Traditional polish frame drum and bass VR simulator

Łukasz Nizik

1 Introduction

Goal of the project is to create real-time virtual reality (VR) simulation for interactive presentation of traditional polish folk instruments. The instruments that I selected are common in regions of Central Poland. This project researches polish frame drum and bass (basetla).

2 Methods

2.1 Technology

Targeted platform is Oculus Quest 2 VR headset with quest link connection.

I used Unreal Engine 4 as a base for creating the simulation. I created the early prototypes using Unity 5 engine, however I could reach my expectations for lowering latency between input and output due to technical limitations.

2.2 Audio recordings

Real instruments recordings specially prepared for this project was important for getting desired quality. For this purpose we recorded samples of selected sounds of the instruments.

For the standard polish frame drum playing technique we can distinguish two most popular types of sound - hitting a skin with a stick and hitting an edge of the drum with a bare hand. They are both simulated in the application that I created. The drum is held by the player by the lower part of the edge.

I divided the drum into zones. Every zone uses different pool of samples. It is done to simulate changing of the sound pitch, timbre and intensity of cymbals depending on the place hit with a stick. Hitting the middle of the drum in a real instrument emits lower and richer sound with less cymbals excitation. Figure [1] shows the division of the drum that I used. Zone R_b is the place where the drum is held with player's hand, and R_t is the upper edge of the drum. For simplification of studio operations zones R_m and G_m use the same recording as R_t. I used 3 types of articulation for every zone from 3 recordings accordingly: piano, mezzo-forte, forte. The impact force influences loudness and other qualities of the sound. It works similarly with strokes with a hand against the edge.

The bass is recorded with 1 type of articulation. It is tuned to A440 Hz in standard cello tuning (without C string) A, D, G. The strings are recorded in pairs (fifths).

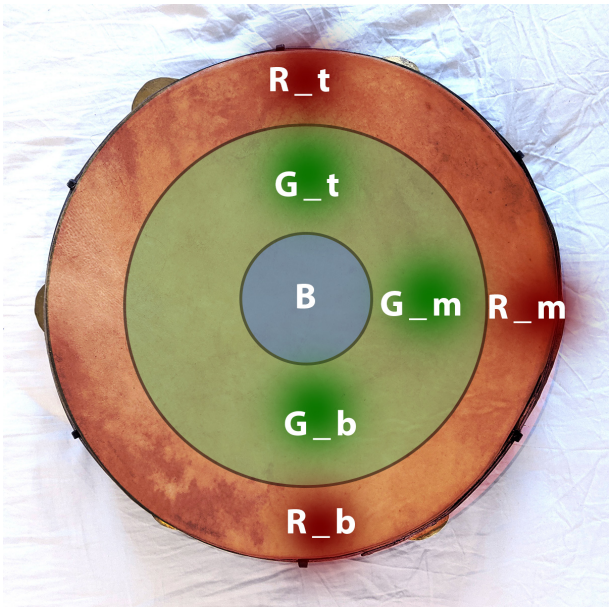

Figure 1: Drum audio recording zones

An important element to achieve good looping results was to produce possibly most stable long sound.

2.3 Implementation - bass

The bass sound is a looped recording that is played continuously and its loudness is proportional to the current speed of bow along the strings tangent vector. For smooth looping I use linear cross-fade between end and start of the sound. The recording was initially cut so that it contains the main part of the sound with stable loudness without attack and release parts. The looping is implemented by overlapping start with an end of the sound and blending between them with linear interpolation. It is important to find a correct overlap length that is synchronized in phase otherwise the overlapping part might differ in loudness to an extent when its easily audible. Another way to overcome it is by using a different cross-fade method. The blending between the overlapping clips that I used is described by an equation [1].

$$f(x) = a \cdot \alpha + b \cdot (1 - \alpha) \tag{1}$$

a - value of the sample of the first recording
b - value of the sample of the second recording

$$\alpha = \frac{(i_a - O_{start})}{N_o} \tag{2}$$

N_o - count of overlapping samples
i_a - current sample index for the first recording (a)
O_{start} - overlapping offset index

The implementation in Unreal Engine 4 uses a C++ class `ULoopPlayer` that derives from engine's `USynthComponent`. The main method `int32 OnGenerateAudio(float* OutAudio, int32 NumS` returns a pointer via argument `OutAudio`. It points to a samples data that will be submitted to a sound card. The number of the required samples is equal to the audio buffer size multiplied by the number of audio channels that are used. The same method of playing audio is used by the drum simulation as well.

To produce sound with a real instrument it is needed to press the bow hair against the strings and rub them. The player feels the pressing force, friction and the angle between strings and the bow. Controllers that are used with Oculus Quest 2 does not allow to reliably send such force feedback which makes it difficult to know whether the bow is correctly placed on the strings. To overcome this issue I made a calculation that automatically assures the correct position of the bow so the hairs always remain in contact with the strings. In addition I made a constraint that stops the bow from being dragged to far. I placed $StringContactPoint$ object that helps me defining the correct position P_S and orientation in relation to the strings (figure [**??**]). I use point P_B as a reference on the bow. It denotes a point where the bow is held with a hand (figure [3]). The axes colors are according to UE4 standard:

red - X axis $Forward$
green - Y axis $Right$
blue - Z axis Up

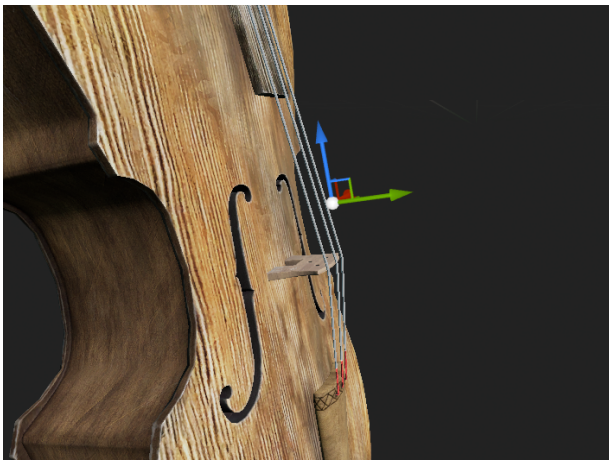

Figure 2: Contact point between bow and string

Equations [3 4 5 6] calculate new rotation matrix of the bow based on its location P_B in relation to P_S.

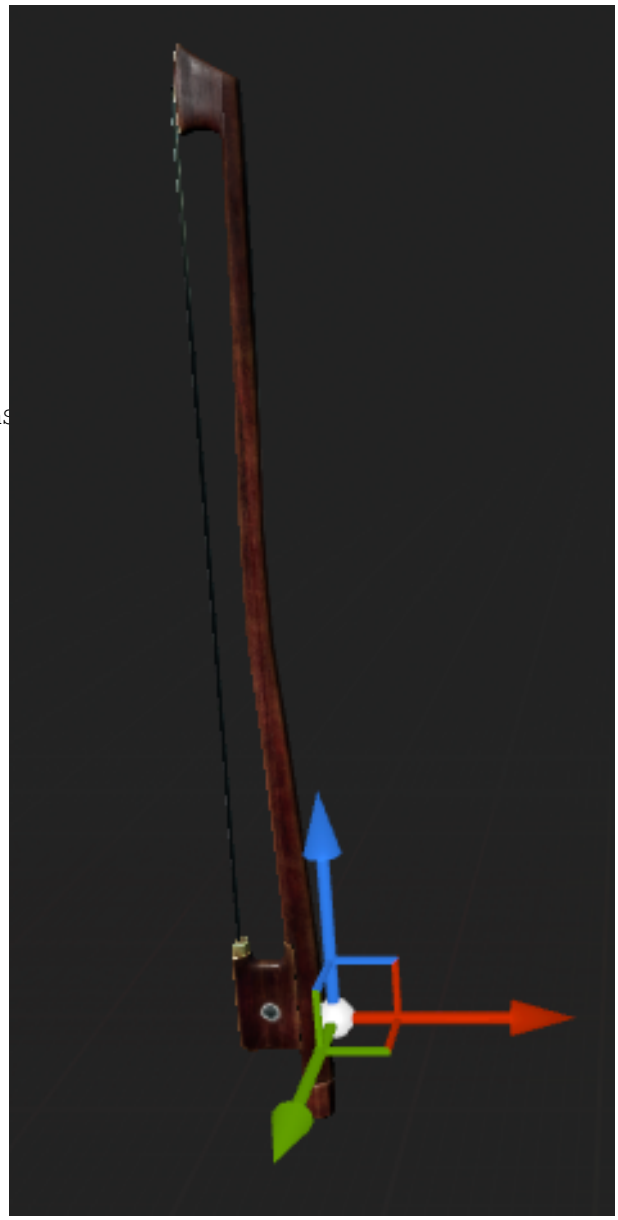

Figure 3: Bow hand position

$$\hat{Z_S}' = \hat{V_{BS}} = \frac{P_B - P_S}{|P_B P_S|} \tag{3}$$

$$\hat{Y_S}' = \hat{Z_S}' \times \hat{Y_S} \tag{4}$$

Y_S - axis normal vector Y for local coordinate space in the contact point P_S is the $Right$ vector from transformation matrix in Unreal Engine 4.

$$\hat{X_S}' = \hat{Y_S}' \times \hat{Z_S}' \tag{5}$$

$$T'_S = [\hat{X'_S} \hat{Y'_S} \hat{Z'_S}] \tag{6}$$

The main difference between implementation between double and triple stringed bass is in the possibility of changing string between two pairs. I prepared the algorithm for selecting which pair of strings should be

used. The angle between handle forward vector and the $StringContactPoint$'s forward vector is the main factor dictating which string sample is selected. The forward vectors' orientation can be seen at figures 3 and **??** as red arrows. The dot product of these is positive if the user plays higher pair of strings and the opposite means the lower pair.

2.4 Implementation - drum

EACH sample is multiplied by the volume factor that depends on linear velocity of the drum stick at the end that hits a drum skin (speed factor S). The equation to calculate it takes to consideration only the component of the velocity vector that is parallel to the normal of the drum's surface (equations [7], [8]) and with opposite direction to the normal.

$$\vec{V} = \frac{\overrightarrow{\delta P}}{\delta t} \tag{7}$$

V - drum stick velocity vector
$\overrightarrow{\delta P}$ - difference between stick's position between two consecutive simulation steps
$\overrightarrow{\delta t}$ - current simulation timestep

$$S = (\vec{V} \cdot \overrightarrow{D_n}) \cdot C \tag{8}$$

S - drum stick speed factor
$\overrightarrow{D_n}$ - drum skin surface normal
C - speed normalization coefficient

Negative value of product $\vec{V} \cdot \overrightarrow{D_n}$ denotes that the back side of the drum was hit. This case does not trigger audio playback.

A sample from a drum recording i multiplied by factor S that is mapped using a curve that was found empirically (figure [4]) $g(s)$ as in equation [9].

$$y' = y \cdot g(S) \tag{9}$$

y' - output sample value
y - input sample value

Value of S is used to determine which audio clip (of which articulation) should be used. For S that is higher than all boundary values the loudest articulation is returned as in listing [1]. A variable $clips$ contains an array of clips with assigned $maxVolume$ value of S. The array is sorted by $maxVolume$.

Listing 1: Clip selection based on volume

```
AudioClip GetClip(float S) {
    for (clip in clips) {
        if (S <= clip.maxVolume)
            return clip;
    }

    return clips.last;
}
```

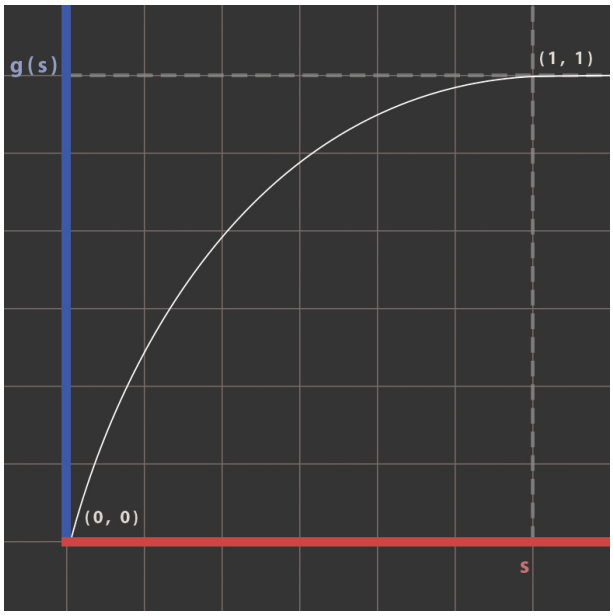

Figure 4: Demonstrative illustration showing the speed mapping curve

One of the major issues that I had to overcome was spurious wake of the skin that happened because of the noise. It was noticeable as doubled of tripled sounds at a single hit of a drum stick or hand. It was caused by instability and noise of controllers movement tracking. To solve it I restricted the frequency of hits to 5 Hz, which allows playing in 300 beats per minute tempo. This value is enough for playing polish traditional music and suppresses the spurious wakes.

3 Results

DRUM simulation is good enough to allow a musician intuitively play the virtual instrument using a technique that is similar to the one required for the real instrument. The main differences are caused by lack of physical force feedback from the instrument and relatively low inside out tracking precision provided by the device that was used for testing. However, during the test a user was able to play the virtual drum together with a violinist playing a real instrument. During this test the user intuitively executed musical ornaments that are characteristic for traditional polish music.

BASS simulation is playable. Besides being able to modulate the loudness using hand movement the users were not able to produce stable, sustainable sound. Filtration that I used to reduce hand tracking noise increased latency and made it difficult to control tempo and ornament the music.

Acknowledgements

Artur Ostrzołek - drum and bass 3d models
Patryk Petersson - sound effects
Maria Nizik - help with promotional videos

The project was granted a scholarship from Ministry of Culture and National Heritage of the Republic of Poland in 2021.

Ministerstwo
Kultury
i Dziedzictwa
Narodowego