**설계과제1 최종보고서**

**4조 : 가상현실 드럼(VR DRUM)**

**목차**

1. 개요 --- 2
2. 연구 필요성 및 기대효과 --- 2
3. 용어 정리 --- 4
4. 전체 구조 --- 5
5. 하드웨어 및 소프트웨어 --- 6
6. 알고리즘 설명 --- 7
7. 애니메이션 --- 14
8. 주요 오브젝트 --- 16
9. 진행상황 및 앞으로의 계획 --- 20
10. 참고자료 및 사진출처 --- 21

|  |  |
| --- | --- |
| 연구기간 | 2016.08.22~2016.12.13 |
| 제출일 | 2016.12.26 |
| 대표/행정/제작 | 권원표(20140931) |
| 프로그래밍/개발 | 유정민(20130203) |
| 디자인/애니메이션 | 이종건(20140707) |

1. **개요**

본 설계과제의 목표는 가상현실 속에서 연주할 수 있는 드럼세트를 만드는 것이다. 기존 드럼세트는 가격이 비싸고 부피가 커서 가정집에서는 찾아보기 힘들고, 있다고 하더라도 소리가 커서 이웃 사이의 소음 문제를 발생시킬 수 있다. 그래서 현재는 드럼을 연습하려면 매번 연습실이나 학원에 가야하는 상황이며 이는 문화 및 취미생활을 방해하는 요소가 된다.

우리는 이런 접근성 문제를 해결할 방법으로 가상현실을 생각했다. VR기기를 이용하여 부피를 없애고, 소리는 헤드폰을 통해 연주자만 들을 수 있는 가상드럼을 만들어 부피와 소음 문제를 동시에 해결하는 것이다. 만들고자 하는 시작품은 현실에서 가상현실 기기와 컨트롤러, 발 동작을 인식할 페달 만을 사용하며 그 외에 북, 심벌 등은 보이지 않는 가상현실 속에서 사용자에게만 보이고, 가상현실 속에서 사용자가 스틱으로 타격하면 소리가 나도록 만든다.

우리는 본 설계과제를 만들기 위해서 HTC사의 가상현실 기기 VIVE와 연동 컨트롤러 한 쌍을 사용하며, 소프트웨어로는 게임엔진 Unity3D와 드럼 가상악기 BFD3 등을 주로 사용한다. 발 동작을 인식하기 위한 페달의 경우 설계과제1에서는 아직 제작하지 않았으며, 물리적인 페달에 가속도 센서 등을 부착하여 만들 예정이다.

현재 진행상황은 가상현실 내에서 드럼과 스틱의 3D 이미지를 출력하고, 컨트롤러의 움직임에 따라 스틱이 함께 움직이도록 했으며, 스틱의 타격을 통해 모든 북과 심벌에서 소리를 낼 수 있다. 또한 실제와 비슷한 타격감을 제공하기 위해 북과 심벌이 흔들리는 애니메이션을 구현했으며, 컨트롤러에서는 햅틱 반응을 통해 손으로 타격 진동을 전달한다.

본 설계과제의 연구를 통해 연습용, 체험용 등으로 사용할 수 있는 드럼세트를 가상현실에서 구현함으로써 악기연주라는 취미생활에 대한 접근성을 개선시키며 나아가 더 많은 악기에도 이 기술이 적용될 수 있다.

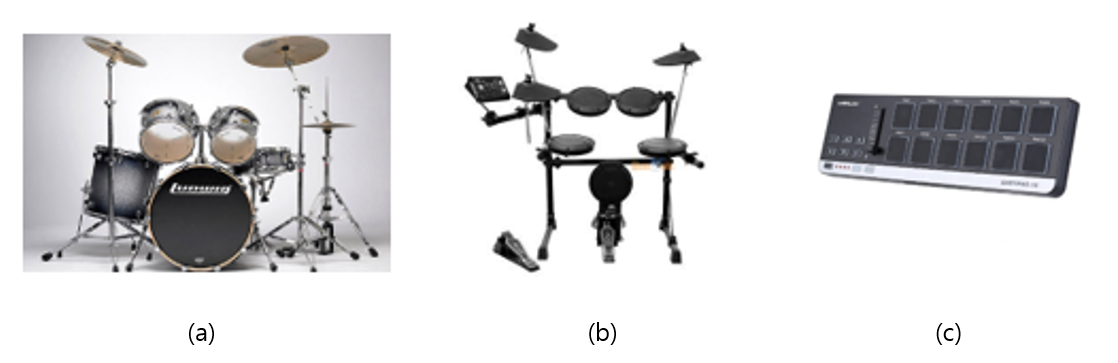
1. **연구 필요성 및 기대효과**
   1. **연구 필요성**

그림 - 기존에 사용중인 여러 종류의 드럼

그림 1-(a)의 일반적인 드럼세트는 부피가 크고 소리도 시끄럽기 때문에 일반 가정집에서 연주하기에 어려움이 있다. 소음 문제를 해결하기 위해 전자 드럼(그림 1-(b))이 등장했지만 악기의 부피 문제를 해결하지는 못했고, 드럼세트의 형태를 포기하고 부피를 대폭 줄인 패드 형태의 전자 드럼(그림 1-(c))도 등장했지만 실제 드럼과의 이질감이 커 연주 시 불편한 점이 많으며, 특히 실제와 다르므로 연습용으로 사용하기에는 적합하지 않다.

따라서 본 설계과제는 이러한 드럼세트의 큰 부피와 소음을 해결하기 위하여 가상현실을 이용한 새로운 형태의 드럼을 고안했다. 가상현실 드럼은 부피가 거의 없으므로 휴대성을 대폭 늘릴 수 있고 VR기기에서 헤드폰으로 소리신호를 바로 출력할 수 있어 소음 문제도 해결할 수 있다.

* 1. **목적 및 기대효과**

본 설계과제의 목적은 기존 악기의 큰 부피와 소음 문제를 해결하여 휴대성을 증대시키고, 악기 연주라는 취미 생활에 대한 접근성을 높여 문화생활 확대에 기여하는 것이다. 이를 통해 초심자의 악기 체험 및 연습용으로 사용가능하면서 동시에 일반적인 연주와 전문적인 연주도 가능한 드럼 세트를 구현하는 것이 목적이다.

완성된 제작품의 보급은 어플리케이션을 마켓에 등록하거나 PC용 소프트웨어를 배포하여 일반 사용자가 쉽게 접근할 수 있도록 한다. 다만 아직 VR기기가 많이 보급되지 않았기 때문에 당장은 어플리케이션 판매를 통한 수익은 기대하기 힘들다.

한편 연구의 활용방안으로 동일한 기술을 다른 악기에 적용해 볼 수 있다. 일부 악기는 부피가 크고 가격도 비싸 접근성이 떨어질 뿐만 아니라, 다세대주택이 많은 한국에서는 집에서 악기연주를 하기 쉽지 않은 상황이다. 하지만 본 연구로 팀파니 등 기타 타악기나 건반악기, 현악기 등도 구현할 수 있다면 이런 문제를 해결하고 일반 가정에서 음악 취미를 즐기기 더 쉬워진다.

또한 다른 적용방향으로는 특정 위치에 사람 또는 물체가 지나갈 때 조건에 따라 다른 소리나 기타 반응들이 나타나게 할 수 있다. 기존에는 현관에 들어서면 형광등이 켜지거나 가게에 손님이 오면 벨이 울리는 정도였지만, 컨트롤러가 특정 위치에 도달하는 것을 카메라가 인식해서 소리를 내는 것처럼 TV 리모컨을 방안에서 휘두르는 것으로 스마트 가구를 조작할 수 있을 것이다. 이 부분에 대해서는 많은 아이디어와 고민이 필요할 것으로 보인다.

* 1. **사업화 가능성**

최근 VR/AR 기술은 가능성을 주목받으며 시장을 확대해 나가고 있으며, 특히 게임을 위해 VR 기기를 구매하는 사람이 증가하고 있는 추세다. 따라서 이러한 VR기기로 즐길 수 있는 게임을 개발하면 수익성을 기대할 수 있다.

이를 위해서는 기존에 출시된 터치스크린을 이용한 연주 게임처럼 다양한 악기 사운드와 연습모드 기능 등을 포함하는 게임을 만들어야 한다. 이러한 게임 소프트웨어와 발로 연주하는 페달을 묶어서 판매할 수 있을 것이다. 다만, 이것이 경쟁력을 가지려면 기존 전자음악 소프트웨어/하드웨어 개발사들의 ‘현실감 있는 사운드를 구현하는 기술력’을 따라잡아야 하기 때문에 많은 노력과 비용이 들 것이다.

우리의 개발 목적은 드럼이라는 악기의 시장이 작을 수밖에 없는 이유를 VR이라는 최신 기술로 해결하는 데에 있다. 따라서, 현재는 시장 규모가 작지만 이 제품을 기존의 어쿠스틱드럼과 전자드럼보다 훨씬 저렴한 가격에 판매한다면 시장 규모를 확대할 수 있다.

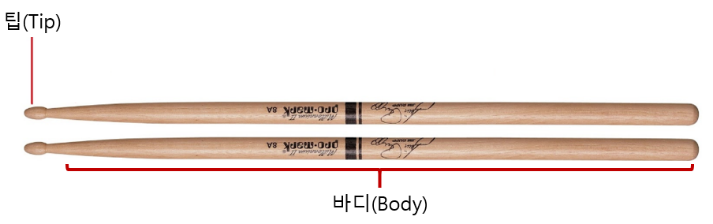
1. **용어 정리**

앞으로의 내용에 대한 이해를 돕기 위해 드럼의 기본적인 용어 및 개념을 정리한다.

* 1. **각 북과 심벌의 명칭**

그림 – 드럼세트의 구성과 명칭

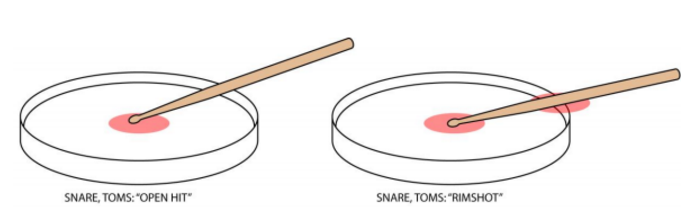
일반적인 드럼세트에서 북과 심벌의 구성과 명칭은 그림 2와 같다. 본 보고서에서는 구별의 편의를 위해 톰톰(Tom-toms) 중 작은 것부터 차례로 하이톰, 미드톰, 로우톰이라고 하겠다.

* 1. **드럼 스틱**

그림

본 설계과제에서는 드럼 스틱을 크게 두 부분으로 나누어 구분하고, 각 부분을 따로 3D 이미지로 모델링했다. 그 이유는 아래에서 설명할 북과 심벌의 여러가지 타격 방법을 구분하기 위해서다.

* 1. **북 타격 종류**

일반적으로 북을 타격하는 방법은 크게 오픈 히트와 림샷으로 나눌 수 있다(그림 4). 오픈 히트란 스틱으로 북을 때리는 가장 기본적인 방법이며, 림샷은 오픈 히트를 하면서 동시에 림(북의 가장자리)을 때려 더 크고 강한 소리를 내는 방법이다. 이외에도 스틱으로 림 만을 치는 방법 등도 있다.

그림

* 1. **심벌 타격 종류**

심벌의 소리를 내는 방법은 스틱의 어떤 부분으로 심벌의 어떤 부분을 치느냐에 따라 크게 세 종류가 있다. 일반적으로 크래시 심벌은 스틱의 바디 부분으로 심벌의 가장자리를 치며, 하이햇과 라이드 심벌은 스틱의 팁 부분으로 심벌의 윗면을 때린다. 이중 라이드 심벌의 경우 특이하게 가운데 볼록하게 올라온 부분을 때리기도 한다. 이 경우 심벌에서 나는 소리가 완전히 다르다.

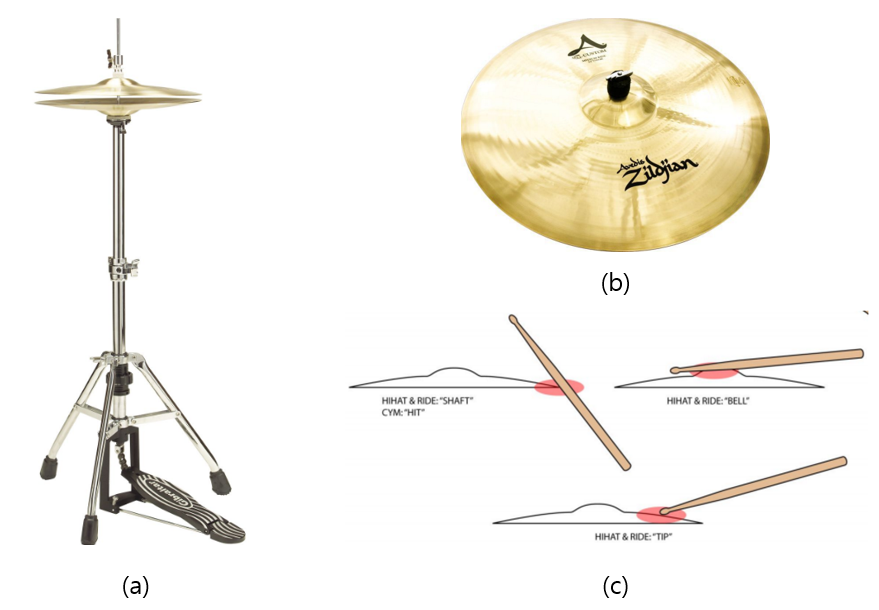
하이햇 심벌은 페달을 통해 두 심벌 사이의 거리를 조절할 수 있으며, 그 정도에 따라 소리가 달라진다. 두 심벌의 거리에 따라 오픈, 세미오픈, 클로즈로 구분할 수 있으며 본 설계과제1에서는 페달 대신 컨트롤러를 이용해 오픈과 클로즈만을 구현했다.

그림 – (a) 하이햇, (b) 라이드 심벌, (c) 심벌 타격의 종류

1. **전체 구조**

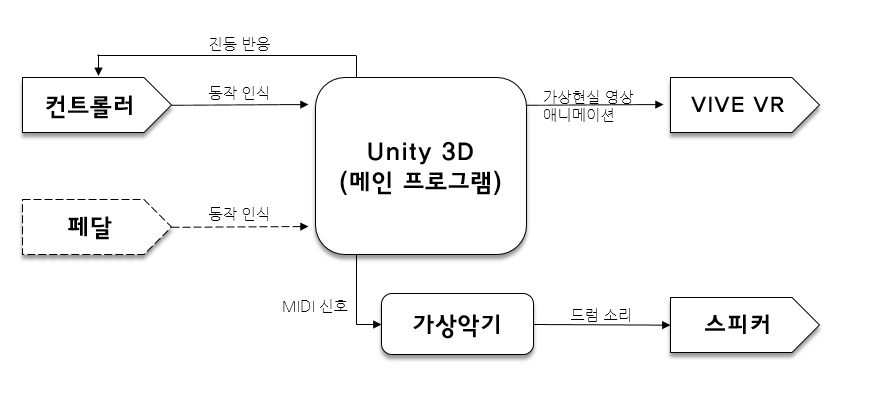


그림 – 전체 프로그램 구조

1. **하드웨어 및 소프트웨어**
   1. **HTC VIVE**

HTC VIVE는 2016년에 공개된 HTC와 Valve Corporation이 공동개발한 가상현실 헤드셋이다. VIVE는 기존 가상현실 기기에는 없는 ‘room scale’ 기능을 도입하여 실제 공간을 베이스 스테이션의 센서를 통해 3D 공간으로 설정할 수 있다.



그림 – HTC VIVE와 컨트롤러, 베이스 스테이션

VIVE는 패키지 구입시 컨트롤러 한 쌍을 기본 제공한다. 베이스 스테이션이 room scale 안에서 이 컨트롤러의 위치를 추적하는데, 인식 속도가 빠르고 정확한 편이다. 이를 통해 양 손의 위치를 항상 추적할 수 있다는 장점과 사용자에게 실제 물체를 잡는 느낌까지 동시에 줄 수 있어 드럼 스틱의 역할을 하기에 가장 적합하다.

* 1. **Unity3D**

Unity 3D는 3D 비디오게임이나 건축 시각화, 실시간 3D 애니메이션 같은 기타 인터랙티브 콘텐츠를 제작하기 위한 통합 저작 도구다.1) 개발언어는 C#, 자바스크립트, Boo를 지원하며 본 설계과제는 C#을 사용한다.

Unity는 Steam VR 플러그인과 연동이 가능하기 때문에 Unity에서 제작한 가상현실 컨텐츠를 VIVE VR을 통해 바로 볼 수 있다는 장점이 있어 본 설계과제를 만드는데 가장 적합한 툴이다. Unity의 또 다른 장점은 엔진 자체에 라이트 맵핑이나 물리 엔진 등의 미들웨어를 탑재하고 있다는 점이다. 이 내장 물리 엔진을 이용해 드럼의 흔들림을 구현했다.

* 1. **Blender**

Unity는 게임 엔진 기술이자 통합개발환경(Integrated Development Environment, IDE)이다. 그렇기 때문에 3D model을 제작하기에는 부족하다. 따라서 외부 프로그램에서 3D model을 제작하여 Unity에서 사용하여야 한다. 이 프로젝트에서는 ‘Blender’이라는 프로그램을 사용하였다.



그림 – Blender 로고

Blender는 Free Software로서 3D computer graphics software이다. Free software임에도 불구하고 좋은 개발환경을 제고해주어 실제 상업용 애니메이션에도 활용된다.

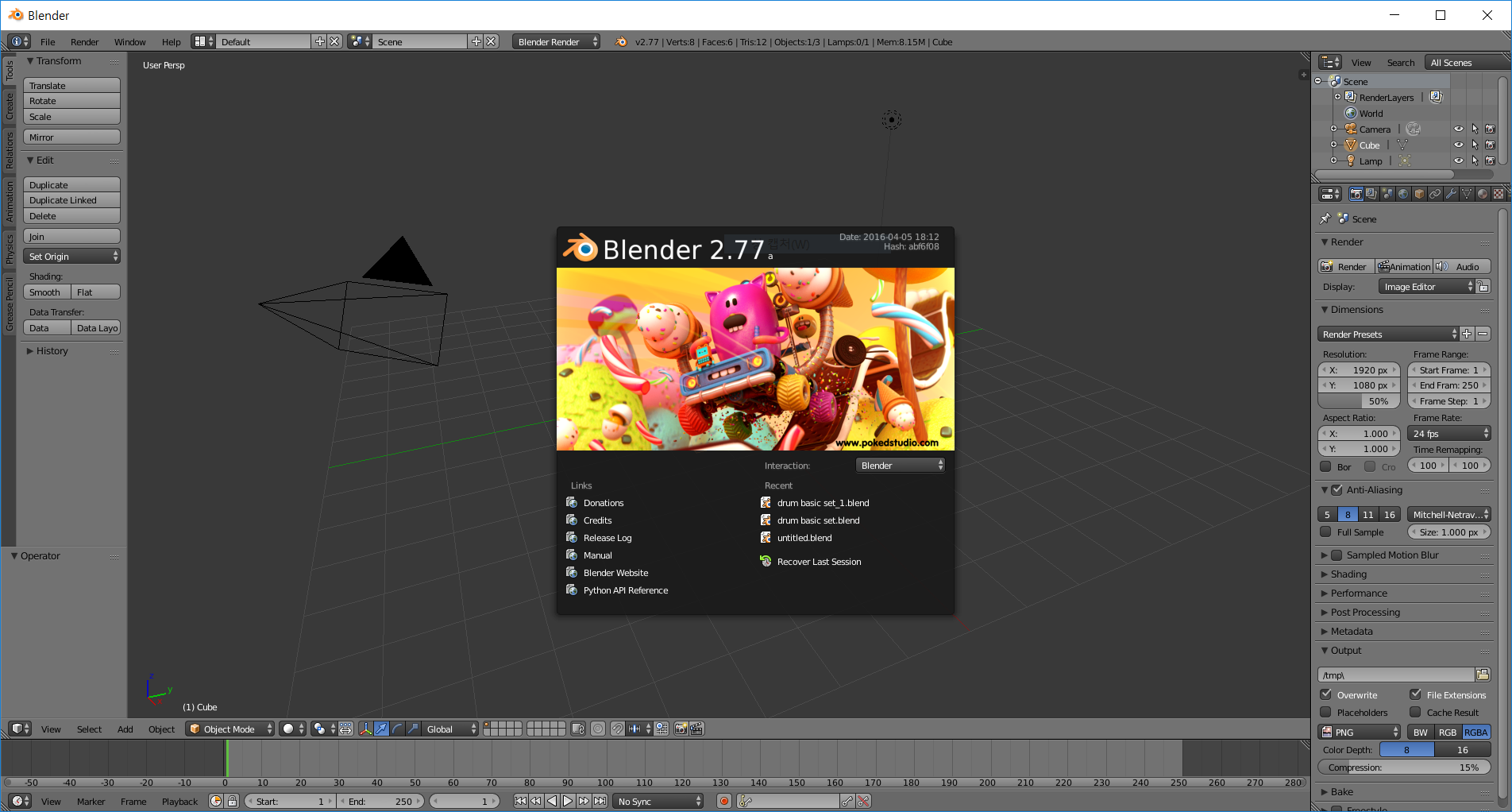


그림 – Blender 실행 화면

우리는 과제 수행을 위해 무료로 공개되어 있는 드럼의 3D 모델을 가져와 프로젝트에서 맞게 blender를 통해 수정하였다.

1. **알고리즘 설명**
   1. **MIDI 신호 규약**

MIDI(musical instrument Digital Interface)는 MIDI Manufacturer Association과 AMEI가 제정하고 공표한, 전자악기의 연주 데이터를 전송하고 공유하기 위한 업계 표준 규격이다. 물리적인 수준의 인터페이스 회로 규격, 연주 데이터를 주고 받을 때 필요한 통신 프로토콜, 파일 형식 등에 대한 표준을 규정하고 있다.2)

MIDI로 송수신하는 데이터 규격은 16진수로 0x00VVNNSC다. 여기서 VV가 velocity, NN이 note, S가 Status, C가 Channel을 각각 의미한다. 이중에서Status가 9이면 Note On 명령, 8이면 Note Off하라는 명령이다. C의 경우 0이 1번 채널을 나타내며, 본 가상드럼은 1번 채널을 쓰고 있다.

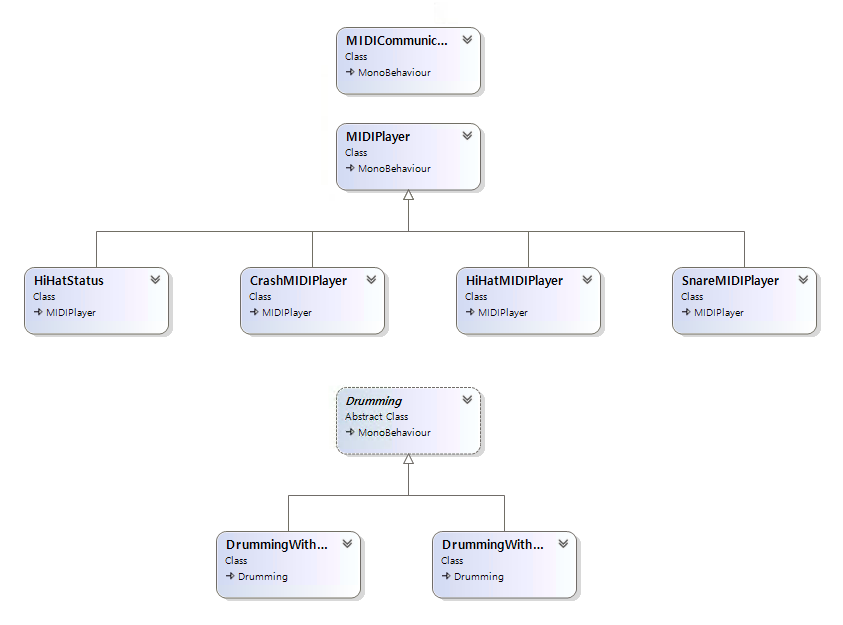
* 1. **Class Diagram**

그림 – Class Diagram

* MIDICommunicator : Drum에 Attach 되어 있으며, 프로그램 시작 시 MIDI 통신 Port를 열고, 종료 시 닫는다. MIDIPlayer 클래스가 호출할 수 있는 sendNote 함수가 있어 MIDI 신호를 실제로 송신하는 역할을 한다.
* MIDIPlayer : 드럼의 각 북과 심벌에 Attach된다. Stick 속도를 MIDI Velocity 값이나 Haptic 반응 세기로 변환해주는 함수를 가지고 있다. 또한 Drumming 클래스가 호출할 수 있는 play 함수가 있어 MIDICommunicator에 재생할 note 정보를 보낸다. Stick의 tip과 body 중 먼저 충돌한 것만 처리하도록 한다.
  + HiHatStatus : 플레이어가 Hi-Hat을 여닫는 동작을 할 때 해당 노트를 재생한다. MIDIPlayer의 자식클래스이긴 하지만 Stick(에 Attach되는 Drumming 클래스)과 상호작용하지는 않는다.
  + CrashMIDIPlayer : Crash 심벌에 특화된 기능을 가지고 있다. DrummingWithBody와 상호작용하여 스틱의 body로 연주할 수 있게 한다. Stick 속도를 MIDI Velocity 값으로 변환할 때 사용하는 상수를 심벌에 적절한 값으로 가지고 있다.
  + HiHatMIDIPlayer : Hi-Hat에 특화된 기능을 가지고 있다. 충돌 시 HiHatStatus를 통해 Hi-Hat이 열려 있는지 닫혀 있는지 확인하고 상태에 맞는 note를 MIDICommunicator에 보낸다. Stick 속도를 MIDI Velocity 값으로 변환할 때 사용하는 상수를 심벌에 적절한 값으로 가지고 있다.
  + SnareMIDIPlayer : Snare 에 특화된 기능을 가지고 있다. DrummingWithTip, DrummingWithBody와 모두 상호작용하여 스틱의 body와 tip으로 동시에 치는 RimShot 이 가능하도록 한다.
* Drumming : 추상클래스이다. 자식클래스가 Stick에 Attach 된다. 북이나 심벌과 충돌했을 때 스틱의 속도를 계산하고 MIDIPlayer의 함수들을 호출하여 MIDIPlayer가 가지고 있는 note를 재생하도록 한다. 스틱의 속도를 계산하는 함수를 자식 클래스가 구현하여야 한다.
  + DrummingWithTip : 스틱의 끝부분(tip)에 Attach된다. 끝부분의 속도를 매 프레임마다 계산해놓고, 충돌 시에 는 최근 20개의 속도 중 최댓값을 사용한다.
  + DrummingWithBody : 스틱의 몸체(body)에 Attach된다. 충돌하는 순간 충돌 지점의 속도를 계산한다.
  1. **알고리즘 설명**

아래 ①~⑨의 모든 스크립트에서 공통적으로 사용되는 함수는 Start(), Update(), OnApplicationQuit(), Mathf.Lerp(a,b,t)가 있다.

Start() 함수는 프로그램이 시작할 때 호출되는 함수이며, Update() 함수는 매 프레임마다 호출되는 함수이다. 프로그램에 사용된 모든 스크립트의 Update 함수가 끝까지 처리되어야 다음 프레임으로 넘어간다. 그리고 OnApplicationQuit() 함수는 프로그램이 종료될 때 호출되는 함수이며, 마지막으로 Mathf.Lerp(a,b,t) 함수는 a와 b 사이를 t:1-t 로 내분하는 값을 리턴하는 함수이다.

1. **MIDI Communicator**

|  |
| --- |
| //윈도우 내장 winmm.dll 파일에서 MIDI 관련된 함수 Import  private int handleMIDI;  void Start() {  //통신 Port Open  }  //MIDIPlayer 객체들이 호출하는 함수  public void sendNote(int note, int velocity) {  //새로운 Thread를 만들어 sendNoteOnOff 함수를 처리하도록 함  new Thread(() => sendNoteOnOff(note, velocity)).Start();  }  private void sendNoteOnOff(int note, int velocity) {  //드럼은 Note On 명령 시 타격음이 끝까지 재생되므로  //Note Off 명령은 적절한 시간이 지난 후 해주면 된다.  //“note를 velocity의 세기로 Note On”메시지를 보냄  //Thread를 10ms 동안 일시정지  //“note를 velocity의 세기로 Note Off”메시지를 보냄  }  void OnApplicationQuit() {  //통신 Port Close  } |

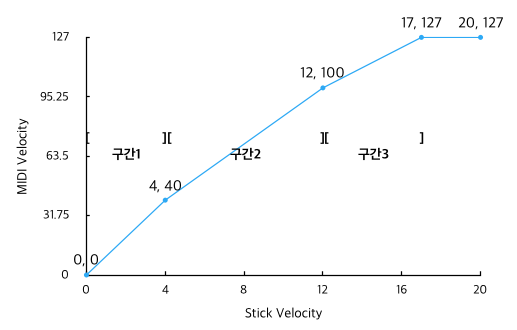
1. **MIDI Player**

그림 – MIDI velocity

|  |
| --- |
| public String Note; //어떤 note를 재생할 지 가지고 있다.  //MIDI Velocity를 Haptic 진동 세기로 변환  public Vector2 hapticInformation(int midiVel) {  float hapticTime, hapticStrength;  //구간1일 때  hapticTime = 0.05f; //0.05초 진동  //구간2일 때  hapticTime = 0.10f; //0.10초 진동  //구간3일 때  hapticTime = 0.15f; //0.15초 진동    //진동 세기를 MIDI Velocity에 따라 선형적으로 설정  hapticStrength = Mathf.Lerp(0.1f, 1.0f, midiVel / max\_midi);  //(hapticTime, hapticStrength) 를 리턴  }  //Stick Velocity를 MIDI Velocity로 변환  public virtual int convertVelToMIDIVel(float velocity) {  //위 그래프와 같이 구간1,2,3에 따라 MIDI Velocity 값 설정, 리턴  }  //Drumming이 호출하는 함수  public virtual void play(string stick, int midiVel) {  if(Stick body가 이미 호출한 후 아직 충돌이 끝나지 않았는데 tip이 호출했거나  반대로 tip이 이미 호출했는데 body가 이미 호출한 경우만 아니면)  play(midiVel); //아래 play 함수 호출  }  protected void play(int midiVel) {  //가지고 있던 note 정보와 인자로 전달받은 velocity로  //MIDI Communicator의 sendNote 호출  midicomm.sendNote(noteNumber, midiVel);  } |

MIDI Velocity는 0부터 127까지의 값을 가지는데, 본 프로젝트에서 사용한 가상악기의 경우 0~40, 40~100, 100~127의 세 구간에서 각각 서로 다른 특징을 가진 소리가 재생되는 것을 확인했다. 그 중 무난한 소리가 나는 40~100 구간을 측정하기 위해 MIDI Velocity가 40, 50, 60, 70, 80, 90, 100 인 가상악기 스네어 소리를 드러머에게 들려준 후, 스틱과 컨트롤러를 동시에 쥔 채로 들려준 가상악기 소리와 비슷한 소리가 나도록 실제 스네어를 연주하게 했다. 그 결과 컨트롤러로 측정된 스틱의 피크 속도가 위 그래프(구간 2)와 같이 선형적으로 나타났다. 또한, 드러머가 최대한 세게 연주할 때의 스틱 속도가 17로 나타났다. 이에 따라 구간1과 구간3은 자동으로 결정되었다. 구간1과 구간3도 선형으로 매핑한 이유는 소리나 주법 자체가 구간2와 크게 다르지는 않고, 곡선으로 매핑하지 않아도 충분히 실제 연주에 근사할 것이라 생각했기 때문이다.

1. **Hi Hat Status**

|  |
| --- |
| //열고 닫을 때 나는 소리의 note를 가지고 있다.  public string OpeningNote;  public string ClosingNote;  bool \_Closed; //닫힌 상태면 true, 열린 상태면 false  void Update () {  if(사용자가 닫는 동작을 하면) {  \_Closed = true;  //ClosingNote play  }  else if(사용자가 여는 동작을 하면) {  \_Closed = false;  //OpeningNote play  }  } |

1. **Crash MIDI Player**

|  |
| --- |
| //생성자에서 구간1,2,3 상수 설정  //Drumming이 호출하는 play함수 오버라이드  public override void play(string stick, int velocity) {  //Stick body로 쳤을 때에만 원래의 play함수 처리  } |

1. **Hi Hat MIDI Player**

|  |
| --- |
| //생성자에서 구간1,2,3 상수 설정  //Drumming이 호출하는 play함수 오버라이드  public override void play(string stick, int velocity) {  //Stick body로 쳤을 때  //HiHatStatus의 \_Closed에 따라  //열려있을 때의 note와 닫혀있을 때의 note를 play  } |

1. **Snare MIDI Player**

|  |
| --- |
| //Drumming이 호출하는 play함수 오버라이드  public override void play(string stick, int velocity) {  //이미 body가 충돌중인 상태였는데 tip이 호출한 것이라면 림샷 note play  //그냥 tip이 호출한 것이라면 오픈히트 note play  } |

1. **Drumming**

|  |
| --- |
| public abstract float getVelocity(Collision col); //자식클래스에서 구현할 함수  //충돌하여 겹치기 시작할 때 호출되는 함수  void OnCollisionEnter(Collision col) {  //충돌한 object의 MIDIPlayer에 충돌했다고 알림  //(이 클래스가 attach되어있는 object가 body인지 tip인지도 알림)  //충돌한 지점이 그 object의 표면일 때에만 아래를 처리    //getVelocity 함수로 Stick 속도를 얻은 후  //MIDI Player의 함수로 MIDI Velocity와 Haptic 반응 세기를 얻음  int midiVel = player.convertVelToMIDIVel(getVelocity(col));  Vector2 hapticInformation = player.hapticInformation(midiVel);  //얻은 값으로 MIDI Player의 play함수 호출 및 Haptic 진동 시작  StartHapticVibration(controller, hapticInformation.x, hapticInformation.y);  player.play(name, midiVel);  }  //충돌이 끝나 따로 떨어지게 되었을 때 호출되는 함수  void OnCollisionExit(Collision col) {  //충돌한 object의 MIDIPlayer에 충돌이 끝났다고 알림  } |

1. **DrummingWithTip**

|  |
| --- |
| void Update() {  //현재 position과 이전 프레임의 position 차이를  //프레임간 시간으로 나누어 velocity 계산  //최근 20개의 velocity를 보관  }  public override float getVelocity(Collision col) {  //보관하고 있는 velocity 들의 북(혹은 심벌) 표에 대한 수직방향 성분을 계산  //그 성분들 중 가장 최근의 peak를 구하여 return  } |

1. **DrummingWithBody**

|  |
| --- |
| public override float getVelocity(Collision col) {  //충돌 지점의 velocity를 얻고 북(혹은 심벌) 표면에 대한 수직방향 성분을 계산  } |

1. **애니메이션**

가상현실에서 드럼 타격 시 연주자가 현실과의 이질감을 느끼지 않게 하기 위해서, 드럼이 흔들리는 모습을 구현해야한다. 이를 구현하는 방법으로는 물리 엔진(physics engine)과 Animation이 있다. 물리 엔진은 타격 시 충돌 크기에 따라 흔들림이 다른 경우에 사용하며, 이와 반대로 Animation은 흔들림이 적어 타격 시 충돌 크기와 거의 무관한 경우에 사용한다. Animation은 record된 motion을 재생시키는 것이기 때문에 컴퓨터의 연산 과정이 물리 엔진보다 적어 latency를 최소화 시킬 수 있다.

* 1. **물리 엔진(physics engine)**

충돌에 Unity 물리 엔진을 사용하기 위해서는 먼저, ‘Rigidbody’와 ‘Collider’를 설정해 주어야한다.

* Rigidbody: Object가 물리 법칙 안에서 동작하게 된다. 따라서 힘과 토크를 받아 현실적으로 오브젝트를 움직일 수 있게 된다.
* Collider: Object간의 충돌 감지를 위해 object를 감싸는 경계를 설정하는 것이다. Object의 모양이나 충돌 특징에 따라 Box collider, Sphere collider, Mesh collider 등에서 한 가지를 선택하여 사용한다. 복잡한 model인 경우 part별로 나누어 적용하는 경우도 있다.

이후 실제 드럼과 같이 각 part별로 연결을 해주어야 한다. 이 연결을 Unity에서는 ‘Joint’라 부른다. Joint에는 여러 종류가 있으면 이번 프로젝트에서는 Fixed Joint와 Hinge Joint를 주로 사용하였다.

* Fixed Joint: 명칭 그대로 두 Object를 고정하는 Joint이다. Tom과 Kick을 연결하거나 세분화된 Stand를 연결하는데 사용한다. 한 rigidbody에 fixed joint를 add하고 connected body에 다른 rigidbody를 선택하여 두 rigidbodies를 연결할 수 있다. Break Force와 Break Torque를 설정하여 joint가 해제되는 조건을 부여할 수 있으나 이번 프로젝트에서는 사용하지 않았다.

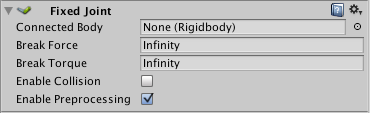


그림 – Fixed Joint

* Hinge Joint: Hinge Joint는 여닫이 문의 경첩 부분과 같다. 그렇기 때문에 보통 문에 적합하지만 설정을 통해 사슬이나 진자와 같은 model에도 사용할 수 있다. 한 rigidbody에 hinge joint를 add하고 connected body에 다른 rigidbody를 선택하여 두 rigidbodies를 연결할 수 있다. Joint가 움직이는 방향을 Aixs component를 통해 설정할 수 있다. 또한 추가적인 설정을 통해 joint에 spring과 같은 탄성을 가진 운동을 하도록 설정할 수 있고, Limits를 설정하여 joint의 운동 범위를 제한할 수 있다. 이번 프로젝트에서는 실제 드럼에서의 탄성과 연결부분에 의해 각도가 제한되는 것을 확인하고 Hinge Joint의 Spring과 Limits를 사용하였다. Crash의 흔들림을 구현하기 위해 Hinge Joint를 사용하였다.



그림 – Hinge Joint

* 1. **Animation**

Animation은 Object를 선택 한 후에 rotation 값을 조절하여 설정한다. 이후 Collision이 되었을 때 animation이 재생 되도록 하는 script를 attach한다. Animation이 사용된 object는 High Tom과 Low Tom이다.

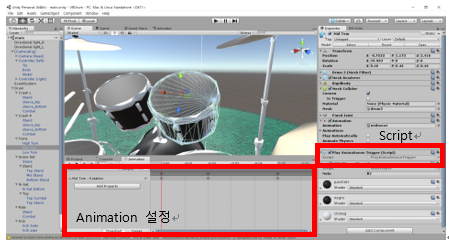


그림 – Animation 설정

|  |
| --- |
| private Animation move;  void Start () {  move = GetComponent<Animation>(); // Object의 Animation을 받아온다.  }  void OnCollisionEnter(Collision col) {  move.Play(); // Collision이 일어나면 Animation을 play한다.  } |

1. **주요 오브젝트**
   1. **Crash L**

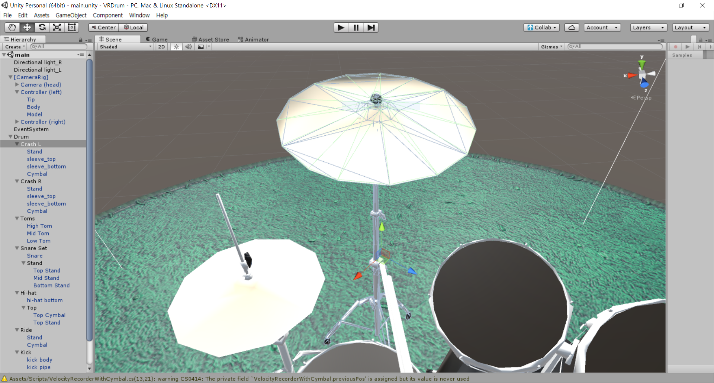


그림 – Crash L

|  |  |
| --- | --- |
| 하위 object | Stand, Cymbal, sleeve\_top, sleeve\_bottom |
| Attached Scripts | Crash MIDI Player |

Mesh Collider를 사용하여 충돌 판정을 하였으며, Hinge Joint를 사용하여 흔들림을 구현하였다.

* 1. **Crash R**

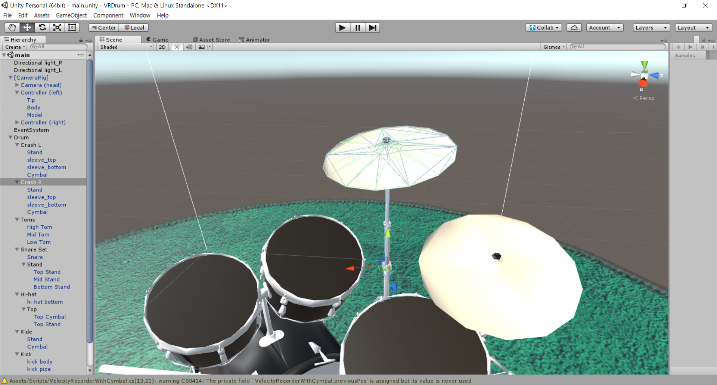


그림 – Crash R

|  |  |
| --- | --- |
| 하위 object | Stand, Cymbal sleeve\_top, sleeve\_bottom |
| Attached Scripts | Crash MIDI Player |

Mesh Collider를 사용하여 충돌 판정을 하였으며, Hinge Joint를 사용하여 흔들림을 구현하였다.

* 1. **High Tom**

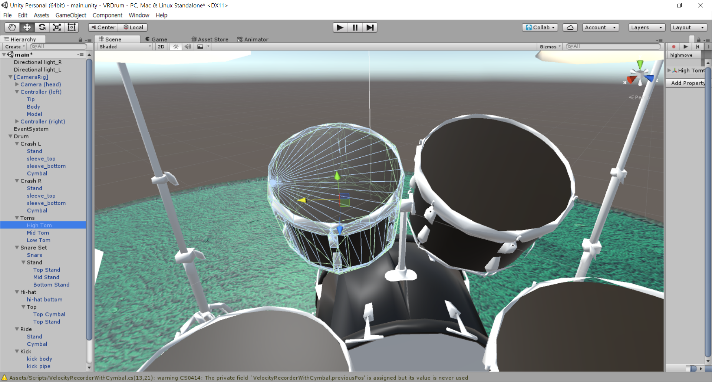


그림 – High Tom

|  |  |
| --- | --- |
| 하위 object | 없음 |
| Attached Scripts | Play Animation Trigger, MIDI Player |

Mesh Collider를 사용하여 충돌 판정을 하였으며, Fixed Joint를 사용하여 Kick과 연결하였다. 또한 Animation을 사용하여 흔들림을 구현하였다.

* 1. **Mid Tom**

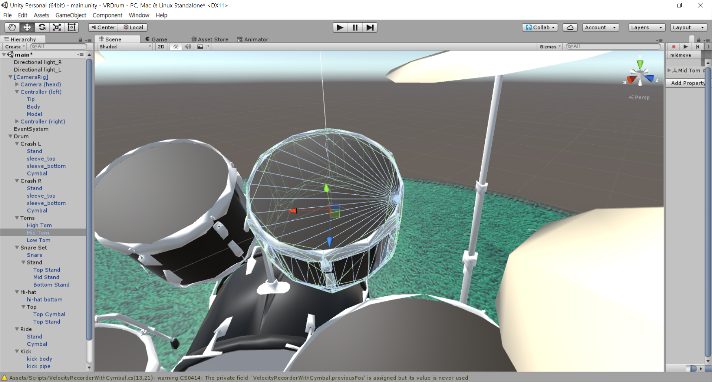


그림 – Mid Tom

|  |  |
| --- | --- |
| 하위 object | 없음 |
| Attached Scripts | Play Animation Trigger, MIDI Player |

Mesh Collider를 사용하여 충돌 판정을 하였으며, Fixed Joint를 사용하여 Kick과 연결하였다. 또한 Animation을 사용하여 흔들림을 구현하였다.

* 1. **Low Tom**

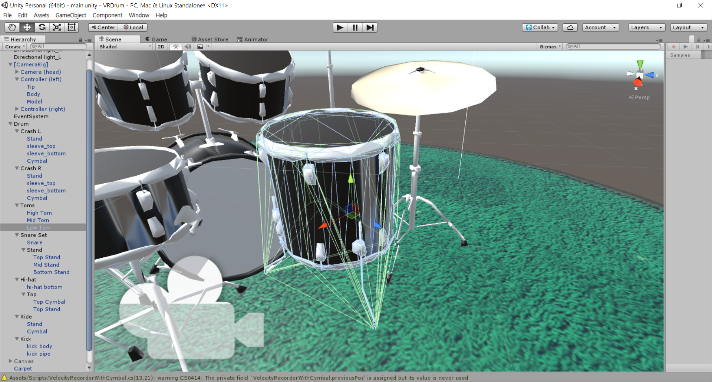


그림 – Low Tom

|  |  |
| --- | --- |
| 하위 object | 없음 |
| Attached Scripts | MIDI Player |

Mesh Collider를 사용하여 충돌 판정을 하였다.

* 1. **Hi-Hat**

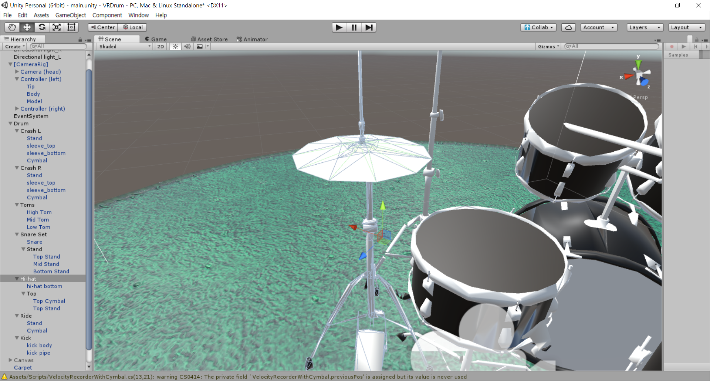


그림 – Hi-Hat

|  |  |
| --- | --- |
| 하위 object | Hi-hat bottom, Top(Top Cymbal, Top Stand) |
| Attached Scripts | Hi Hat MIDI Player |

Model이 복잡하여 전체를 하나의 Collider로 설정할 수 없으며, 후에 페달을 구현할 경우를 생각하여 많은 part로 나누었다. Top Cymbal에 Mesh Collider를 사용하여 충돌 판정을 하였다.

* 1. **Snare Set**

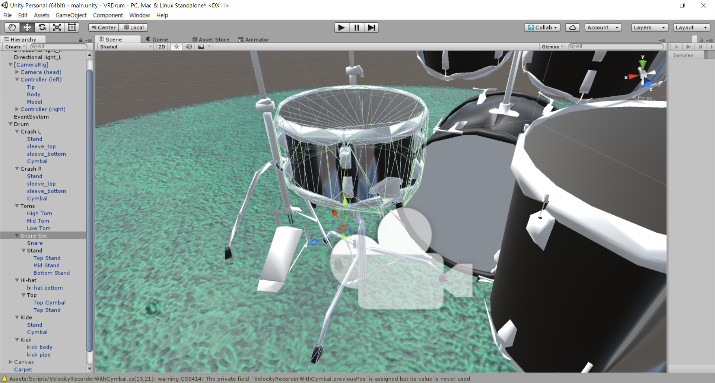


그림 – Snare Set

|  |  |
| --- | --- |
| 하위 object | Snare, Stand(Top stand, Mid Stand, Bottom Stand) |
| Attached Scripts | Snare MIDI Player |

Model이 복잡하여 전체를 하나의 Collider로 설정할 수 없으며, snare의 높이 조절을 위해 Stand를 여러 단계로 분리하였다.

* 1. **Kick**

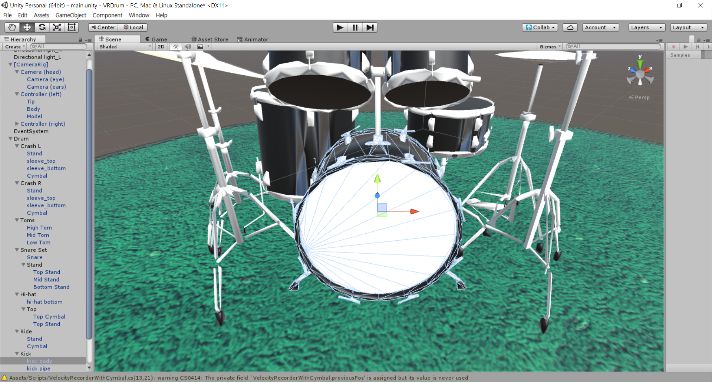


그림 - Kick

|  |  |
| --- | --- |
| 하위 object | Kick body, kick pipe |
| Attached Scripts | 없음 |

Kick은 페달이 구현되어야만이 테스트해볼 수 있기 때문에 아직 model만 Import 해놓은 상태이다.

* 1. **Ride**

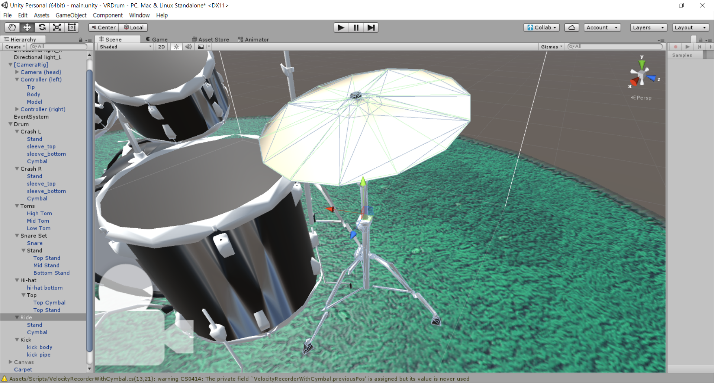


그림 - Ride

|  |  |
| --- | --- |
| 하위 object | Stand, Cymbal |
| Attached Scripts | Crash MIDI Player |

Mesh Collider를 사용하여 충돌 판정을 하였으며, Hinge Joint를 사용하여 흔들림을 구현하였다. 라이드 심벌에 특화된 MIDI Player는 아직 만들지 않았다.

* 1. **Controller(Stick)**

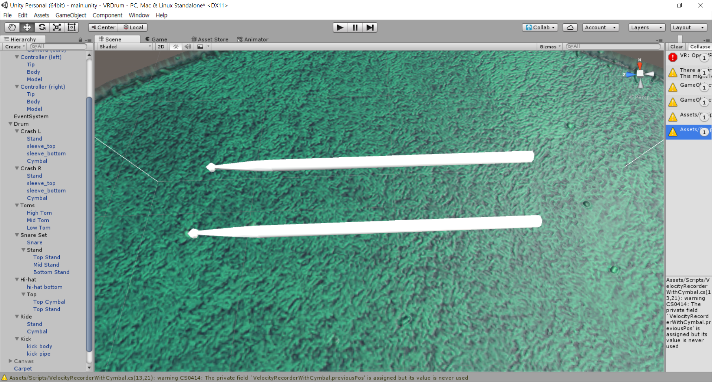


그림 - Stick

|  |  |
| --- | --- |
| 하위 object | Tip, Body, Model |
| Attached Scripts | Drumming With Tip, Velocity Recorder With Cymbal |

실제 드럼을 치는 경우 Stick의 body보다는 tip으로 타격을 한다. 따라서 Tip과 Body를 분리하여 좀 더 정확한 충돌 판정이 가능하도록 하였다. Tip은 Sphere Collider, Body는 Box Collider를 사용하였다. Mesh Collider는 또다른 Mesh Collider와 충돌 판정이 불가능 하기 때문에 Body는 Box Collider를 사용하였다.

1. **진행상황 및 앞으로의 계획**

현재 대부분 오브젝트의 흔들림을 구현했지만 타격 시 북의 통 자체가 움직이지는 않는 스네어와 로우톰에 대해서는 타격 애니메이션을 구현하지 않았다. 드럼의 현실감을 살리는 것도 중요하지만 이 부분은 가상현실의 부족한 타격감을 살리기 위해서 약간의 과장된 애니메이션을 추가할 예정이다.

한편 라이드 심벌에 특화된 MIDIPlayer가 없다. Stick의 body로 연주하는지 tip으로 연주하는지, 가장자리를 타격하는지 중심을 타격하는지에 따라 다른 소리가 나도록 해야 한다. 이는 Hi-Hat MIDI Player에도 부족한 부분이다. 또한 심벌들의 경우 주로 스틱의 body부분으로 연주하는 경우가 많은데, 현재 body로 타격하는 알고리즘이 타격 지점의 속도를 정확하게 계산하지 못하고 body의 무게중심의 속도로 근사하고 있다. 이를 보완해야한다.

마지막으로 현실에선 불가능하지만 가상현실에서는 스틱이 심벌과 북을 통과할 수 있어 발생하는 부작용들이 있다. 물리 엔진을 사용하여 흔들림을 구현한 심벌들이 부자연스럽게 움직이는 것이 그 중 하나다. 이러한 부작용들에 대한 해결책이 필요하다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ~1/13 | ~1/27 | ~2/17 | ~3/17 | ~4/14 | ~5/12 |
| 아두이노 페달 제작 |  |  |  |  |  |  |
| VR에 페달 적용 |  |  |  |  |  |  |
| 메트로놈 기능 |  |  |  |  |  |  |
| MIDI Recording |  |  |  |  |  |  |
| BGM 설정 기능 |  |  |  |  |  |  |
| 마무리 및 시연 |  |  |  |  |  |  |

그림 - Timeline

앞으로 설계과제2에서 진행할 내용은 페달의 제작, 만든 페달과 VR의 연동, 그리고 기타 추가기능 구현이다. 겨울방학 중에는 페달에 가속도 센서를 부착하고 아두이노 우노를 통해 신호를 받아들이도록 만든다. 2월 17일까지 만든 페달을 가상현실 속에서 kick 및 Hi-Hat과 연동하여 소리재생을 완료한다.

이후 봄학기가 시작되면 연주자의 편의를 위한 여러 추가기능을 도입한다. 메트로놈 기능은 드럼 연주자들이 박자에 맞춰 연주할 수 있도록 하기 위한 기능이며, 배경음악 설정과 녹음 기능을 추가하여 음악을 들으며 연주함과 동시에 자신의 연주를 녹음할 수 있도록 한다.

5월에는 마무리 작업을 통해 프로젝트를 완성하고, 교내 POSEF 행사에서 부스 내 시연을 할 예정이다.

1. **참고자료 및 사진출처**

-참고자료

1) Unity3D : [https://ko.wikipedia.org/wiki/%EC%9C%A0%EB%8B%88%ED%8B%B0  
\_(%EA%B2%8C%EC%9E%84\_%EC%97%94%EC%A7%84)](https://ko.wikipedia.org/wiki/%EC%9C%A0%EB%8B%88%ED%8B%B0_(%EA%B2%8C%EC%9E%84_%EC%97%94%EC%A7%84))

2) MIDI : <https://namu.wiki/w/MIDI>

-사진 출처

그림 1-(b) : <https://namu.wiki/w/>

그림 1-(c) : [https://www.linio.com.mx/p/worlde-easypad-12-mini-usb-porta-til-12  
-drum-pad-midi-controlador-ylc4qd](https://www.linio.com.mx/p/worlde-easypad-12-mini-usb-porta-til-12-drum-pad-midi-controlador-ylc4qd)

그림 2 : <https://creativedrumming.wordpress.com/2013/04/07/knowing-your-drum-set-part-i/>

그림 3 : <http://audioboxed.blogspot.com/2016/04/bagian-bagian-stik-drum.html>

그림 4, 그림 5-(c) : :<http://blog.cakewalk.com/ad2-sounds-real/>

그림 5-(a) : <http://acehmusician.org/sejarah-lahirnya-hi-hat/>

그림 5-(b) : <http://www.ebay.com/gds/Ride-Cymbal-Buying-Guide-/10000000177330374/g.html>

그림 7 : <https://namu.wiki/w/HTC%20Vive>