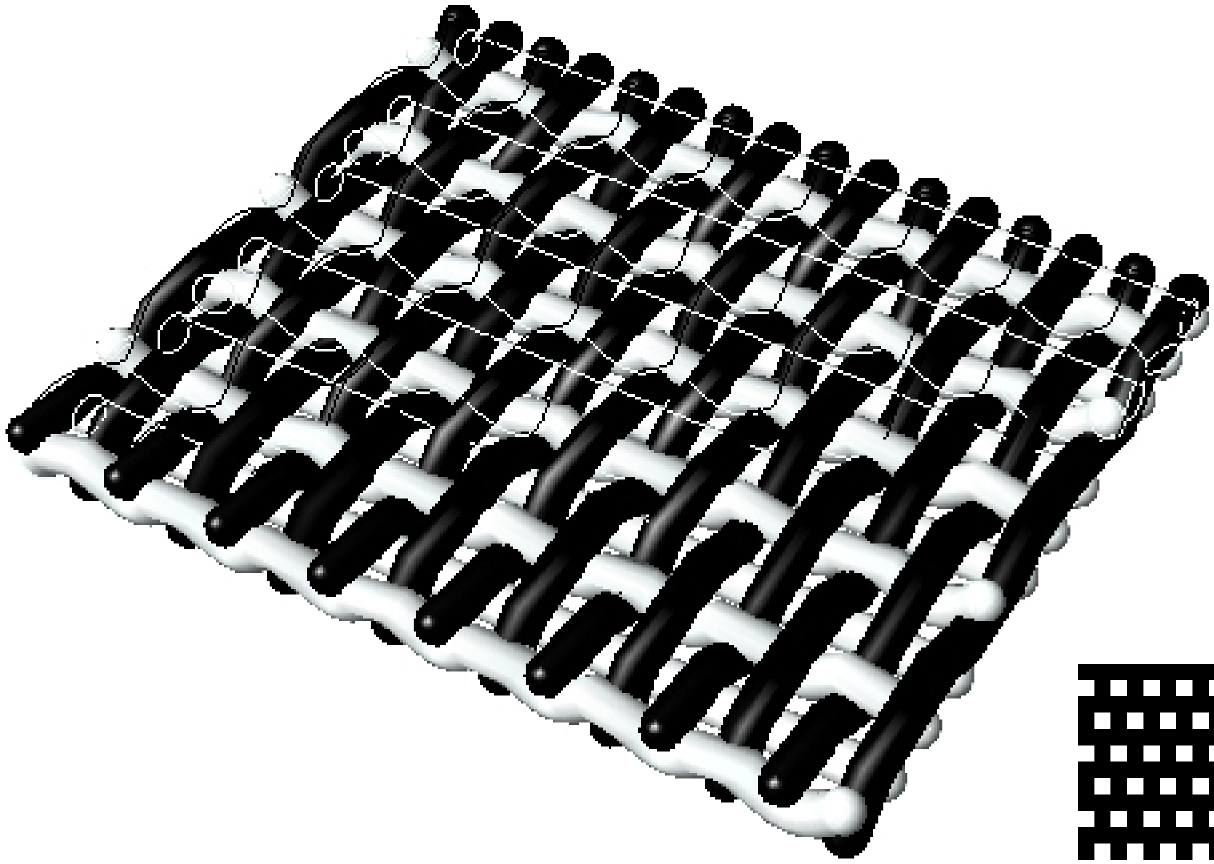
```
36 byte maximum)
ensor
  size(500, 250);
  println(Serial.list());
  port = new Serial(this, "COM8", 9600);
  port.bufferUntil('\n');
}
```

Modeling Background

As part of my investigations this semester, I developed a series of descriptions that sought to render textiles. I had come across some documentation online, but was searching for a simple/ easy to understand way of doing this. Initial attempts focused on constructing simple weaves. I also tested using the Image Sampler in Grasshopper to import bitmaps developed in a textile program. This proved problematic because the tool read “grey” areas of the bitmap, producing unpredicable weaves.

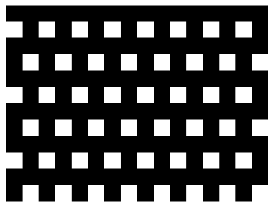


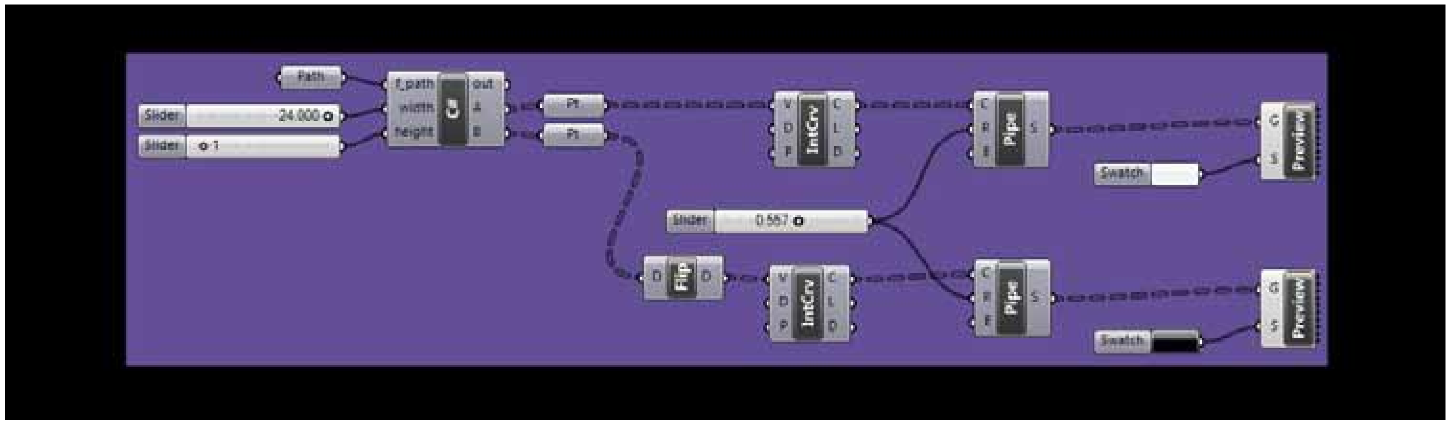




Final Solutions

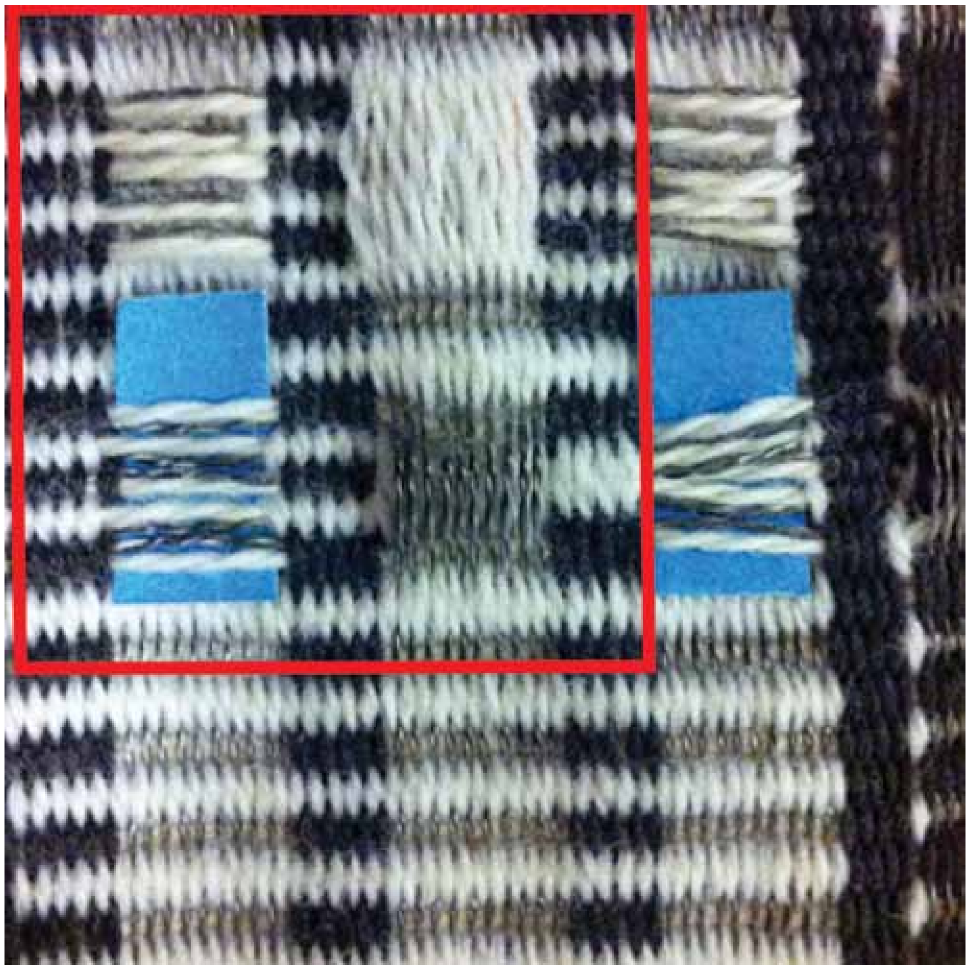
The most relaible solution involved importing Bitmaps through a variable path in the C# component of Grasshopper. In general, the Grasshopper description became much simpler, once I started using this . To do this, I added the Variable Path to the list of inputs. I then set the Path to the desired Bitmap.

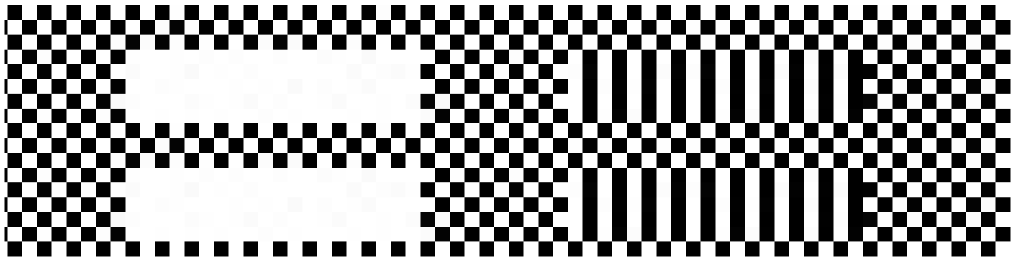




C # Code

```
private void RunScript(string f_path, double width, double height, ref object A, ref object B)
 {
  //......................load bitmap and get resolution along x and y
  Bitmap my_image = new Bitmap(f_path);

  int rx = my_image.Width;
  int ry = my_image.Height;

  //.................................calculate pixel width
  double PixelWidth = width / (double) (rx - 1.0);

  //...................................Create the array of elevation values
  double [,] weftht = new double[rx, ry]; //height values
  double [,] warpht = new double[rx, ry]; //height values
  DataTree<Point3d> warp = new DataTree<Point3d>();
  DataTree<Point3d> weft = new DataTree<Point3d>();

  int i;
  int j;
  //....................................Compute the Wefts
  //....................................extract elevation values from brightness component of bitmap
  for(j = 0; j < ry;++j) {
   for(i = 0; i < rx; ++i) {
     Color c = my_image.GetPixel(i, j);
     weftht[i, j] = (c.GetBrightness()) * height;
     weft.Add(new Point3d(i * PixelWidth, ry - (j * PixelWidth), weftht[i, j]), new GH_Path(j));
   }
  }
  //....................................Compute the Warps
  //....................................extract elevation values from brightness component of bitmap
  for(j = 0; j < ry;++j) {
   for(i = 0; i < rx; ++i) {
     Color c = my_image.GetPixel(i, j);
     warpht[i, j] = height - (c.GetBrightness()) * height;
     warp.Add(new Point3d(i * PixelWidth, ry - (j * PixelWidth), warpht[i, j]), new GH_Path(j));
   }
  }

  A = weft;
  B = warp;
```



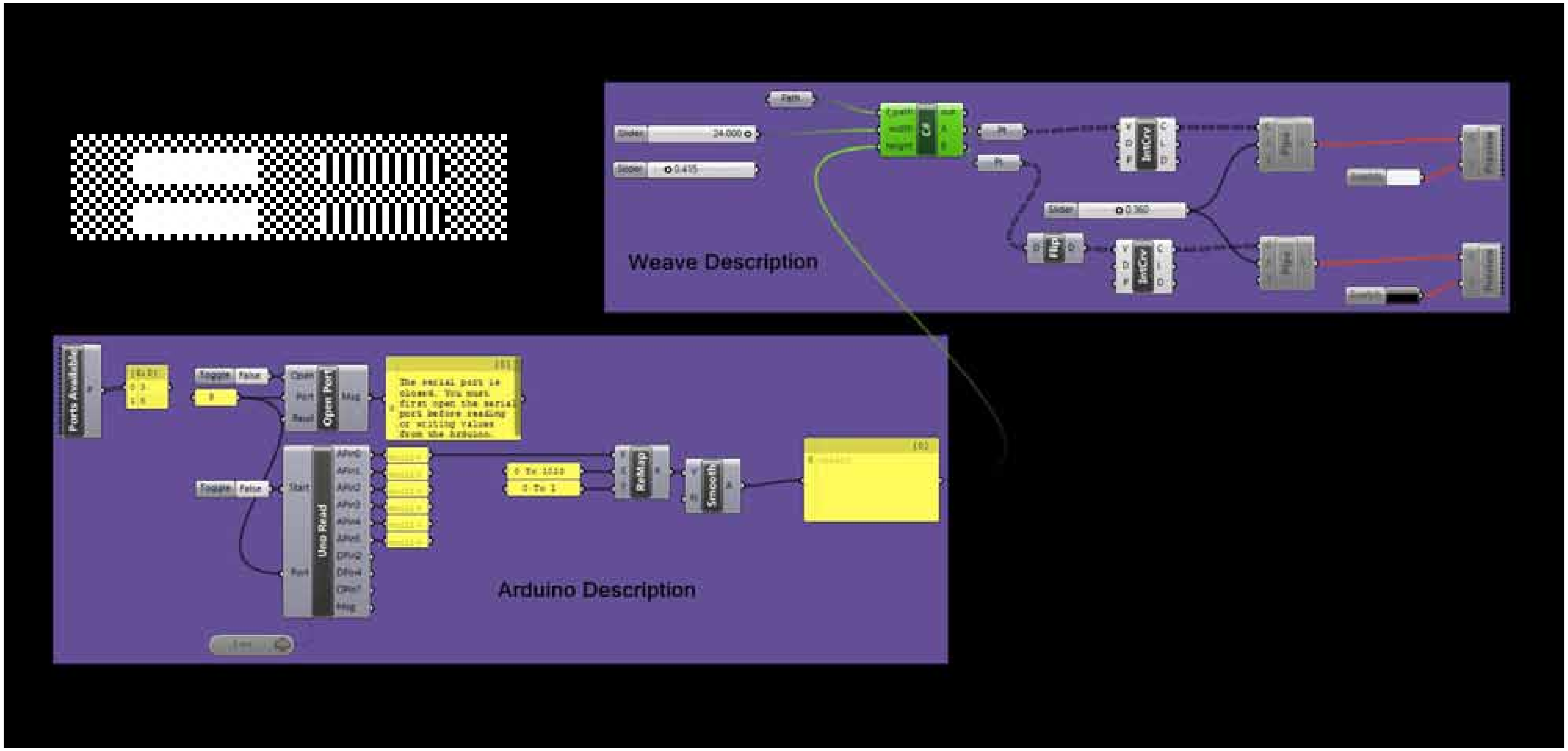


Weave Draft

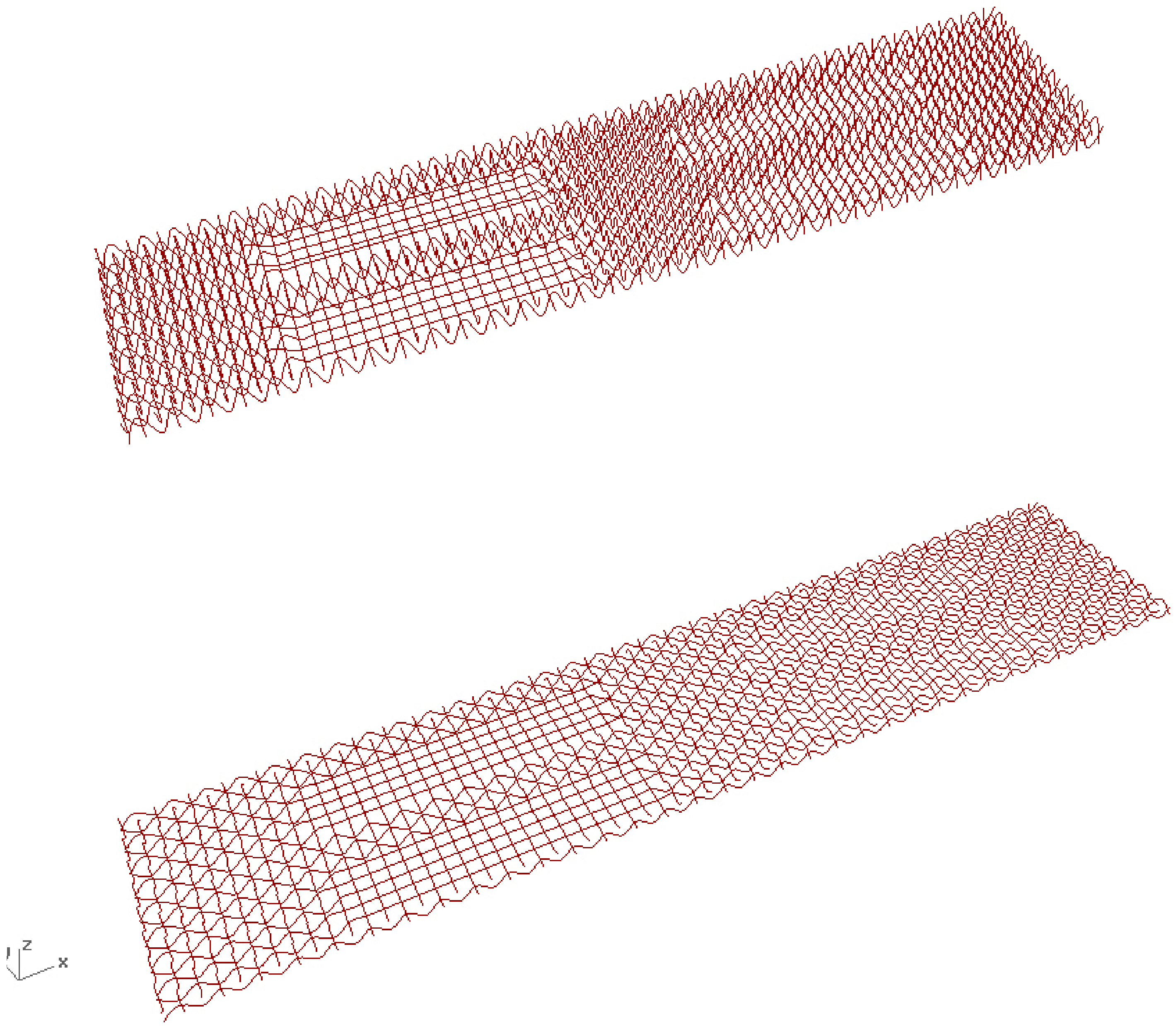
I selected an area of my weave draft with pocket sensors to render, as using the entire file tended to crash Grasshopper. I explored inputting the weave data into Excel and importing this information in place of Bitmaps, but Bitmaps are still the most visually useful.

Arduino ,Grasshopper, and Firefly

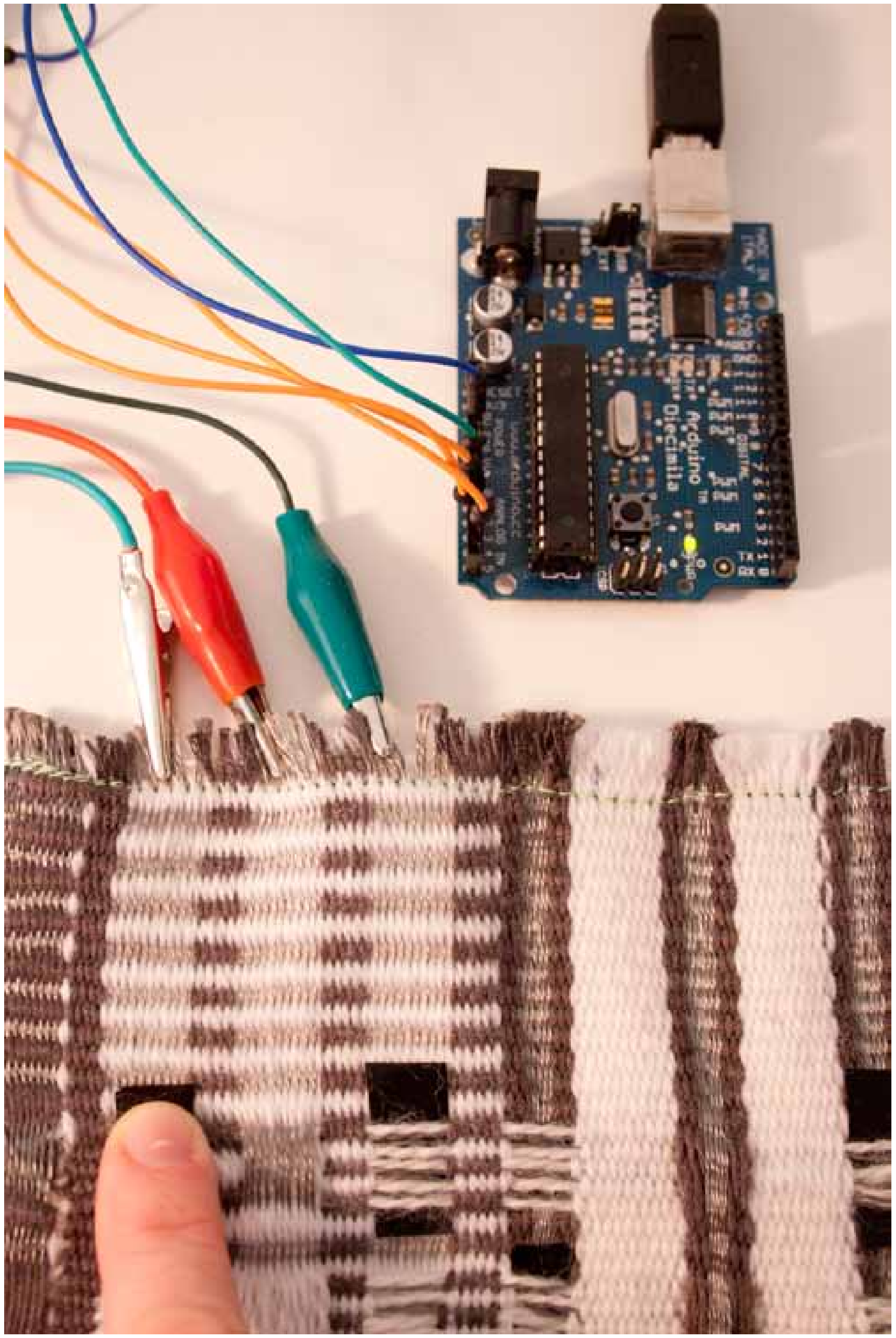
Firefly is a plug in for Grasshopper, which allows you simulate data read from an Arduino. To use these tools in tandem, I uploaded Firefly code to the Arduino that allows communication between Arduino and Grasshopper/Firefly. In the GH description, I used Arduino/Firefly components that matched my physical Arduino and port. The Grasshopper simulation is not un like a Processing sketch in that it visually represents the data otherwise seen in Arduino's serial monitor.

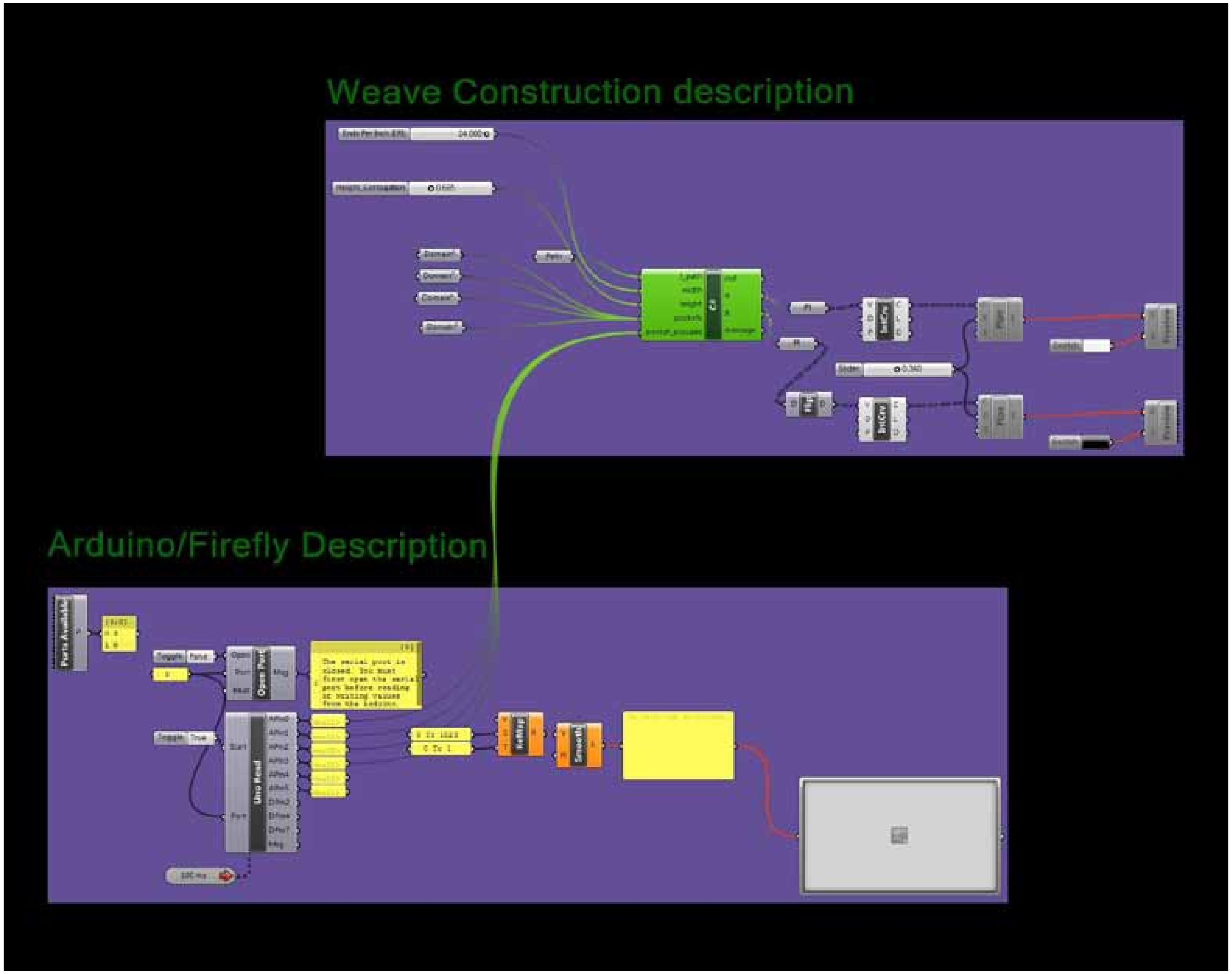


The serial port is closed. You must first open the serial port before reading or writing values from the Arduino.



By replacing the Corrogation_Height slider with an input pin (A0) from the Arduino Description, I was able to control the height by pressing down on the woven pressure sensor connected to this same pin on the physcial Arduino. At this point, all 3 input pins connected to the textile and Arduino could be used to control height.

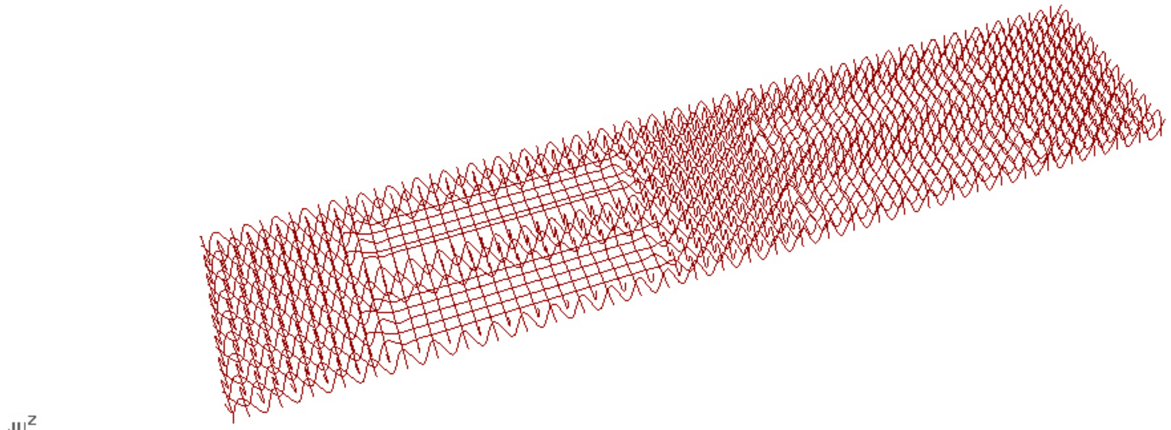

Arduino
Diecimila

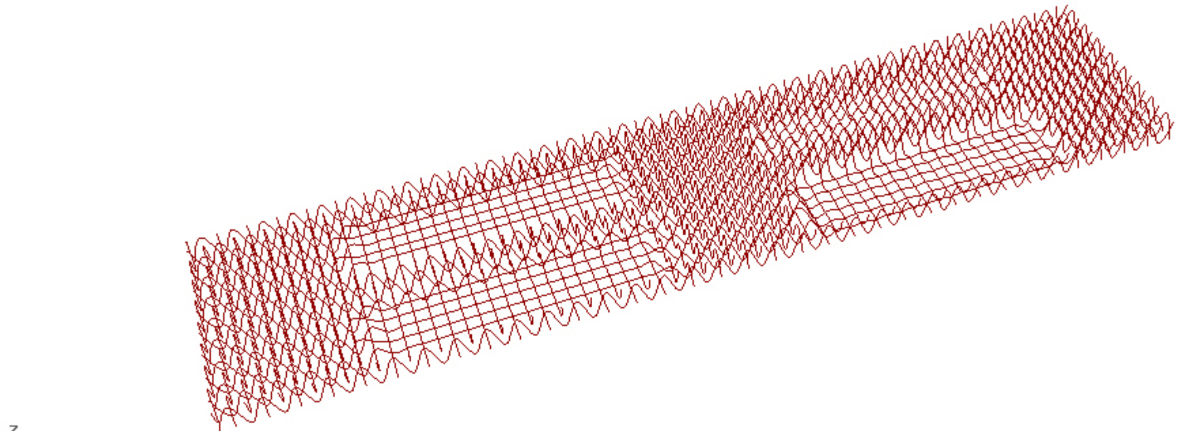

Weave Construction description
Arduino/Firefly Description
Ports Available
Open Port
Uno Read
ReMap
Smooth
The serial port is closed. You must first open the serial port before reading or writing values from the Arduino.
0 To 1023
0 To 1

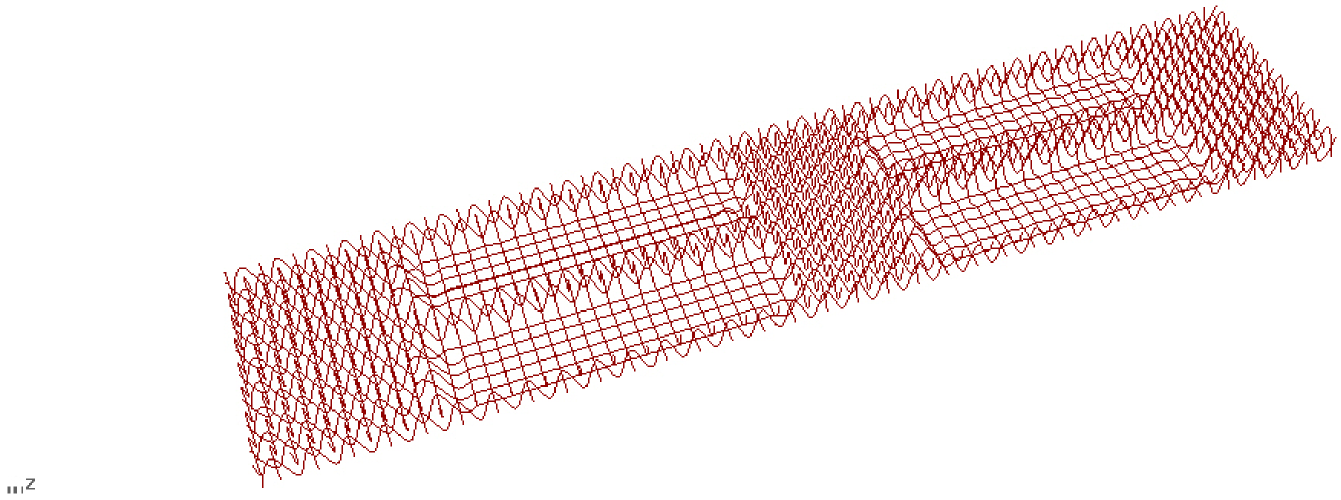
For the Final solution, I wanted to control the areas of deformation indepedently with respect to the actual location of the pressure sensors on the textile. For the circuitry, This involved disconnecting the conductive wefts connected to a common ground and having them reconnected independently as Digital Outputs on the Arduino. For the Grasshopper Description, this involved minimal changes in the available Firefly code, as well as defining individual domains (UV) for each “pocket” pressure sensor and using these domains as variables in the C# component. As a final step, I bypassed remapping the values of the sensor readings (with the remap component) by adding a Variable “Pocket Pressure” and dividing the incoming values by 1023.

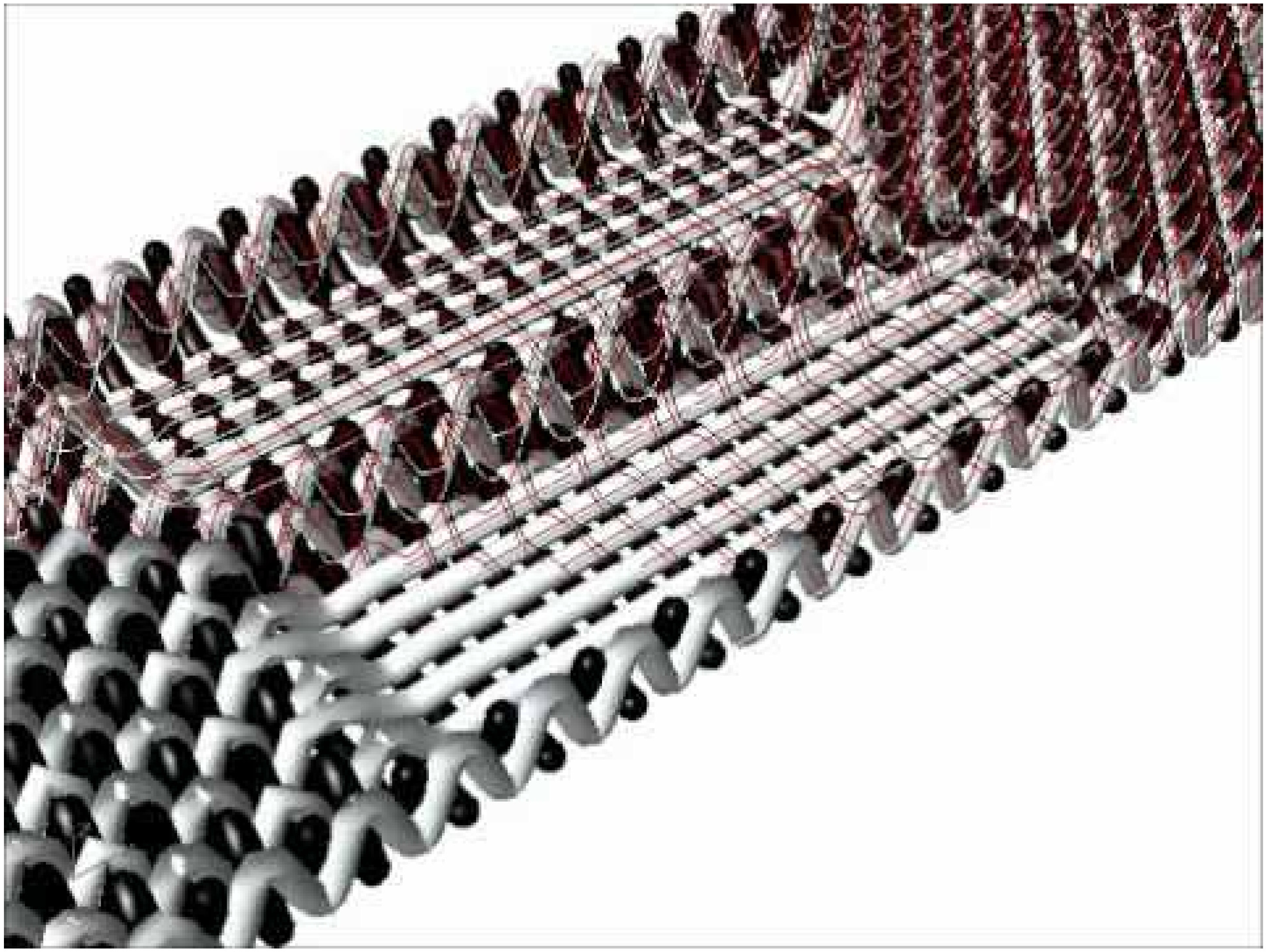
Independent Sensor Simulations

From Top:
Not pressed;
Bottom Right pressed
All Pressed









Thank you

Video of Final Simulation:

http://www.youtube.com/watch?v=2zG2UjtKTqw&feature=youtu.be