DMD 센싱, 인터페이싱 중간 보고서

아두이노, 오픈프레임웍스를 활용한 인터랙티브 아트의 기초

V2017111 박형준

1. 서론

기존의 전통적인 예술작품들이 관객에게 수동적인 감상만을 요구해온 반면 현대의 많은 미디어아트는 관객과 상호작용하는 인터랙티브 아트를 지향한다. 인터랙티브 작품에서 예술가는 적절히 통제된 환경 안에서 관객이 자유롭게 작품을 변화시키고 체험할 수 있도록 유도한다.

관객에게 반응하는 '능동적인' 작품을 만들기 위해서는 우선 변화되는 외부 조건을 받아들이기 위한 센서가 필요하다. 그리고 센서를 통하여 얻어지는 실시간 정보를 컴퓨터에 입력 받는 통신작업이 수행되며, 이러한 정보를 알고리즘화 하여 창작자가 원하는 방식으로 표현하기 위한 프로그래밍 작업이 이어진다.

이번 연구에서는 주변의 밝기에 따라 화면에 표현되는 도형의 크기가 달라지는 간단한 실험을 통하여 인터랙티브 아트의 기초를 연습해본다. 센서는 일반적인 CDS조도센서, 시리얼 통신에는 아두이노를 사용하였다. 오픈프레임웍스 라이브러리를 사용한 C++코딩으로 프로그램을 구성하였다..

2. 센서

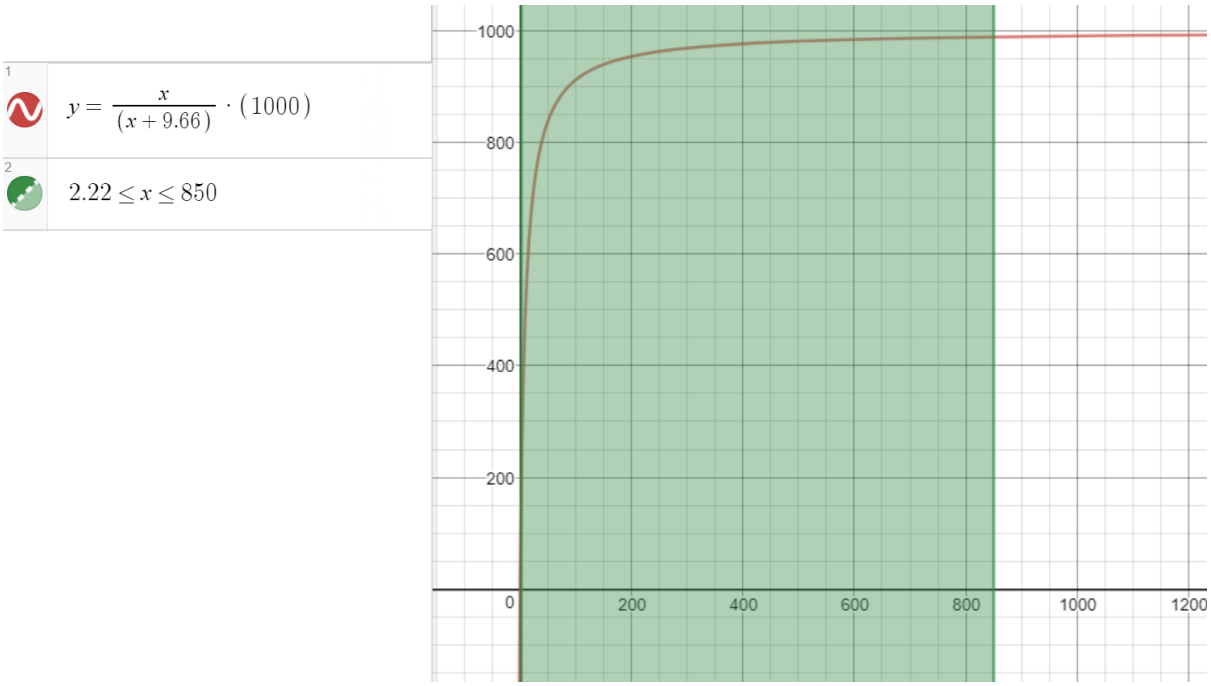
센서는 기본적으로 가변저항의 하나로써 저항 크기를 변화시키는 요인에 따라 온도 센서, 압력 센서 등 다양한 종류가 존재한다. 조도센서는 조도, 즉 밝기를 탐지하는 센서로써, 여기서 사용된 CDS센서는 황화 카드뮴(cadmium sulfide)이 빛에 따라 저항이 달라지는 성질을 이용한다.

사진 : CDS 조도센서

우선 사용될 센서에 가장 밝은 환경과 가장 어두운 환경을 적용하여 저항 값의 범위를 측정하였다. 멀티미터를 사용하여 측정한 결과 가장 밝은 빛을 비추었을 때 센서의 저항 값은 2.22kΩ, 가장 어두울 때의 저항 값은 850kΩ이었다.

측정된 센서의 저항 크기 범위를 고려하여 직렬로 연결할 저항을 선택하였다. 센서의 범위를 가능한 한 넓게 활용할 수 있도록 10kΩ크기의 저항을 연결하였다. 색띠로 읽는 값과 약간의 오차가 있을 것을 고려하여 실제 멀티미터로 측정한 결과 저항 크기는 9.66kΩ이었다.

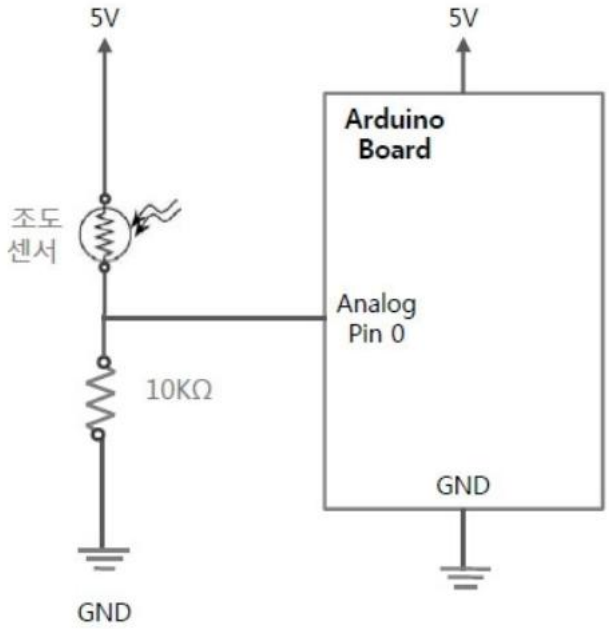
따라서 두 저항을 직렬로 연결한 상태에서의 조도센서의 저항 값은 $\frac{2.22}{9.66+2.2} \leq R \leq \frac{850}{9.66+850}$, 즉, 약0.19kΩ~0.99kΩ(190Ω~990Ω)의 범위를 지니게 된다.



그래프 : 밝기에 따른 센서의 저항 값 변화. 변화량을 그래프상으로 더욱 크게 보기 위하여 y값에만 상수 1000을 곱해주어 세로축을 Ω단위로 표현하였다.

3. 시리얼 통신

전원을 공급하고 저항 값을 실시간으로 읽어서 컴퓨터로 전송하는 도구로써 아두이노를 사용하였다. 아두이노는 오픈소스를 기반으로 한 단일 보드의 마이크로 컴퓨터로 회로를 구성하여 각 핀에 연결한 뒤 sketch 프로그램을 통해 간단한 코드를 작성하여 보드에 업로드 하면 이를 반복적으로 수행한다.



회로도 : 5V의 전원을 공급하고 A0 pin을 통해 조도센서 구간의 전압을 읽는다.

Arduino Sketch프로그램으로 data rate를 설정하고 A0핀의 값을 읽는 코드를 아래와 같이 작성하였다.

```
void setup() {
        Serial.begin(115200);
}

void loop() {
        Serial.println(analogRead(A0));
}
```

setup()안의 Serial.begin()함수는 아두이노에서 정보를 읽어 들일 속도를 결정하는 함수로 여기서는 최대 속도인 초당 115200bit로 설정하였다.

loop()부분의 Seial.printIn()함수는 데이터를 사용자가 읽을 수 있도록 ASCII코드로 변환하여 전달한다. 연속적으로 들어오는 정보에 대해 각 데이터의 끝에 return('\r')과 new line('\n')을 붙여서 구분 짓는다.

analogRead()는 해당 핀으로 들어오는 0~5V 사이의 전압을 내장된 ADC(analog to digital converter)을 통해 0~1023값의 숫자로 변환한다. 앞서 조도센서가 가질 수 있는 저항 값의 범위가 약 190Ω~990Ω 이었으므로, 이 구간의 해당하는 전압은 약 0.95~4.95V의 값을 가지게 된다. 따라서 analogRead()를 통해 변환된 수치는 194~1012 정도의 범위에서 움직일 것으로 예상할 수 있다.

4. 프로그래밍-Library Setting

Openframeworks는 다양한 클래스와 함수, 또 많은 예제들을 포함한 C++오픈 소스 라

이브러리로써 다양한 미디어 데이터를 만들고 조작할 수 있도록 만들어졌다. 복잡한 수학적 계산이 필요한 3D 오브젝트의 생성 및 움직임 등을 표현하고자 할 때 모든 과정을 처음부터 계산하지 않아도 라이브러리에 있는 함수를 사용하여 해결할 수 있도록 하였다.

오픈프레임웍스에 포함된 project generator로 라이브러리를 포함한 새 프로젝트를 만든 상태에서 코드를 작성한다. 소스는 크게 두 개의 cpp코드(main, ofApp)와 헤더파일(ofApp.h)로 되어있다. 헤더파일과 main코드 그리고 ofApp의 기본 틀은 이미 기본적으로 작성된 상태에서 코딩을 시작할 수 있다. main.cpp는 아래와 같다.

```
#include “ofMain.h”
#include “ofApp.h”

int main() {
        ofSetupOpenGL(1024, 768, OF_WINDOW);
        ofRunApp(new ofApp());
}
```

ofMain.h 파일은 ofApp.h를 제외한 라이브러리 상의 모든 헤더파일을 포함한 파일로써 Light, Color, Serial등 오픈프레임웍스 안의 모든 기능을 사용할 수 있도록 해준다. main 함수 안의 ofSetupOpenGL()로 지정된 크기의 디스플레이 화면을 만들고 ofRunApp()으로 ofApp파일을 실행하여 원하는 기능을 수행한다. ofApp는 크게 setup(), update(), draw() 세 부분으로 구성되며(다른 많은 부분들은 이번 작업에서는 사용하지 않는다.) setup()으로 초기화 한 뒤 update()와 draw()를 반복적으로 수행하게 된다. 이 세 함수는 헤더파일 ofApp.h의 클래스 안에 선언되어있다.

```
#pragma once

#include "ofMain.h"

class ofApp : public ofBaseApp{

        public:
                void setup();
                void update();
                void draw();

                ofSerial                mySerial;

};
```

헤더파일의 코드는 따로 변경하지 않았으나 시리얼 통신관련 함수를 사용하기 위하여 mySerial이라는 이름의 ofSerial 클래스를 하나 선언해주었다.

5. 프로그래밍-Code 작성

프로그램이 실행되면 가장 먼저 수행할 초기화 부분을 다음과 같이 작성하였다.

```
void ofApp::setup(){

	mySerial.listDevices();
	vector <ofSerialDeviceInfo> deviceList =
					mySerial.getDeviceList();

	int baud = 115200;
	mySerial.setup("COM3", baud);

	if (mySerial.isInitialized() == true) {
		cout << "serial Initialized" << endl;
		mySerial.flush();
	}
	else {
		cout << "serial not Initialized" << endl;
		return;
	}

	ofBackground(54, 54, 54, 255);
}
```

ofSerial 클래스의 setup함수는 해당 포트와 샘플레이트를 입력받아 초기화한다. 이름(COM3)으로 사용할 USB 포트를 지정해 주고 이전의 아두이노 코드에서 설정한 속도와 동일하게 115200으로 초기화 하였다. 초기화 여부를 Bool타입으로 나타내는 isInitialized()함수를 통해 초기화가 되었으면 클래스의 버퍼를 비우고 아니면 에러 메시지를 출력하도록 하였다. 더불어ofBackground()로 of윈도우의 기본 배경색을 rgb값 54(회색)으로 설정하였다.

```
void ofApp::update(){

	if (mySerial.available() > 0) {

		const int bufSize = 1024;
		unsigned char buf[bufSize];
		int bufReq = mySerial.available() > bufSize ?
			bufSize : mySerial.available();

		mySerial.readBytes(buf, bufReq);

		for (int i = 0; i < bufReq; i++) {
			serial2int(buf[i]);
		}
	}
}
```

update에서는 포트로 계속적으로 들어오는 값들을 integer array에 저장한다. available()함수는 몇 개의 byte가 시리얼 포트에서 읽힐 수 있는지를 나타내는데 buf라는 배열을 만든 뒤 1024개byte 이내의 정보들을 저장하도록 하였다. 아두이노는 각각의 숫자 값을 ASCII코드로 변환하여 한byte씩 내보내는데 0~1023의 값은 최대 네

자리까지 표현되기 때문에 이를 유의미한 정보로 취급하기 위해서는 숫자 한 개씩 들어오는 정보를 단위 별로 합쳐 재구성할 필요가 있다. 여기서는 serial2int()함수를 만들어 들어오는 ASCII코드들을 모아 문자열로 합친 뒤 return('\r')이 들어오면 10진수로 변환하여 숫자열(a2d)에 배치하도록 하였다.

```
std::string str;
std::vector<int> a2d;

void serial2int(int byteRead) {

	int myByte = byteRead;

	if (myByte == OF_SERIAL_NO_DATA) {
		cout << "no data was read" << endl;
	}
	else if (myByte == OF_SERIAL_ERROR) {
		cout << "en error occurred" << endl;
	}
	else if (myByte >= '0' && myByte <= '9') {
		str.push_back((char)myByte);
	}
	else if (myByte == 10 && !str.empty()) {
		int num = std::stoi(str);
		str.clear();
		a2d.push_back(num);
	}
	else {
	}
}
```

지금까지 센서의 값(정확히는 센서의 상태에 따라 변하는 전압의 양을 아두이노가 1024단계의 숫자로 계산, 이를 ASCII코드로 변환하여 한 개씩 전송한 문자들)을 다시 유의미한 단위로 저장하는 과정을 살펴보았다. 이제 이 값에 따라 화면에 표시되는 원의 크기가 변화하도록 하는 프로그램을 구성하려 한다. draw파트에서는 of윈도우 상에 원을 하나 표시하고 가장 최근에 들어온 a2d의 값을 원의 크기에 변수로 넣는다.

```
void ofApp::draw() {

	if (a2d.empty()) {
		printf("a2d empty\n");
		return;
	}
	else {
		int val = a2d.back();

		ofSetColor(255, 255, 0);

		float xCenter = ofGetWidth()/2;
		float yCenter = ofGetHeight()/2;

		ofDrawCircle(xCenter, yCenter, radiousScale(val));
	}
}
```

아두이노에서 보낸 값이 저장되기 이전에 draw가 실행되었을 경우는 a2d벡터가 비어있다는 메시지만 출력하고 다시 처음으로 돌아간다.

ofDrawCircle()함수를 이용하여 창에 원을 표시할 수 있는데, 이 함수는 중심의 x좌표, y좌표와 반지름의 길이를 입력으로 받는다. 오픈프레임웍스 윈도우상에서 좌표는 왼쪽 위를 (0,0)으로 하여 오른쪽으로 갈수록, 또 아래쪽으로 갈수록 숫자가 늘어나는 식으로 되어있다. 화면의 중심에 원을 위치시키기 위해서는 가로길이의 절반지점, 세로길이의 절반지점을 각각 x, y로 설정하여야 한다. 처음 main에서 ofSetupOpenGL(1024, 768, OF_WINDOW)가 윈도우의 가로세로 길이를 입력 받았다는 점을 기억한다면 (512, 384)로 원의 중심을 지정해줄 수도 있겠으나 이는 창의 크기가 변할 경우(가령 Full Screen으로 바꾸었을 경우)에 다시 도형이 치우치는 결과를 낳게 된다. 그리하여 ofGetwidth()와 ofGetHeight()함수를 사용하여 x와 y의 중앙값을 취하였다.

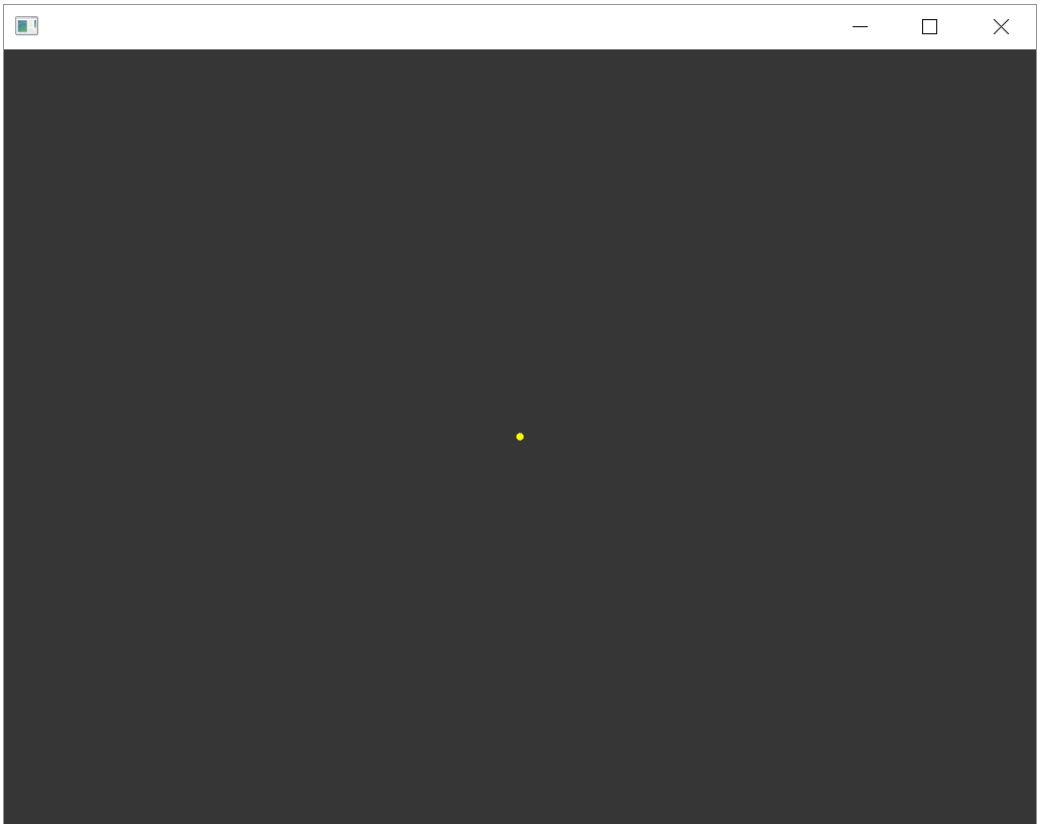
ofDrawCircle()의 세 번째 입력변수인 radious(반지름)가 원의 크기를 결정짓는 요소인데 이는 아래의 함수를 사용하여 조절되도록 하였다.

```
float radiousScale(int num) {
	float inMin = 1012, inMax = 194;
	float outMin = 0, outMax = ofGetHeight() / 2;

	float rad = (num - inMin)*(outMax - outMin) /
                                        (inMax - inMin);
	return rad;
}
```

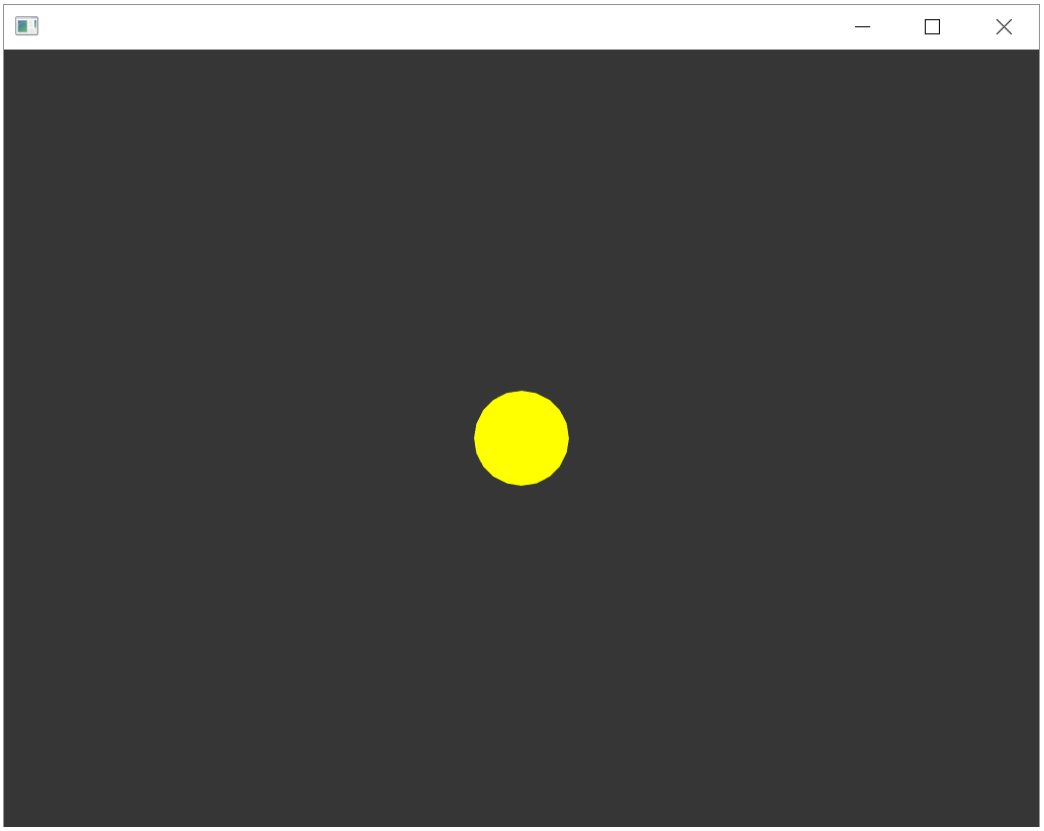
194~1012의 범위를 지닌 숫자 값은 0부터 화면 높이의 절반까지로 규격화된다. 원이 가장 커졌을 때에도 화면을 벗어나지 않도록 하기 위함이다. 주변이 밝아질수록 조도센서의 저항 값이 낮아지고 전압 역시 낮아지는데 이 실험은 반대로 밝아질수록 원이 커지는 것을 목표로 하기 때문에 inMin값과 inMax값을 반대로 설정하였다.

6. 결과

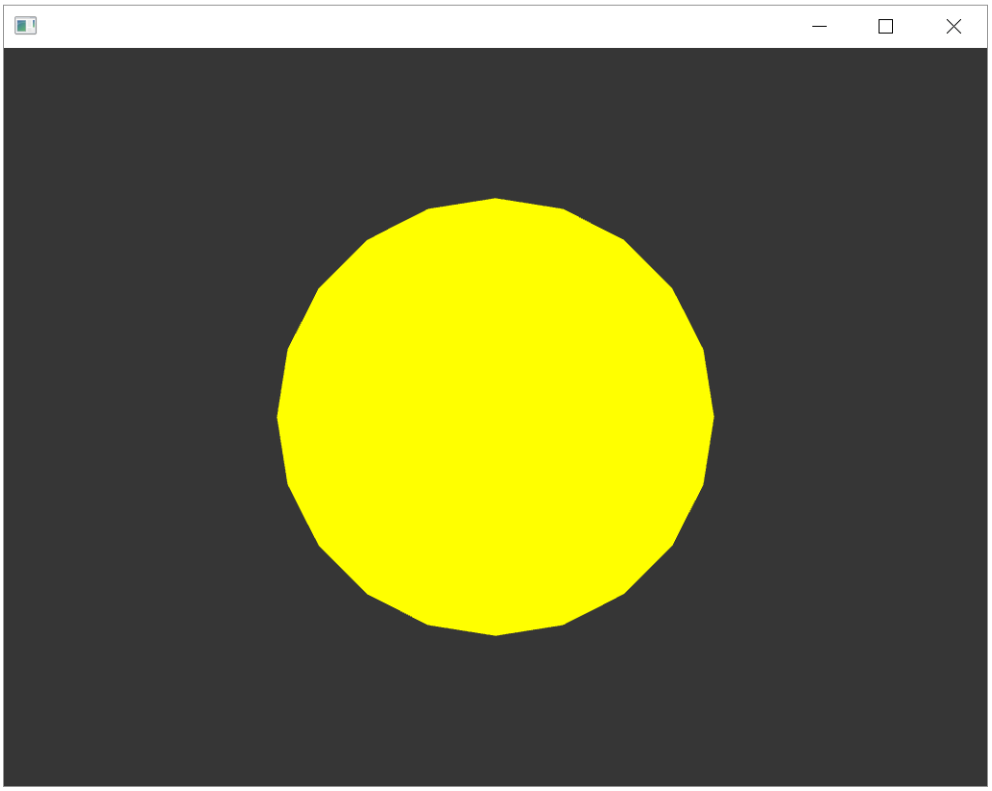
밝기의 변화에 따라 센서가 빠르게 반응하였고 어둡고 밝은 정도에 따라 표현할 수 있는 원의 크기 범위도 최대한으로 활용되었다. 손가락으로 조도 센서의 윗부분을 박자에 맞춰서 두드리면 원이 리듬감있는 움직임을 보여주기도 하여 터치식 스위치처럼 활용할 수도 있을 듯 하다.



캡처화면 : 아주 어두운 환경에서의 화면



캡처화면 : 불이 꺼진 방에서의 화면



캡처 화면 : 불이 켜진 방에서의 화면

캡처 화면 : 아주 밝은 빛을 비추었을 때의 화면

7. 응용

실습한 프로그램을 바탕으로 이를 좀더 발전시키는 방안을 모색해 본다. '밝기에 따라 크기가 변하는 원'이라는 개념은 인간의 눈에서도 찾을 수 있다. 눈동자, 동공은 빛이 조절되어 들어가는 입구로써 빛의 양을 조절하기 위하여 수축, 팽창한다. 실제로 빛에 따라 크기가 변하는 동공의 이미지를 화면에 표현함으로써 관객과 작품이 서로를 바라보며 반응하는 작품을 구현해본다.

눈 이미지를 배경으로 사용하기 위하여 ofImage 클래스를 헤더에 추가해준 뒤에 이미지 파일을 프로젝트 폴더의 bin/data/images안에 넣어준다. 기존의 ofBackground(54,54,54,255) 대신 setup부분에 아래의 한 줄을 추가해준다.

```
myImage.load("images/coloredEyeImage.jpg");
```

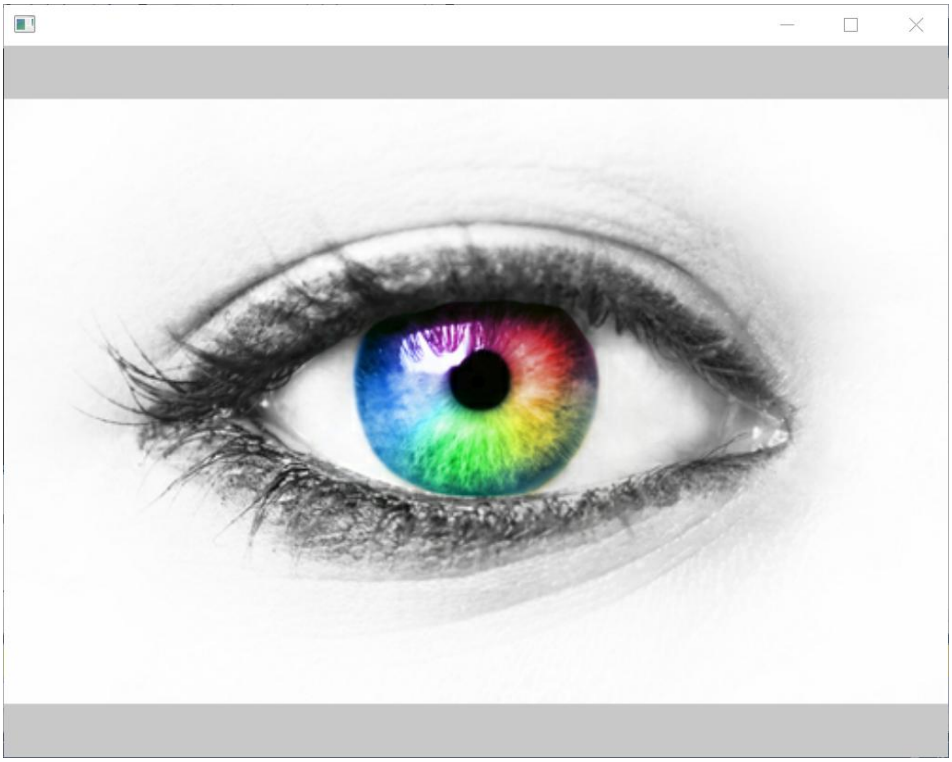
draw 부분에서는 이미지 파일이 본래의 비율을 유지하며 배경에 자리잡도록 하고 눈동자에 해당되는 부분이 원의 중심이 되도록 비율을 잡은 뒤에 눈 크기 안에서 크기가 변하도록 한다.

```
void ofApp::draw() {

	ofSetColor(255, 255, 255, 255);
	float imageHeight = ofGetWidth() * 319 / 500;
	float imageY = (ofGetHeight() - imageHeight) / 2;
	myImage.draw(0, imageY, ofGetWidth(), imageHeight);

	if (a2d.empty()) {
		printf("a2d empty\n");
		return;
	}
	else {
		int val = a2d.back();

		float xPupil = ofGetWidth() * 251 / 500;
		float yPupil = ofGetHeight() * 151 / 319;

		ofSetColor(0, 0, 0);
		ofDrawCircle(xPupil, yPupil, pupilScale(val));
	}
}
```

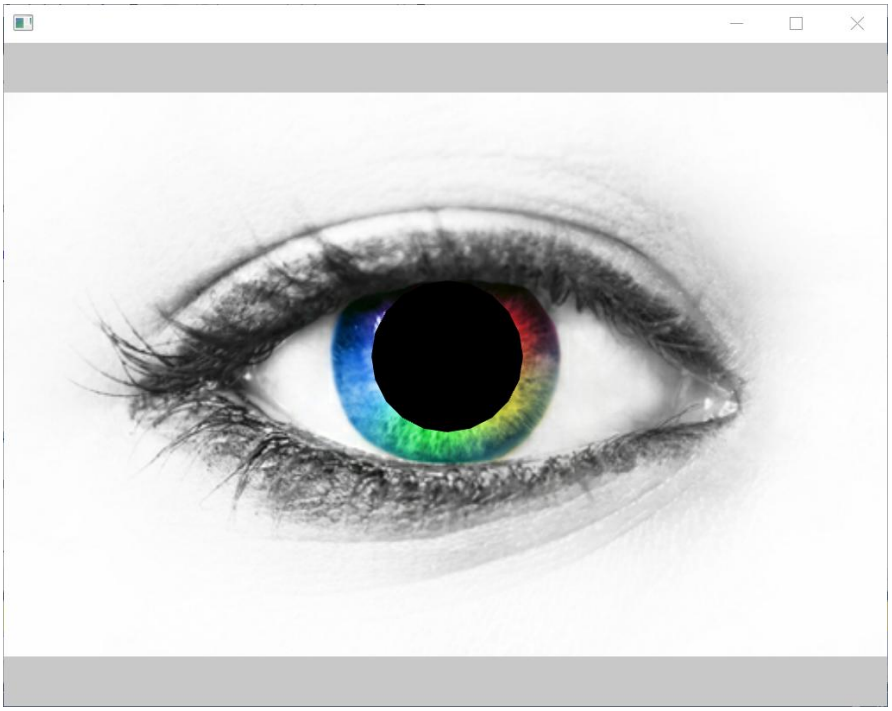
ofImage.draw를 사용하기 전에 투명한 색을 설정하여 원의 색깔이 전체 이미지를 덮어버리지 않도록 한다. 이미지의 비율이 다소 가로로 긴 모양(500*319)이기 때문에 창 크기에 가로를 맞추고 높이는 이미지 비율에 맞추어 가운데에 오도록 한다. 이미지 안에서 눈동자의 중심이 위치하는 픽셀좌표를 기준으로 ofDrawCircle의 x좌표와 y좌표를 찾고 원의 크기 역시 이미지상의 눈의 크기를 전체 이미지와의 비율로 계산하여 다음과 같은 Scale함수를 만들었다.

```
float pupilScale(int num) {
        float inMin = 194, inMax = 1012;
        float outMin = 0, outMax = ofGetHeight() * 40 / 319;
        float pupil = (num - inMin)*(outMax - outMin)
                                                / (inMax - inMin);
        return pupil;
        }
```

동공의 크기를 어두울수록 크게, 밝을수록 작아지게 하기 위하여 inMin, inMax값은 다시 순서대로 조정되었다.



캡처 화면 : 밝을 때의 화면



캡처 화면 : 어두울 때의 화면

8. 결론

센서의 값에 따라 화면에 오브젝트를 표현하는 간단한 연습을 해 보았고 이를 활용하여 유동적인 이미지 표현을 수행하였다. 기능을 오류 없이 구현하고 프로그램이 작동하는데 있어서는 별다른 문제가 없었으나 오픈프레임웍스에서 제공되는 함수에 대한 깊은 이해나 이미지의 작은 부분들까지 세세하게 처리하는 작업 등이 부족하여 다소 어색하고 부자연스러운 결과물에 그치게 되었다. 원이 부드러운 곡선으로 표현되지 않고 각진 원의 형태를 띠거나 눈동자 주변에 이미지 처리가 안되어 배경 그림과 도형이 따로 떨어져 보이거나 조명에 따라 영향을 받는 과정이 표현되지 못한 것 등 향후 더 학습을 거쳐 발전시킬 수 있는 여지가 많이 남아있다고 여겨진다.

본문에 사용된 프로그램, 소스코드는 https://github.com/gud9325/CDS-Circle 에서 다운로드 할 수 있다.