Symulator VR gry na bebnie obreczowym polskim i basetli

Łukasz Nizik

1 Wstep

PROJEKT ma na celu stworzenie symulacji czasu rzeczywistego dla gogli wirtualnej rzeczywistości (dalej VR) pozwalajacej na interaktywna prezentacje instrumentów ludowych charakterystycznych dla sekcji rytmicznej kapel Polski Centralnej. Na potrzeby tej pracy wybrałem beben obreczowy polski oraz basy (basetle) dwu i trójstrunowe.

2 Metody

2.1 Technologia

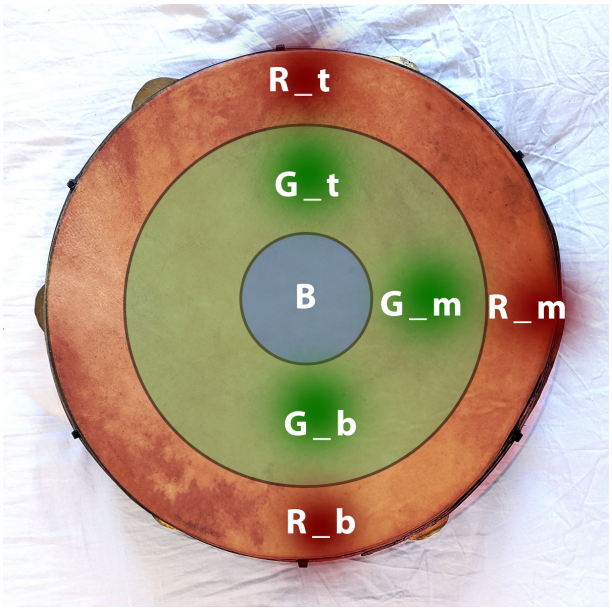
PLATFORMA DOCELOWA jest urzadzenie Oculus Quest 2. Do stworzenia symulacji posłużyłem sie silnikiem Unreal Engine 4.x. Pierwsze prototypy przygotowałem z wykorzystaniem silnika Unity 5, ale jego ograniczenia techniczne nie pozwoliły mi na zadowalajace zmniejszenie latencji dźwieku w stosunku do ruchu wykonywanego przez użytkownika.

2.2 Nagrania audio

NA potrzeby symulacji sporzadziłem nagrania audio prawdziwych instrumentów. Sa to pojedyncze dźwieki wydawane przez dany instrument.

Przy standardowej technice gry na bebnie obreczowym wyróżnia sie dwa podstawowe rodzaje uderzenia - pałka w skóre oraz reka w drewniana krawedź bebna. Te sposoby uderzania symulowane sa w stworzonej przeze mnie aplikacji. Sposób trzymania bebna to chwyt za drewniana rame u dołu.

Beben podzieliłem na strefy korzystajace z dźwieków z różnych pul klipów dźwiekowych. Ma to symulować zmiany barwy instrumentu oraz zachowania dzwonków w zależności od miejsca uderzenia w skóre bebna. W przypadku prawdziwego instrumentu uderzenie bliżej środka powoduje, że dźwiek skóry jest niższy, ale dzwonki odzywaja sie ciszej. Rysunek [1] przedstawia podział na strefy, którego użyłem. Strefa R_b znajduje sie w miejscu trzymania bebna, a R_t przy górnej krawedzi. Dla uproszczenia stref R_m i G_m użyłem dla dwóch krańców bebna symetrycznie (przód i tył). Do każdej se stref użyłem nagrań 3 rodzajów artykulacji t.j. lekkiego, średniego i mocnego uderzenia w skóre. Siła uderzenia ma wpływ na głośność oraz barwe bebna i dzwonków. W podobny sposób nagrany jest dźwiek uderzenia reka w krawedź bebna.



Rysunek 1: Strefy do nagrań bebna obreczowego

Basy 3-strunowe nagrane sa w 1 rodzaju artykulacji. Nastrojone sa do czestotliwości referencyjnej A440 Hz w stroju wiolonczelowym (bez struny C) A, D, G. Struny nagrane sa parami (kwintami). Ważnym elementem nagrania było uzyskanie jak najdłuższego dźwieku o niezmiennej dynamice.

2.3 Implementacja - basy

KOmponent odgrywajacy dźwiek basów odtwarza ciagły, zapetlony dźwiek, którego głośność jest proporcjonalna do predkości ruchu smyczkiem wzdłoż wektora stycznego do strun. Zapetlenie dźwieku korzysta z techniki cross-fade (przenikania). Nagranie audio basów jest fragmentem stabilnego pociagniecia smyczkiem z ucietym poczatkiem (atak) oraz końcem (zwolnienie). Odtwarzanie w petli polega na rozpoczeciu odtwarzania nastepnego powtórzenia nagranego klipu przed zakończeniem poprzedniego. Aby zachować stała głośność miejsce zapetlenia musi być zgodne w fazie lub korzystać z odpowiedniej funkcji konserwacji energii dla efektu przenikania. W momencie gdy odgrywane sa 2 klipy jednocześnie głośność

pierwszego zmniejsza sie, a drugiego zwieksza zgodnie ze wzorem [1].

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

a - wartość próbki z pierwszego nagrania
b - wartość próbki z drugiego nagrania

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

N_o - liczba nakładajacych sie próbek
i_a - indeks aktualnie odtwarzanej próbki pierwszego (a) nagrania
O_{start} - indeks próbki, na której zaczyna sie zapetlenie

Do odtwarzania zapetlajacego sie dźwieku stworzyłem klase w jezyku C++ o nazwie `ULoopPlayer`, która implementuje interfejs UE4 (Unreal Engine 4) `USynthComponent`. Jedna z metod tego interfejsu jest `int32 OnGenerateAudio(float* OutAudio, int32 NumSamples)`. Metoda ta powinna zwracać przez wskaźnik `OutAudio` na dane próbek, które nastepnie zostana odtworzone przez karte dźwiekowa. Liczba próbek jaka należy wygenerować jest równa rozmiarowi bufora audio pomnożonego przez liczbe kanałów.

Podczas gry na prawdziwym instrumencie smyczek dociskany jest do strun, a reka opiera sie na smyczku. Grajacy czuje zarówno siłe nacisku, tarcia smyczka o struny jak i kat nachylenia smyczka. Obecnie kontrolery VR nie przesyłaja informacji zwrotnej pozwalajacej odwzorować te zjawiska. Przez to miedzy innymi utrzymanie smyczka na strunach staje sie bardzo trudne. Stworzona przeze mnie symulacja automatycznie zapewnia umiejscowienie smyczka, tak aby zewnetrzna cześć włosia zawsze stykała sie ze strunami. Dodatkowo zdefiniowałem ograniczenie nie pozwalajace na tak zwane przeciagniecie smyczka za daleko, przez co straciłby on kontakt ze strunami. W edytorze silnika ustawiłem obiekt $StringContactPoint$, który wyznacza pozycje P_S oraz orientacje punktu styku smyczka i strun (rysunek [**??**]). Użytym przeze mnie punktem odniesienia P_B (pozycja, orientacja) dla smyczka jest miejsce chwytania smyczka (rysunek [3]). Oznaczenia osi wg. standardu UE4:

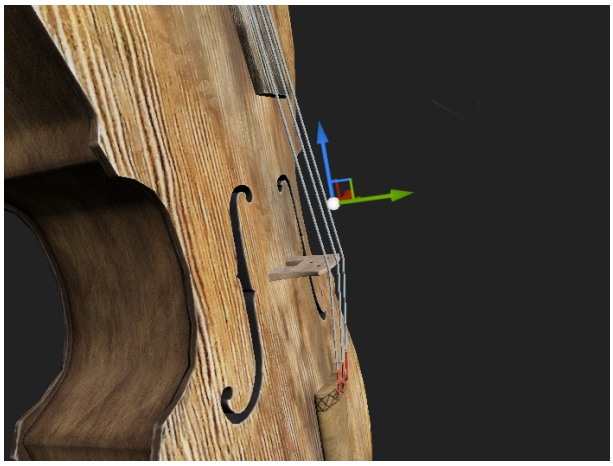
czerwony - oś X $Forward$
zielony - oś Y $Right$
niebieski - oś Z Up

Równania [3 4 5 6] przedstawiaja sposób liczenia nowej macierzy rotacji smyczka na podstawie wzajemnego położenia punktów P_B oraz P_S.

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

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

Y_S - wektor osi Y dla lokalnego układu współrzednych w punkcie styku na strunach (pozycja P_S), w konwencji Unreal Engine 4 jest to wektor $Right$



Rysunek 2: Punkt styku włosia smyczka i strun

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

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

Implementacja pozwala na symulacje basów zarówno dwu jak i trzystrunowych. Dodatkowo możliwa jest zmiana strun za pomoca zmiany wychylenia smyczka. Kat miedzy wektorami $Forward$ obiektu oznaczajacego miejsce chwytu smyczka, a obiektu $StringContactPoint$ mapowany jest na nagranie o odpowiedniej wysokości. Jeśli iloczyn skalarny tych wektorów jest dodatni bedzie to użyty jest dźwiek wyższej pary strun, jeśli ujemny to pary niższych strun. Orientacja tych wektorów widoczna jest na rysunkach 3 i **??** jako czerwone strzałki.

2.4 Implementacja - beben

Każda z próbek wyjściowych mnożona jest przez głośność zależna od predkości liniowej pałki na jej końcu uderzajacym w skóre bebna (współczynnik predkości S). Pod uwage biore tylko składowa wektora predkości równoległa do wektora normalnego powierzchni skóry bebna (wzory [7], [8]) o przeciwnym zwrocie.

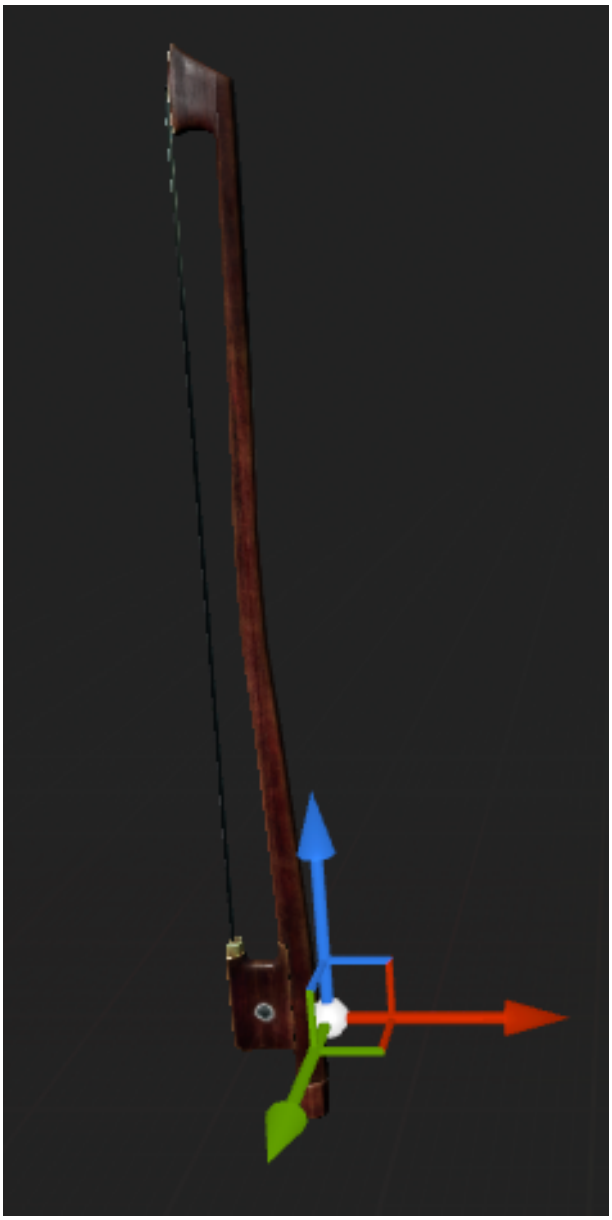
$$\vec{V} = \frac{\overrightarrow{\delta P}}{\delta t} \tag{7}$$

V - wektor predkości końcówki pałki
$\overrightarrow{\delta P}$ - różnica pozycji pałki miedzy kolejnymi krokami symulacji
$\overrightarrow{\delta t}$ - różnica czasu miedzy kolejnymi krokami symulacji

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

S - współczynnik predkości pałki
$\overrightarrow{D_n}$ - wektor normalny płaszczyzny skóry bebna
C - stała normalizacji

Wartość ujemna iloczynu skalarnego $\vec{V} \cdot \overrightarrow{D_n}$ oznacza uderzenie w tylna cześć skóry. Dla tej sytuacji symulacja nie generuje dźwieku wyjściowego.



Rysunek 3: Punkt chwytu dłoni za smyczek

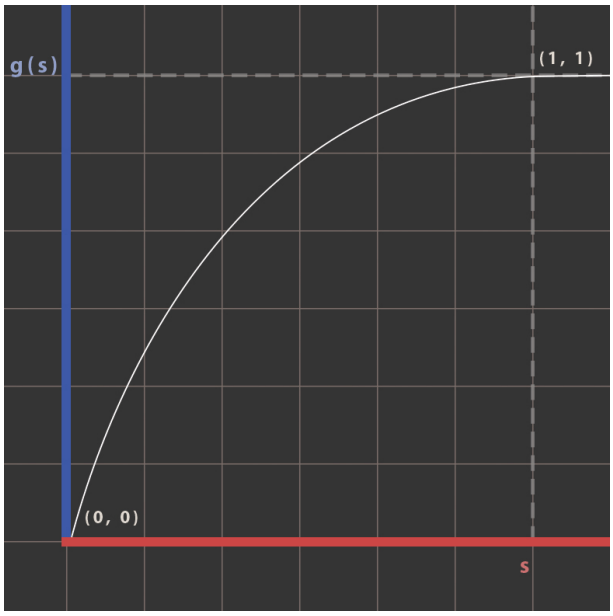
Próbka pochodzaca bezpośrednio z nagrania przemnożona jest przez współczynnik S przetworzony eksperymentalnie dobrana krzywa (rysunek [4]) $g(s)$ według wzoru [9].

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

y' - wartość próbki wyjściowej dla karty dźwiekowej
y - wartość próbki odczytana z nagrania

W zależności od wartości S zwracany jest inny klip audio. Jeśli wartość S bedzie wieksza od wszystkich maksymalnych wartości klipów zostanie zwrócony ten o najwyższej głośności co przedstawia listing [1]. Zmienna $clips$ zawiera tablice klipów audio wraz z $maxVolume$ maksymalnymi wartościami dla S. Tablica posortowana jest rosnaco według pola $maxVolume$.

Listing 1: Algorytm wyboru klipu od głośności



Rysunek 4: Ilustracja pogladowa krzywych przetwarzania głośności dźwieku

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

    return clips.last;
}
```

W trakcie grania na wirtualnym bebnie czesto pojawiały sie sytuacje fałszywego wzbudzenia instrumentu. Pojedyncze uderzenie uruchamiało odtwarzanie klipu dwu lub trzykrotnie. Jest to spowodowane niestabilnościa kontrolerów. Pozycja i orientacja jest niedokładna oraz obarczona szumem, co szczególnie objawia sie przy szybkich ruchach takich jak machniecie pałka. Na potrzeby aplikacji zastosowałem ograniczenie czestotliwości uderzeń do 5 Hz co pozwala na zagranie 300 uderzeń na minute. Jest to wartość wystarczajaca do grania jednocześnie zapobiegajac fałszywym wzbudzeniom.

3 Rezultaty

Symulacja bebna pozwala na płynne granie. Technika ruchu potrzebna do wzbudzenia bebna w symulatorze jest zbliżona do ruchu wykonywanego przez muzyków z prawdziwym bebnem. Główne różnice spowodowane sa brakiem fizycznego czucia sił działajacych na instrument oraz niedokładnościa śledzenia kontrolerów dłoni wynikajaca z jakości urzadzenia użytego do testów. W trakcie testów bez problemu udało sie nagrać utwór zagrany ze skrzypcami. W trakcie gry użytkownik był w stanie utrzymać stabilne tempo pozwalajace na synchronizacje z drugim instrumentem oraz wykonywać ozdobniki charakterystyczne dla polskiej muzyki tradycyjnej.

Basy sa w stanie podstawowej grywalności, jest możliwość modulacji głośności szybkościa pociagnie-

cia, jednak dźwiek nie jest płynny. Przez zastosowane filtracje zmniejszajace szum ruchu kontrolerów bardzo trudno jest utrzymać stabilne tempo grania lub wykonać ozdobniki.

Podziekowania

Artur Ostrzołek - grafika 3d basów i bebna
Patryk Petersson - realizacja audio
Maria Nizik - pomoc przy nagraniach wideo

Ministerstwo Kultury i Dziedzictwa Narodowego

Zrealizowano w ramach stypendium Ministra Kultury, Dziedzictwa Narodowego i Sportu 2021.