SOEN487 PM2.3 - Team 10 and 11

Initial ideas for OpenISS integration
Magenta, p5.js and OpenNI2

Magenta:

[Magenta is a research project developed by Google Brain team that explores the role of machine learning in the creation of art and music. Magenta uses models that are trainable using user data such as music, audio and images. Once trained, the Magenta model can produce images or audio clips that are similar to the one entered by 'moving between' the training data and using deep learning patterns. Magenta is built on top of tensorflow which is an open source python math library and machine learning framework.

Some interesting research being done within Magenta are:

- MusicVAE (Variational AutoEncoder), which creates a palette for music scores using machine learning.

- NSynth (Neural Synthesizer), a new approach to audio synthesis using neural networks. In traditional synthesizer we used hand-designed components such as oscillators and wavetables to generate an audio or soundtrack, however, NSynth creates sounds by using deep neural networks. NSynth helps the musician to create new sounds and audio that were difficult or impossible with a hand-tuned synthesizer.

- Onset and Frames, for piano transcription. This feature transcribes piano solos into midi using machine learning techniques.

For use with OpenISS, Magenta's ability to alter images into certain styles and geometrical forms using given training data will be useful and could be a final step in the service pipeline.] [1]

[1] Reference: https://magenta.tensorflow.org/

[2] Reference: http://thepsychreport.com/wp-content/uploads/2015/06/forest.jpg

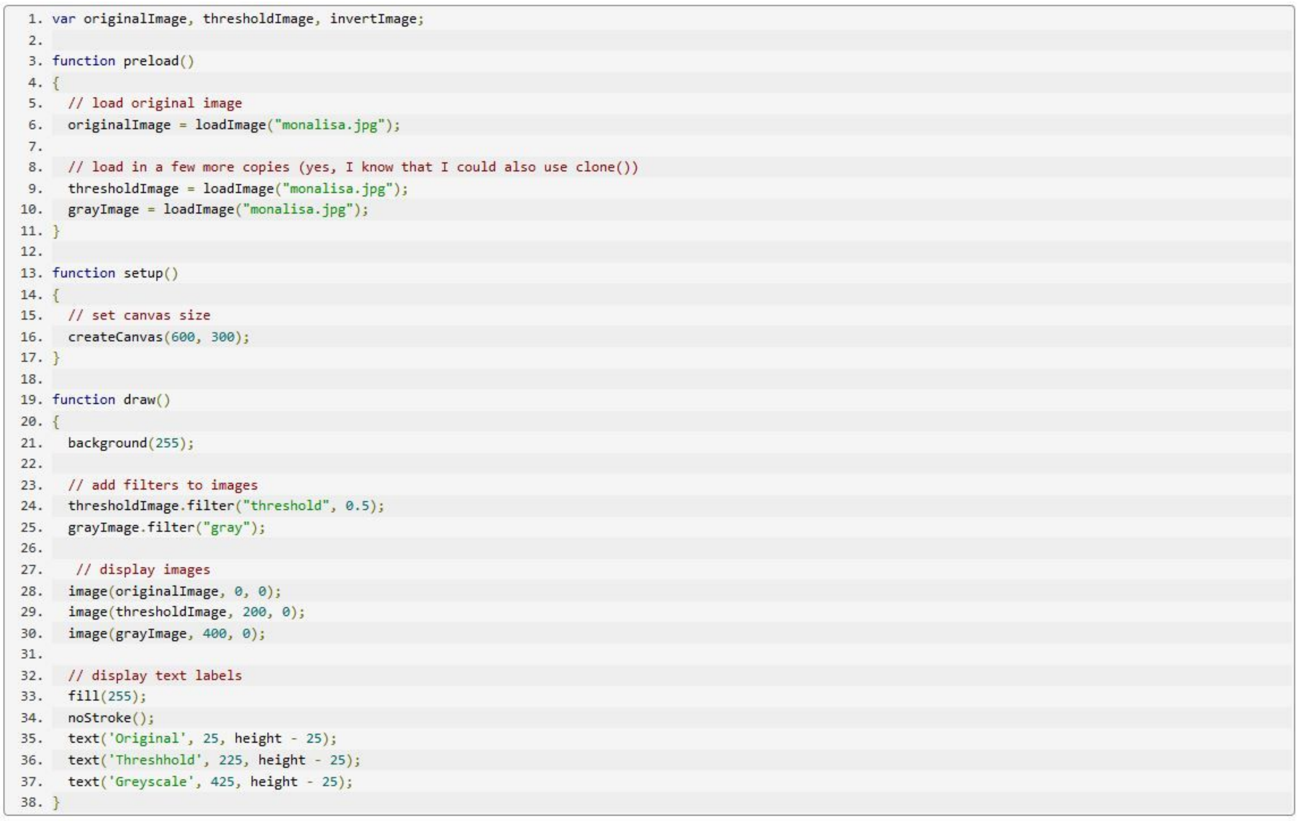
p5.js

[**p5.js** is a JavaScript library with a full set of drawing functionality that makes coding accessible for artists and designers. p5.js can interact with HTML5 objects, such as text, input, video, webcam, and sound.
One benefit of p5.js is that it can be extended with addon libraries allowing a developer to manipulate HTML objects with p5 and adding sliders, buttons, form elements, etc.

All of the p5.js code is written inside two function declarations called **setup** and **draw**. The setup function contains the code related to initialization of the program.
The code inside the setup function is executed once at the beginning of the program.
The draw function contains the drawing related functionality. It gets executed after the setup function and it gets executed continuously (around 60 times/frames, per second by default) which allows the creation of all kinds of interactive and animated visuals.] [3]
[3] Reference: https://css-tricks.com/creating-book-cover-using-javascript-p5-js/

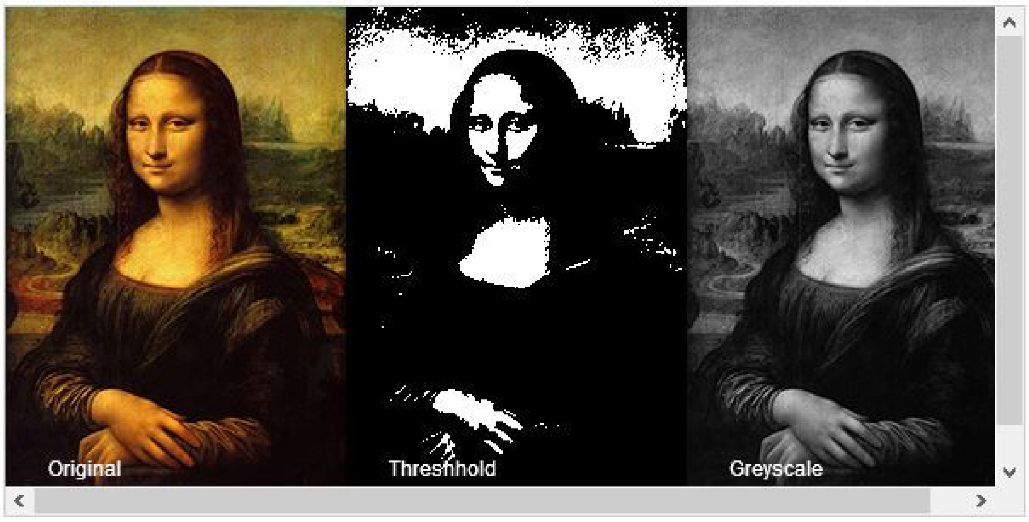
[**p5.dom library** for interaction with HTML5 objects-text, hyperlink, image, input, video, audio, and webcam.
p5.sound library provides an interface to HTML5 web audio API for loading, playing, and synthesizing sounds.] [4]
[4] Reference: https://github.com/processing/p5.js/wiki/p5.js-overview

[**Instructions on how to use the p5.js libraries** and how to create your own libraries can be found here: https://p5js.org/libraries/
P5.js can manipulate live video, access information about each pixels, apply filters, etc.] [5]



```
var originalImage, thresholdImage, invertImage;

function preload()
{
  // load original image
  originalImage = loadImage("monalisa.jpg");

  // load in a few more copies (yes, I know that I could also use clone())
  thresholdImage = loadImage("monalisa.jpg");
  grayImage = loadImage("monalisa.jpg");
}

function setup()
{
  // set canvas size
  createCanvas(600, 300);
}

function draw()
{
  background(255);

  // add filters to images
  thresholdImage.filter("threshold", 0.5);
  grayImage.filter("gray");

   // display images
  image(originalImage, 0, 0);
  image(thresholdImage, 200, 0);
  image(grayImage, 400, 0);

  // display text labels
  fill(255);
  noStroke();
  text('Original', 25, height - 25);
  text('Threshhold', 225, height - 25);
  text('Greyscale', 425, height - 25);
}
```

[5] Reference: http://coursescript.com/notes/interactivecomputing/images/



[5] Reference: http://coursescript.com/notes/interactivecomputing/images/

[**APPLYING FILTERS:**

Syntax: filter(filterType,[filterParam])
Parameters:
filterType (constants): THRESHOLD, GRAY, OPAQUE, INVERT, POSTERIZE, BLUR, ERODE, DILATE or BLUR.
filterParam (number): an optional parameter unique to each filter
Example:

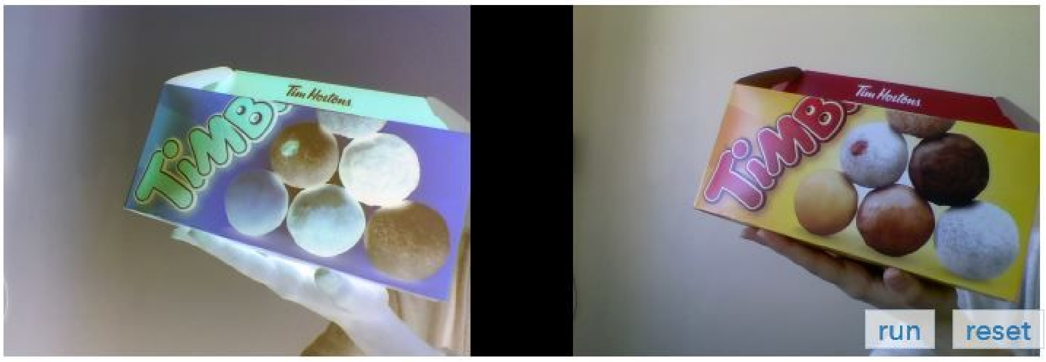
```
        var img;
        function preload() {
                img = loadImage('assets/bricks.jpg');
        }
        function setup() {
                image(img, 0, 0);
            filter(THRESHOLD);
        }
```

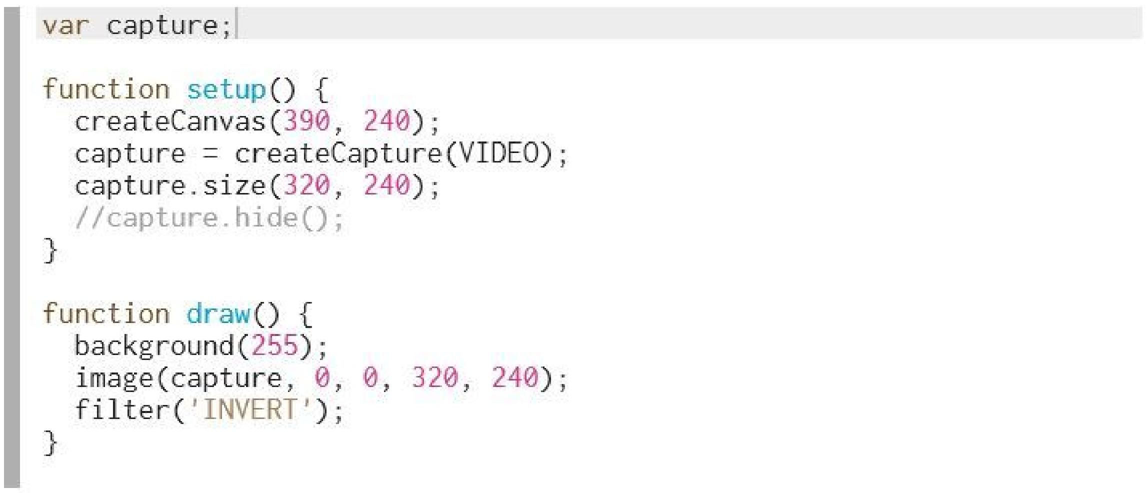
*THRESHOLD - Converts the image to black and white pixels depending on the level parameter which has values between 0.0 (black) and 1.0 (white). 0.5 is the default value.] [6]
[6] Reference: https://p5js.org/reference/#/p5/filter

[**Kinect and p5.js**
Kinectron is a node-based library used for receiving real-time Kinect streams in the browser and in creative coding libraries like p5.js. It can send Kinect depth, color and skeletal data over a peer network, broadcast Kinect data across local or global networks to one (or many) users, and combine Kinect streams from more than one locations in one single webpage.] [7]
[7] Reference: https://kinectron.github.io/

[**blendMode() in p5.js**
Syntax: blendMode(mode) - blends the pixels in the display window according to the defined mode.
Parameters: mode (constant) - BLEND (linear interpolation of colours: C = Afactor + B. This is the default blending mode), DARKEST, LIGHTEST, DIFFERENCE, MULTIPLY, EXCLUSION, SCREEN, REPLACE, OVERLAY, HARD_LIGHT, SOFT_LIGHT, DODGE, BURN, ADD (sum of A and B) or NORMAL] [8]
[8] Reference: https://p5js.org/reference/#/p5/blendMode

[**Video Capture**
Capture video from the webcam and display on the canvas as well with invert filter.
var capture;





```
var capture;

function setup() {
  createCanvas(390, 240);
  capture = createCapture(VIDEO);
  capture.size(320, 240);
  //capture.hide();
}

function draw() {
  background(255);
  image(capture, 0, 0, 320, 240);
  filter('INVERT');
}
```

] [9]
[9] Reference: https://p5js.org/examples/dom-video-capture.html

[**WebGL in p5**
WEBGL allows 2D and 3D shapes. This is possible by specifying a third parameter (third dimension Z) in the createCanvas() function.
Example:

```
function setup() {
    createCanvas(200, 200, WEBGL);
}
```

] [10]

[10] Reference and for more information:
https://github.com/processing/p5.js/wiki/Getting-started-with-WebGL-in-p5)

Run JavaScript on Java backend

Run JavaScript file on the server side:
https://www.ibm.com/developerworks/web/library/wa-aj-javaee/index.html
Call remote JavaScript with Ajax:
https://www.ibm.com/developerworks/web/library/wa-aj-javaee2/index.html
Use Java scripting API with JSP:
https://www.ibm.com/developerworks/web/library/wa-aj-javaee3/index.html

OpenNI2/SimpleOpenNI

[The OpenNI 2.0 API provides access to PrimeSense compatible depth sensors. It allows an application to initialize a sensor and receive depth, RGB, and IR video streams from the device. It provides a single unified interface to sensors and .ONI recordings created with depth sensors.

Getting access to the depth streams requires the use of four main classes:

1) openni::OpenNI – Provides a single static entry point to the API. Also provides access to devices, device related events, version and error information. Required to enable you to connect to a Device.

2) openni::Device – Provides an interface to a single sensor device connected to the system. Requires OpenNI to be initialized before it can be created. Devices provide access to Streams.

3) openni::VideoStream – Abstracts a single video stream. Obtained from a specific Device. Required to obtain VideoFrameRefs.

4) openni::VideoFrameRef – Abstracts a single video from and related meta-data. Obtained from a specific Stream.] [11]

[11] Reference:
https://s3.amazonaws.com/com.occipital.openni/OpenNI_Programmers_Guide.pdf