DMD 센싱, 인터페이싱 학기말 보고서

아두이노, 오픈프레임웍스를 활용한 디지털 악기 제작

V2017111 박형준

1. 서론

이전까지 전통적인 악기는 공기의 진동을 만들어내는 방법과 재질에 따라 현악기, 관악기, 타악기등으로 분류되어왔다. 오실레이터(Oscillator)의 등장 이후로는 전자 악기가 생겨나게 되었고 기존의 방식과는 다른 새로운 방식으로 공기의 진동을 만들어낼 수 있게 되었다. 전자 악기는 다양한 소리, 파형의 조합(Synthesize)을 통해 자연에서 얻어내기 어려운 특별한 소리들을 만들어 내는 데 일조해 왔고 전자 음악이라는 새로운 음악 분야를 만들어내기도 하였다.

컴퓨터를 활용한 현대의 전자 악기는 앞서 언급한 소리 합성이나 생성과 더불어 기존의 악기를 대체하는 가상 악기의 역할을 더욱 많이 수행한다. 실제 악기 소리를 모방하여 컴퓨터로 연주할 수 있도록 한 가상 악기는 현재 많은 음악가들에 의해 경제적, 실용적인 이유로 사용되고 있다.

본문에서는 이 가상 악기의 원리를 이용하여 녹음된 악기 소리를 센서를 통해 재생시키는 간단한 디지털 악기를 제작함으로써 센싱과 인터페이싱을 연습해보도록 한다.

2. 작품 구상 및 소재 선택

키넥트로 동작을 인식하여 소리에 반영하거나 기울기 센서, 압력센서 등 다양한 센서를 활용하여 독특한 연주 기법을 만들어낼 수도 있겠지만 너무 비효율적인 연주 방식은 단순히 신기함에서 그치고 음악적으로 활용이 어려울 수 있다. 이번 작품에서는 이미 스마트폰을 통해서 많은 사람에게 익숙해진 터치 방식으로 샘플 음원을 재생시켜 쉽고 직관적으로 연주할 수 있는 악기를 만들기로 한다.

스마트폰에 있는 터치스크린 패널은 정전기를 이용하여 입력 위치를 인식하게 되는데 본 작품에서는 각각의 샘플을 손가락으로 재생할 수 있는 다섯 개의 입력이면 충분하기 때문에 더 간단한 개별 센서들로 구성한다. 센서로는 터치를 인식하는 터치 센서를 사용할 수 도 있지만 더 쉽고 저렴하게 같은 효과를 구성하기 위하여 조도 센서를 사용하였다. 조도 센서는 빛의 밝기에 따라 저항 값이 바뀌지만 손가락 등으로 접촉했을 때 아주 어두운 상태와 같은 효과를 갖기 때문에 터치 센서처럼 사용할 수 있다.

3. 재료 음원

센서의 입력에 따라 재생될 음원들은 wave파일로 따로 준비해둔다. 디지털 악기는 전통적인 악기와는 달리 하나의 악기가 하나의 소리로 제한되지 않기 때문에 이 장점을 활용하기 위해 악기를 여러 세트로 구성하여 바꿔가며 연주할 수 있도록 하였다. 각각 5개의 소리를 가진 악기를 4개 세트로 구성하여 총 20개의 wave파일을 미리 준비하였다. 악기는 Drum set, Hangdrum(Stealdrum), Marimba, Woodblock 네 종류이며 손가락으로 터치할 때와 음색적인 연관성이 느껴지도록 타악기 위주로 선정하였다.

오픈프레임웍스의 `ofSoundBuffer()`가 이미 파일의 헤더값에서 channel, sample rate, bit rate등 필요한 정보를 받아서 처리하도록 설정되어 있지만 음원은 모두 2 channel(stereo), 44100hz의 샘플레이트, 24bit의 비트레이트의 기본적인 포맷으로 통일하였다. 터치 입력 시 즉시 소리가 날 수 있도록 offset 없이 00:00초에 소리가 시작되도록 하였고 리버브를 충분히 두어 끊김이 없도록 하였다.

악기 소리와 더불어 배경음악 트랙을 따로 두었는데, 사용자가 원하면 배경음악을 튼 상태로 악기를 연주할 수 있도록 하기 위함이다. 배경음악은 계속 반복되어도 끊기는 부분이 없도록 끝부분과 첫 부분이 이어지도록 작곡하였고 사용자의 의도에 의해서만 끄고 켜지도록 설정한다.

4. 회로 구성

센서는 아두이노를 통해 구간의 전압 값을 컴퓨터로 전달하게 된다. 다섯 개의 조도 센서를 각각 아두이노의 아날로그 핀(A0~4)에 연결하여 개별적으로 작동하도록 하고 10kΩ의 저항을 한 개씩 연결해준다.(저항은 멀티미터를 사용하여 가장 밝은 빛을 비추었을 때와 가장 어두울 때 조도 센서의 저항 값을 측정하고 이를 바탕으로 계산하여 조도 센서의 가능한 넓은 범위의 값을 가질 수 있도록 선택)

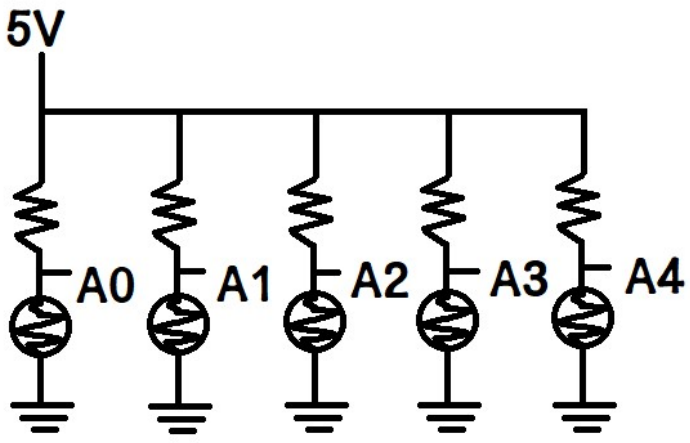


그림: 회로도 - 5개의 센서와 저항을 병렬로 연결하였다.

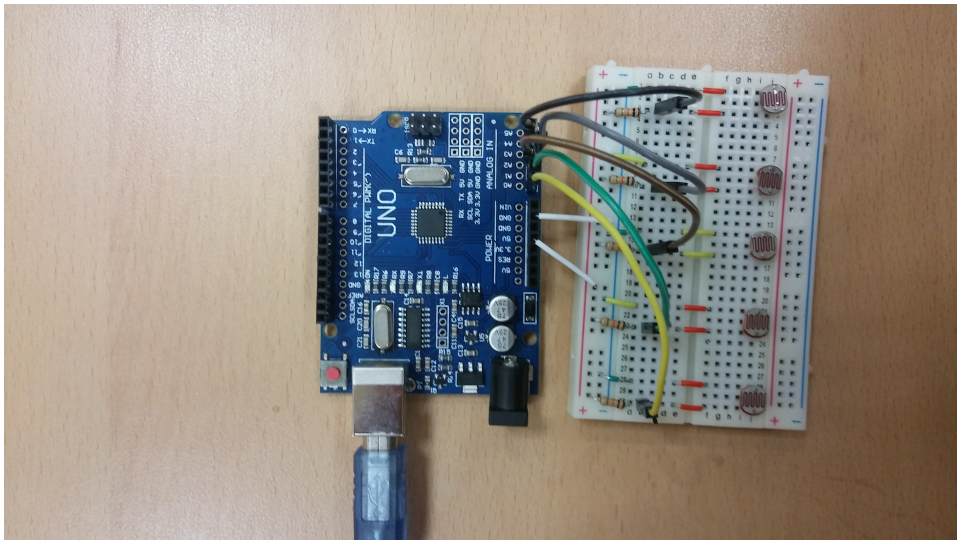


사진 : 브레드 보드에 연결한 모습

5. Code 작성-작품 알고리즘

우선 아두이노 스케치를 이용하여 아날로그 핀의 값을 컴퓨터로 전송하도록 설정하였다. A0~A4의 다섯 개의 핀을 사용하는데 각각의 핀은 0~1023까지의 값을 갖게 된다. 연속적으로 전송되는 핀의 값을 구분하기 위해 각 핀에 1024씩의 숫자를 더하여 범위를 바꿔주었다. 따라서 A0핀은 0~1024, A1핀은 1024~2047, A2핀은 2048~3071... 의 값을 가지게 되고 이는 오픈프레임웍스의 코드 상에서 다시 더했던 값을 빼면서 변수를 나눠줌으로써 각 핀별로 시리얼을 저장하도록 한다.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(57600);
}

void loop() {
  // put your main code here, to run repeatedly:

  Serial.println(analogRead(A0));
  Serial.println(analogRead(A1) + 1024);
  Serial.println(analogRead(A2) + 2048);
  Serial.println(analogRead(A3) + 3072);
  Serial.println(analogRead(A4) + 4096);

}
```

오픈프레임웍스에서는 muySerial이라는 ofSerial클래스를 선언한 뒤에 readBytes()함수를 통해 값을 하나씩 읽어와 처리한다.

```
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]);
		}
	}
}
```

ASCII 코드로 하나씩 들어오는 정보를 유의미한 단위로 합쳐 다시 숫자로 변환하기 위해 serial2int()라는 함수를 만들어 들어오는 ASCII코드들을 모아 문자열로 합친 뒤 return('\r')이 들어오면 10진수로 변환하여 숫자열(a2d)에 배치하도록 하였다. 이때 각각의 핀을 저장하는 벡터 아두이노 코드에서 더했던 값을 다시 빼주면서 나누어서 저장한다.

```
std::string str;
std::vector<int> a2dA0, a2dA1, a2dA2, a2dA3, a2dA4;

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;
	}
	if (myByte >= '0' && myByte <= '9') {
		str.push_back((char)myByte);
	}
	else if (myByte == 10 && !str.empty()) {
		int num = std::stoi(str);
		str.clear();
		if (num < 1024)
			a2dA0.push_back(num);
		else if (num < 2048)
			a2dA1.push_back(num);
		else if (num < 3072)
			a2dA2.push_back(num);
		else if (num < 4096)
			a2dA3.push_back(num);
```

```
                else
                        a2dA4.push_back(num);
        }
        else {
        }
}
```

각각의 벡터에서 취한 정수는 설정된 역치값에 따라 오디오 파일을 재생시킬지를 결정하게 된다.

```
void ofApp::draw() {

        int valA0 = a2dA0.back();
        int valA1 = a2dA1.back();
        int valA2 = a2dA2.back();
        int valA3 = a2dA3.back();
        int valA4 = a2dA4.back();
        if (valA0 > 900) {
                switch (Counter1 % 4) {
        case 0:
                if(!Drum1.isPlaying()
                        || Drum1.getPosition() > 0.04)
                        Drum1.play();
                break;
        case 1:
                if (!Hangdrum1.isPlaying()
                        || Hangdrum1.getPosition() > 0.04)
                        Hangdrum1.play();
                break;
        case 2:
                if (!Marimba1.isPlaying()
                        || Marimba1.getPosition() > 0.04)
                        Marimba1.play();
                        break;
        case 3:
                if(!Woodblock1.isPlaying()
                         || Woodblock1.getPosition() > 0.04)
                        Woodblock1.play();
                break;
        }
        …
        …
        //(A1~A4에도 동일한 방식으로 적용)
```

손가락을 가져다 대었을 때와 그렇지 않을 때를 구분하는 기준값을 실험결과 900으로 설정하였고 악기의 위치, 조명상태, 센서의 오차 등에 따라 각 핀별로 약간씩 조정을 하였다.

아주 빠른 속도로 반복되기 때문에 잠깐의 접촉에도 해당되는 값이 연속적으로 매우 많이 나타날 수 있고 이는 오디오 파일이 순간적으로 여러 개가 겹쳐 재생되는 부작용을 낳게 된다. 따라서 각 음원을 재생하기 전에 이미 실행이 되고 있는지를 isplaying()함수를 통해 미리 체크하고 각 악기의 분명한 attack소리가 들리는 지점까지는 오디오가 겹치지 않도록 하기 위하여 getPosition()함수로 이전에 재생된

음원이 얼마나 시간이 지났는지 또한 체크하도록 하였다.

위 부분에서 Counter1이라는 변수가 나머지 연산을 통해서 4개로 나눠지도록 되어있음을 알 수 있는데, 이는 악기 음색을 결정짓는 변수로써 키보드의 Space bar를 누를 때마다 전환되도록 하였다.

```
void ofApp::keyPressed(int key) {

        int instrumentNum = Counter1 % 4;

        if (key == 32) {
        Counter1++;
        }
```

6. code작성-화면 및 음악 구성.

작품을 위해 준비된 오디오는 공간감을 살릴 수 있도록 각기 다른 패닝으로 효과를 주었다.

```
void ofApp::setup() {
//setup instruments
        Drum1.load("Drum1.wav");
        Drum1.setPan(-0.8f);
        Drum2.load("Drum2.wav");
        Drum2.setPan(-0.4f);
        Drum3.load("Drum3.wav");
        Drum3.setPan(0.0f);
        Drum4.load("Drum4.wav");
        Drum4.setPan(0.4f);
        Drum5.load("Drum5.wav");
        Drum5.setPan(0.8f);
        …
        …
        // 이외의 악기들도 동일한 방식으로 구성
```

또한 이 패닝이 화면상에서도 시각적으로 느껴질 수 있도록 하기 위해 간단한 그래픽 효과를 구현하였다. 우선 화면을 가로로 다섯 개로 분할 한 뒤에 각기 다른 색깔로 구분짓고, 해당되는 악기의 음이 재생될 때마다 분할된 화면이 순간적으로 빛나게 하였다.

```
void ofApp::draw() {

        float widthDiv = ofGetWidth() / 5.0f;

        if (Drum1.isPlaying() && Drum1.getPosition() < 0.02 ||
                Hangdrum1.isPlaying() && Hangdrum1.getPosition() < 0.02 ||
                Marimba1.isPlaying() && Marimba1.getPosition() < 0.02 ||
                Woodblock1.isPlaying() && Woodblock1.getPosition() < 0.02)
                ofSetColor(255, 255, 255);
        else
                ofSetColor(90, 0, 0);

        ofDrawRectangle(0, 0, widthDiv, ofGetHeight());
…
…
// 나머지 구간들도 동일한 방식으로 구성
```

배경음악은 Enter키를 이용하여 켜고 끌 수 있도록 하였고 이 역시 악기를 전환할 때와 동일한 방식으로 Counter와 나머지 연산을 이용하였다.

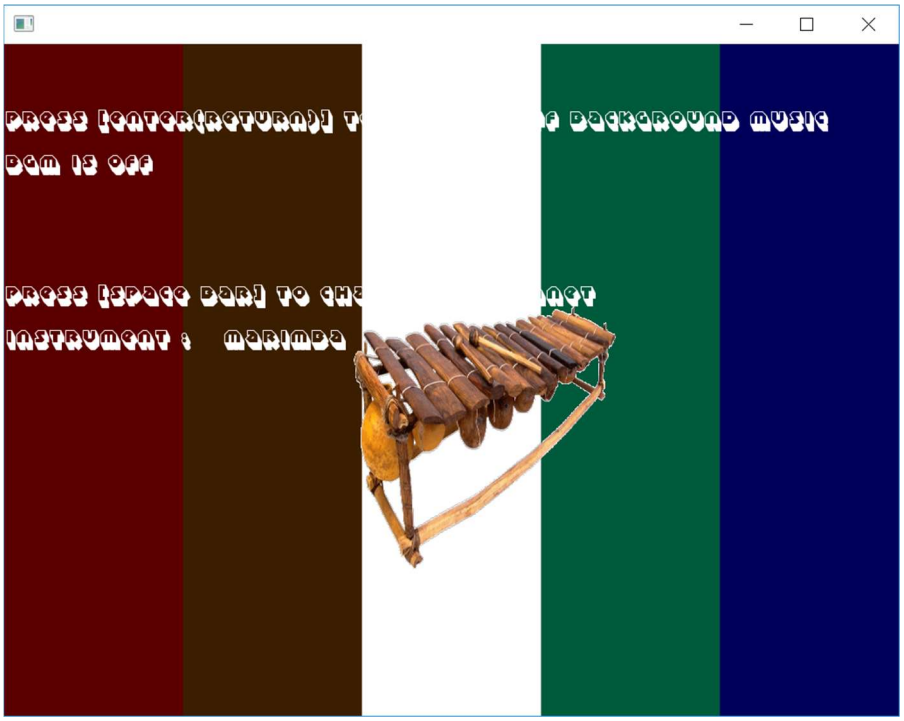
```
void ofApp::keyPressed(int key) {

	int onoff = Counter2 % 2 + 1;

	if (key == 13) {
		Counter2++;
		if (onoff == 1) {
		backgroundMusic.play();
		}
		else {
		backgroundMusic.stop();
		}
	}
}
```

악기가 전환될 때마다 악기 이름과 이미지가 화면에 표시되도록 하여 현재 어떤 악기로 준비되어있는 지를 알 수 있도록 하였다. 아래의 코드는 void ofApp::draw() 안에 작성되었으며 각각의 이미지 파일은 배경을 투명처리한 png파일로 bin/data 폴더 안에 넣어둔 뒤에 헤더에 ofImage 클래스로 선언하고 void ofApp::setup() 부분에서 load()로 세팅해 두었다.

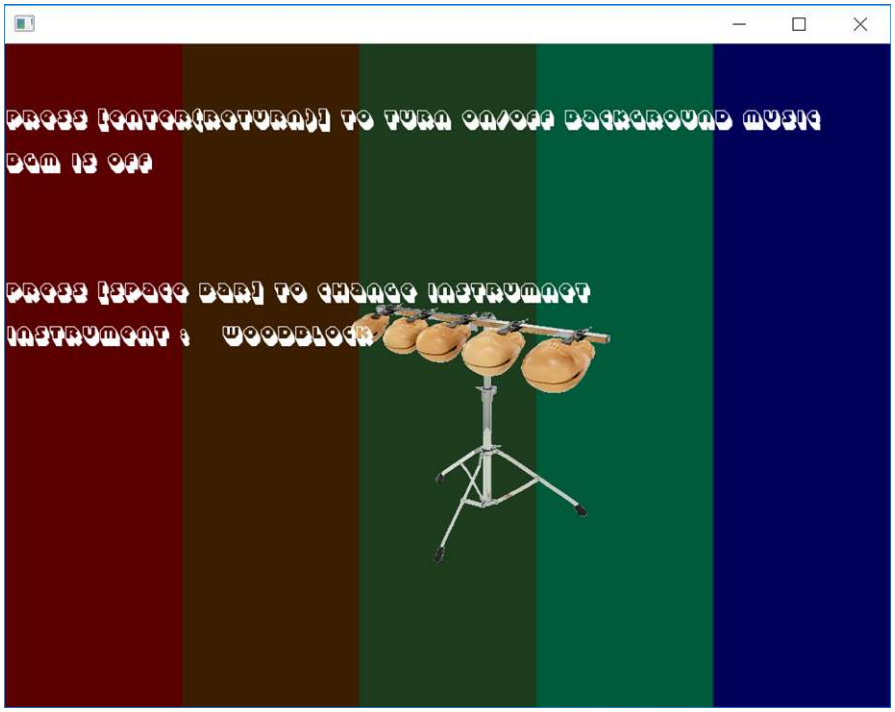
```
font.drawString("Instrument : ", 0, 350);
switch (Counter1 % 4) /* instrumentNum */ {
	case 0:
		DrumImage.draw(400, 300, 300, 300);
		font.drawString("Drum", 250, 350);
		break;
	case 1:
		HangdrumImage.draw(400, 300, 300, 300);
		font.drawString("Hangdrum", 250, 350);
		break;
	case 2:
		MarimbaImage.draw(400, 300, 300, 300);
		font.drawString("Marimba", 250, 350);
		break;
	case 3:
		WoodblockImage.draw(400, 300, 300, 300);
		font.drawString("Woodblock", 250, 350);
		break;
	}
```

7. 결과

전반적인 작품의 기능이 목표한 바 대로 구현되었다. 다섯 개로 분할 된 화면은 악기소리의 순차적인 패닝 위치와 감각적으로 일치되게 느껴진다. 화면을 순간적으로 하얀색으로 바꾸는 것 만으로도 악기가 연주되는 느낌을 직관적으로 표현할 수 있었다. 동시에 여러 센서를 터치하여도 청각적으로 차이를 느낄 만큼의 유의미한 시차가 일어나지 않기 때문에 감상 및 연주에 무리가 없으며 미리 배경음악과 악기의 다섯 음의 음계를 맞춰 두었기 때문에 동떨어진 느낌을 주지 않는다.



캡처 : Marimba악기의 3번째 센서를 터치한 순간의 화면



캡처 : 악기가 우드블락으로 전환되었을 때의 화면

8. 결론 및 향후보완

조도센서를 터치입력 장치로 사용함에 따라 생기는 단점은 주변의 조명 상태, 센서 간의 오차등에 따라 쉽게 값이 편할 수 있다는 점이다. 조도센서는 경제적인 측면에서는 매우 효과적이었지만 아두이도의 아날로그 핀을 통해 맵핑된 값만 해도 무려 1024단계를 표현할 수 있는 센서를 단순한 on/off스위치처럼 사용하였다는 점은 아쉬운 점이라 할 수 있다.

향후 블루투스의 활용으로 선 없이 자체적으로 구동할 수 있게 한다거나 볼륨, 리버브, 필터 등의 다양한 음향효과를 적용하는 방법, 무엇보다도 작품의 외관을 좀더 완성도 있게 마감하는 법 등을 연구 연습해야 할 것이다.

본문의 작업에 사용된 프로그램과 소스코드, 데이터 파일은 https://github.com/gud9325/CDS-Music 에서 다운받을 수 있다.