AKADEMIA GÓRNICZO – HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI

PRACA DYPLOMOWA
magisterska

Implementacja pogłosu hybrydowego w technice VST

Implementation of hybrid reverb using VST technique

Autor: **Adam Korytowski**

Kierunek studiów: Inżynieria Akustyczna

Opiekun pracy: **dr Marek Pluta** ……

podpis

Kraków, rok 2020

OŚWIADCZENIA STUDENTA

Kraków, dnia 10.09.2020

Adam Janusz Korytowski
Imiona i nazwisko studenta

Inżynieria Akustyczna, magisterskie, stacjonarne
Kierunek, poziom, forma studiów

Wydział Inżynierii Mechanicznej i Robotyki
Nazwa Wydziału

Dr Marek Pluta
Imiona i nazwisko opiekuna pracy dyplomowej

Ja niżej podpisany(– a) oświadczam, że:

jako twórca pracy dyplomowej magisterskiej pt.
Implementacja pogłosu hybrydowego w technice VST

1. **uprzedzony(-a) o odpowiedzialności karnej** na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2018 r. poz. 1191, z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, **a także uprzedzony o odpowiedzialności dyscyplinarnej** na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668, z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.” **niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy;**
2. praca dyplomowa jest wynikiem mojej twórczości i nie narusza praw autorskich innych osób;
3. wersja elektroniczna przedłożonej w wersji papierowej pracy dyplomowej jest wersją ostateczną, która będzie przedstawiona komisji przeprowadzającej egzamin dyplomowy;
4. praca dyplomowa nie zawiera informacji podlegających ochronie na podstawie przepisów o ochronie informacji niejawnych ani nie jest pracą dyplomową, której przedmiot jest objęty tajemnicą prawnie chronioną;
5. [TAK]** udzielam nieodpłatnie Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie licencji niewyłącznej, bez ograniczeń czasowych, terytorialnych i ilościowych na udostępnienie mojej pracy dyplomowej w sieci Internet za pośrednictwem Repozytorium AGH.

……

czytelny podpis studenta

Jednocześnie Uczelnia informuje, że:

1. zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668, z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studentaw wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem Jednolitego Systemu Antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych;

2. w świetle art. 342 ust. 3 pkt 5 i art. 347 ust. 1 ustawy Prawo o szkolnictwie wyższym i nauce minister właściwy do spraw szkolnictwa wyższego i nauki prowadzi bazę danych zwaną repozytorium pisemnych prac dyplomowych, która obejmuje: tytuł i treść pracy dyplomowej; imiona i nazwisko autora pracy dyplomowej; numer PESEL autora pracy dyplomowej, a w przypadku jego braku – numer dokumentu potwierdzającego tożsamość oraz nazwę państwa, które go wydało; imiona i nazwisko promotora pracy dyplomowej, numer PESEL, a w przypadku jego braku – numer dokumentu potwierdzającego tożsamość oraz nazwę państwa, które go wydało; imiona i nazwisko recenzenta pracy dyplomowej, numer PESEL, a w przypadku jego braku – numer dokumentu potwierdzającego tożsamość oraz nazwę państwa, które go wydało; nazwę uczelni; datę zdania egzaminu dyplomowego; kierunek, poziom i profil studiów. Ponadto, zgodnie z art. 347 ust. 2-5 ustawy Prawo o szkolnictwie wyższym i nauce ww. dane wprowadzają do Zintegrowanego Systemu Informacji o Szkolnictwie Wyższym i Nauce POL-on (System POL-on) rektorzy. Dostęp do danych przysługuje promotorowi pracy dyplomowej oraz Polskiej Komisji Akredytacyjnej, a także ministrowi w zakresie niezbędnym do prawidłowego utrzymania i rozwoju repozytorium oraz systemów informatycznych współpracujących z tym repozytorium. Rektor wprowadza treść pracy dyplomowej do repozytorium niezwłocznie po zdaniu przez studenta egzaminu dyplomowego. W repozytorium nie zamieszcza się prac zawierających informacje podlegające ochronie na podstawie przepisów o ochronie informacji niejawnych.

---

* - niepotrzebne skreślić;

** - należy wpisać TAK w przypadku wyrażenia zgody na udostępnienie pracy dyplomowej,
NIE – w przypadku braku zgody; nieuzupełnione pole oznacza brak zgody na udostępnienie pracy.

Akademia Górniczo– Hutnicza im. Stanisława Staszica Kraków, 10.09.2020

Wydział Inżynierii Mechanicznej i Robotyki

Kierunek studiów: Inżynieria Akustyczna

Specjalność: Inżynieria Dźwięku w Mediach i Kulturze (M)

Adam Janusz Korytowski
Imiona i nazwisko studenta

Praca dyplomowa magisterska
Implementacja pogłosu hybrydowego w technice VST
(tytuł pracy)

Opiekun: dr Marek Pluta

STRESZCZENIE

Celem pracy magisterskiej było stworzenie programu komputerowego w formie wtyczki VST symulującego efekt akustycznego zjawiska pogłosu. Implementacja aplikacji odbyła się korzystając z frameworku JUCE w języku C++. Pogłos został stworzony w oparciu o literaturę opisującą stosowane rozwiązania realizacji sztucznego pogłosu oraz znajomość percepcyjnych zjawisk przestrzennych. Implementacja poprzedzona była obiektywną oceną zaimplementowanych wcześniej wersji pogłosu za pomocą stworzonych na potrzeby pracy parametrów. Na tej podstawie dokonany został wybór sposobów implementacyjnych, które zostały zastosowane przy docelowej implementacji, tym samym uzyskując hybrydowe rozwiązanie. Aplikacja posiada interfejs użytkownika zawierający elementy pozwalające na modyfikację parametrów pogłosu w trakcie działania programu. Wtyczka umożliwia nałożenie efektu pogłosu na sygnał cyfrowy w oprogramowaniu typu DAW.

AGH University of Science and Technology Kraków, 10 of September 2020

Faculty of Mechanical Engineering and Robotics

Field of Study: Acoustic Engineering

Specialization: Audio Engineering in Arts and Media

Adam Janusz Korytowski
(First name and family name of the student)

Master Diploma Thesis
Implementation of hybrid reverb using VST technique
(Thesis title)

Supervisor: Marek Pluta, PhD

SUMMARY

The purpose of the work was to create a computer program – VST plugin being an effect of acoustic reverberation. The application was implemented in C++ language using JUCE framework. The effect was created according to literature which describe existing ways of implementing artificial reverberators as well as knowledge of perceptual spatial phenomenons. The implementation was preceded by objective assessment of previously implemented versions of reverberators using parameters created for needs of the work. Based on the assessment results, implementation methods have been chosen obtaining hybrid version of the effect. In the application there is a graphical user interface, which contains elements allowing to modify reverberation parameters. The plugin can be used in DAW software on a digital signal to receive the signal with reverberation effect.

Kraków, 10.09.2020

AKADEMIA GÓRNICZO– HUTNICZA
WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI

TEMATYKA PRACY DYPLOMOWEJ

Adam Korytowski
imię i nazwisko studenta

Tytuł pracy dyplomowej:
Implementacja pogłosu hybrydowego w technice VST

Promotor pracy: dr Marek Pluta
Recenzent pracy: dr hab. inż. Robert Barański

……

Podpis dziekana

PLAN PRACY DYPLOMOWEJ

1. Omówienie tematu pracy i sposobu realizacji z promotorem.
2. Zebranie i opracowanie literatury dotyczącej tematu pracy.
3. Przeprowadzenie badań (obliczeń)
4. Opracowanie wyników badań.
5. Analiza wyników badań, ich omówienie i zatwierdzenie przez promotora.
6. Opracowanie redakcyjne.

Praktyka (dyplomowa):
Katedra Mechaniki i Wibroakustyki

Kraków, …… ……
data *podpis dyplomanta*

……
podpis promotora

Termin oddania do dziekanatu: …… 20… r.

Podziękowania dla dr Marka Pluty
za inspirację oraz pomoc w realizacji pracy.

1. Wstęp

Zjawisko pogłosu akustycznego towarzyszy człowiekowi w niemal każdym momencie życia. Pełni kluczową rolę w odczuwaniu przestrzeni akustycznej oraz lokalizacji źródła dźwięku w pomieszczeniu. Bardzo często pogłos towarzyszący dźwiękowi bezpośredniemu wiąże się z jego atrakcyjnością dla człowieka – zarówno naturalny pogłos pomieszczenia (np. w sali koncertowej), jak i przy odsłuchu we własnych warunkach odsłuchowych (nagranie z naturalnym, zarejestrowanym pogłosem lub pogłosem sztucznym dodanym na etapie miksu utworu).

Badania dotyczące pogłosu (realizacja lub ocena jego jakości) prowadzone pod koniec XX wieku często wiązały się z koniecznością wykorzystania do tego celu akustycznego pola swobodnego lub dyfuzyjnego oraz stosunkowo skomplikowanego toru audio [1]. Współczesne możliwości znacznie to ułatwiają. Ułatwienia te to przede wszystkim możliwość odwzorowania większości zjawisk w domenie cyfrowej, jak również większe możliwości obliczeniowe. Korzystając z narzędzi cyfrowych, w przypadku potrzeby dokonania niewielkiej zmiany w parametrach pogłosu, wystarczy niewielka modyfikacja kodu oraz odczekanie niewielkiej ilości czasu w celu otrzymania nowej symulacji pogłosu. Co więcej, większe możliwości obliczeniowe pozwalają realizować większą liczbę testów w tym samym czasie, a co za tym idzie istnieje możliwość przetestowania większej liczby rozwiązań.

W związku z powyższym otwierają się szersze możliwości eksperymentowania z parametrami pogłosu oraz dostępnymi narzędziami – w niniejszej pracy możliwości te zostają wykorzystane do stworzenia cyfrowej symulacji pogłosu. Implementacja jest poprzedzona obiektywną oceną jakości brzmienia poszczególnych jego elementów.

1.1. Cel i zakres pracy

Celem pracy jest projekt i implementacja cyfrowej symulacji zjawiska pogłosu w formacie VST poprzedzona oceną elementów składowych pogłosu przy użyciu stworzonych na potrzeby pracy parametrów.

Praca obejmuje przegląd głównych idei realizacji sztucznego pogłosu oraz percepcyjnych zjawisk przestrzennych, testy wybranych rozwiązań realizacji

poszczególnych elementów pogłosu i ich ocenę na podstawie zaproponowanych parametrów w sposób obiektywny, jak również implementację docelowej aplikacji w formacie VST. Docelowa aplikacja będzie pogłosem w postaci efektu uatrakcyjniającego sygnał, w odróżnieniu do symulacji rzeczywistego zjawiska pogłosu akustycznego. Implementacja będzie hybrydą łączącą elementy istniejących rozwiązań na podstawie ich oceny zaproponowanymi przez autora parametrami. Praca omawia również szczegóły koncepcji realizacji docelowego pogłosu oraz algorytm prowadzący do uzyskania finalnego efektu.

2. Symulacja i percepcja zjawiska pogłosu

Zjawisko pogłos akustycznego występuje naturalnie w większości środowisk [2]. Zdefiniowane jest jako czas, po którym poziom natężenia dźwięku spadnie o 60 dB od wyłączenia źródła dźwięku [3]. Fale emitowane ze źródła dźwięku zwane są dźwiękiem bezpośrednim, a odbite od elementów pomieszczenia tworzą fale wtórne, które nakładają się na fale dźwięku bezpośredniego. Zawartość częstotliwościowa każdego z odbić zależy od kierunkowości źródła oraz pochłaniania odbijających powierzchni [2]. Dzięki obecności kopii sygnału, podczas emisji wzrasta energia akustyczna dźwięku odbitego. Po wyłączeniu źródła następuje faza zaniku dźwięku odbitego, wskutek zmniejszania się ciśnienia akustycznego [4]. Dwa najważniejsze czynniki wpływające na pogłos to całkowita objętość wnętrza oraz łączne pochłanianie przez wszystkie elementy wnętrza. Pomiary praktyczne Sabine'a doprowadziły do powstania wzoru opisującego czas pogłosu w pomieszczeniu [3, 5]:

$$T = 0{,}161 \frac{V}{A} \tag{2.1}$$

gdzie:

V – objętość pomieszczenia [m^3],

A – chłonność akustyczna pomieszczenia [m^2],

T – czas pogłosu [s].

Objętość pomieszczenia i średni współczynnik pochłaniania nie są od siebie zależne. Zwiększenie więc objętości, podobnie jak zmniejszenie średniego współczynnika pochłaniania zwykle skutkuje zwiększeniem czasu pogłosu [5]. Skorygowany wzór nazywany jest wzorem Eyringa i przyjmuje następującą postać [3]:

$$T = \frac{-0{,}0713V}{P\log(1-\alpha)} \tag{2.2}$$

gdzie:

V – objętość pomieszczenia [m^3],

P – suma pól wszystkich powierzchni w pomieszczeniu [m^2],

α – współczynnik pochłaniania.

Pogłos może być także wytworzony w sposób sztuczny na wiele sposobów [6].

2.1. Główne metody realizacji

2.1.1. Rozwiązania elektromechaniczne

W klasycznym podejściu do realizacji pogłosu wykorzystywano komorę pogłosową – pomieszczenie zaprojektowane specjalnie po to, aby uzyskać w nim jak najdłuższy czas pogłosu. W pomieszczeniu tym, długi czas pogłosu uzyskuje się dzięki odpowiednio dużej objętości, silnie odbijającym, nierównoległym ścianom oraz zastosowaniu dodatkowych, silnie odbijających powierzchni. Realizując nagranie w takim pomieszczeniu uzyskuje się sygnał wzbogacony o jego naturalny pogłos. Można uzyskać wiele wersji sygnału w zależności od pozycji mikrofonu w pomieszczeniu oraz pozycji źródła [7].

Jednym z najwcześniejszych rozwiązań pozwalających na zastosowanie pogłosu w nagraniach był pogłos płytowy. Jego konstrukcja opiera się na prostokątnej, stalowej, obramowanej płycie z przymocowanym przetwornikiem elektromechanicznym. Przetwornik rejestruje sygnał dźwiękowy, którego drgania są przenoszone na płytę. Następnie drgania płyty są odbierane przez przetwornik elektroakustyczny, tym sposobem jest uzyskiwany sygnał z pogłosem powstałym dzięki drganiu płyty. Odpowiedź impulsowa takiego układu różni się od odpowiedzi impulsowej pomieszczenia – występuje brak wyraźnych odbić, a jako całość jest bardziej gładka i bardziej przypominająca szum [2].

Podobnym rozwiązaniem jest pogłos sprężynowy, w którym przetwornik elektromechaniczny wprawia w drgania sprężynę. Na drugim końcu sprężyny znajduje się przetwornik elektroakustyczny odbierający sygnał z dodanym pogłosem powstałym dzięki odbiciom fali akustycznej od końców sprężyny. W odróżnieniu do pogłosu płytowego, sprężyna jest w stanie wykonywać drgania zarówno podłużne, jak i poprzeczne. Złożenie tych dwóch rodzajów drgań prowadzi do charakterystycznego brzmienia łączącego w sobie cechy brzmienia rzeczywistego pomieszczenia z pogłosem płytowym [2].

2.1.2. Pogłos splotowy

Najważniejszym pojęciem w akustyce pomieszczeń jest odpowiedź impulsowa pomieszczenia. Opisuje ona akustykę pomieszczenia i jest realizowana jako rejestracja nieruchomego źródła dźwięku w jednym punkcie. Jeśli odpowiedź impulsowa pomieszczenia jest znana, najbardziej wierny sygnał pogłosowy można uzyskać wykonując operację splotu sygnału z odpowiedzią impulsową. Operację taką można wykonać traktując każdą próbkę odpowiedzi impulsowej jako współczynnik filtru FIR, a następnie dokonując nim filtracji sygnału wejściowego [2]. Definicja splotu przedstawia się jako zależność [8]:

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) \tag{2.3}$$

gdzie:

f(m), g(n) – sygnały poddawane operacji splotu.

Ze względu na małą wydajność obliczeniową operacji splotu w dziedzinie czasu, stosuje się podejście wykorzystujące tożsamość operacji splotu w dziedzinie czasu z operacją mnożenia w dziedzinie częstotliwości. Wykonywanie obliczeń w dziedzinie częstotliwości jest znacznie bardziej wydajne obliczeniowo. Często jest to jedyne

możliwe rozwiązanie w przypadku konieczności działania algorytmu w czasie rzeczywistym [2].

2.1.3. Rozwiązanie oparte o filtry wszechprzepustowe i grzebieniowe

Jednymi z najważniejszych, pionierskich prac dotyczących realizacji sztucznego pogłosu są prace M. Schroedera z lat 60 XX wieku [9, 10]. Zaproponował on podejście wykorzystujące filtry wszechprzepustowe i grzebieniowe jako narzędzie umożliwiające symulację pogłosu. Filtr wszechprzepustowy przedstawia się jako zależność [2]:

$$y(n) = -g * x(n) + x(n-m) + g * y(n-m) \tag{2.4}$$

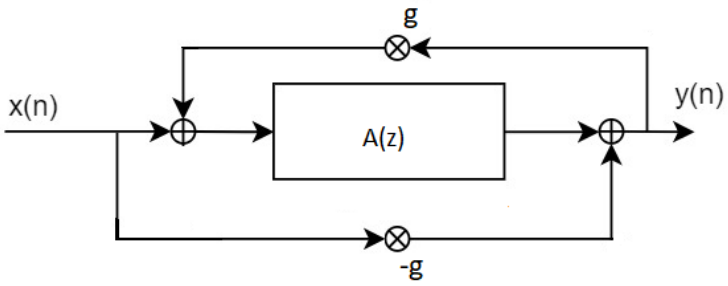
gdzie:

x(n) – sygnał wejściowy,

y(n) – sygnał wyjściowy,

m – długość opóźnienia w próbkach,

g – wzmocnienie.



Rys. 2.1. Struktura filtru wszechprzepustowego [2].

W schemacie filtru wszechprzepustowego przedstawionego na Rys. 2.1, współczynnik *g* oznacza wzmocnienie, a *A(z)* linię opóźniającą. Zastosowanie szeregu takich filtrów pozwala na uzyskanie odpowiedzi impulsowej o płaskiej charakterystyce [11]. Podejście oparte o filtry wszechprzepustowe w krótkim czasie

stało się standardem w niemal wszystkich stosowanych algorytmach sztucznego pogłosu [1].

2.1.4. Feedback Delay Network

W 1982 roku J. Stautner i M. Puckette jako strukturę realizującą pogłos zaproponowali sieć linii opóźniających połączonych w pętli sprzężenia zwrotnego [12]. W późniejszym czasie struktura taka została nazwana siecią Feedback Delay Network. Pierwotnie zaproponowana sieć miała postać [2]:

$$y(n) = x(n-m) + g * y(n-m) \tag{2.5}$$

gdzie:

x(n) – sygnał wejściowy,

y(n) – sygnał wyjściowy,

m – długość opóźnienia w próbkach,

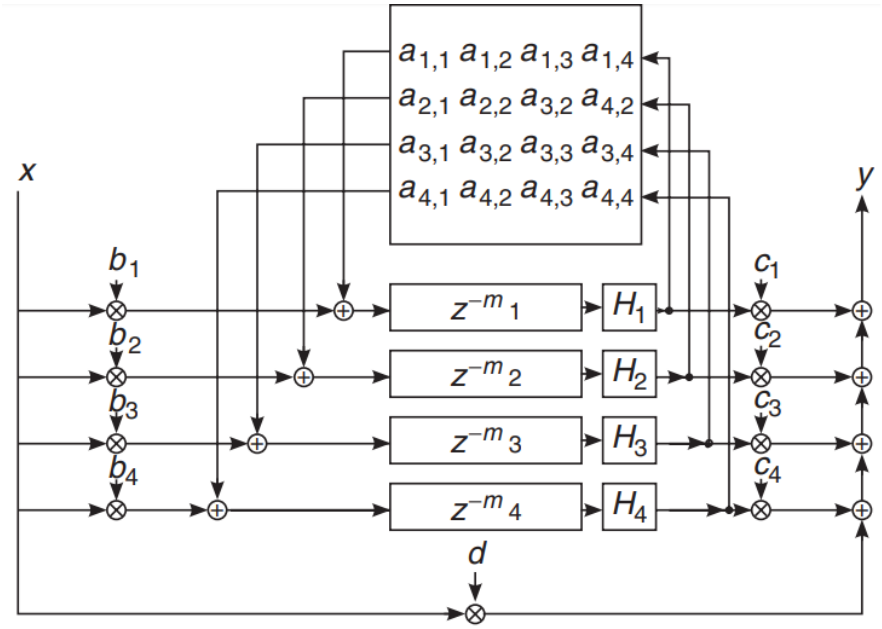
g – wzmocnienie.

Linia opóźniająca *m* w filtrze wszechprzepustowym została zastąpiona szeregiem linii o różnych długościach, *a* parametr *g* oznaczający wzmocnienie – macierzą sprzężenia zwrotnego *G* daną jako:

$$G = g \begin{bmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \Big/ \sqrt{2} \tag{2.6}$$

Dzięki zastosowaniu sieci Feedback Delay Network uzyskać można wysoką gęstość późnego ogona pogłosowego [2]. Zależy ona jednak od liczby linii opóźniających i wzajemnych relacji długości linii opóźniających. Według teorii,

największą gęstość późnego ogona pogłosowego można uzyskać, kiedy stosunki długość linii opóźniających są liczbami pierwszymi [1, 4, 5].



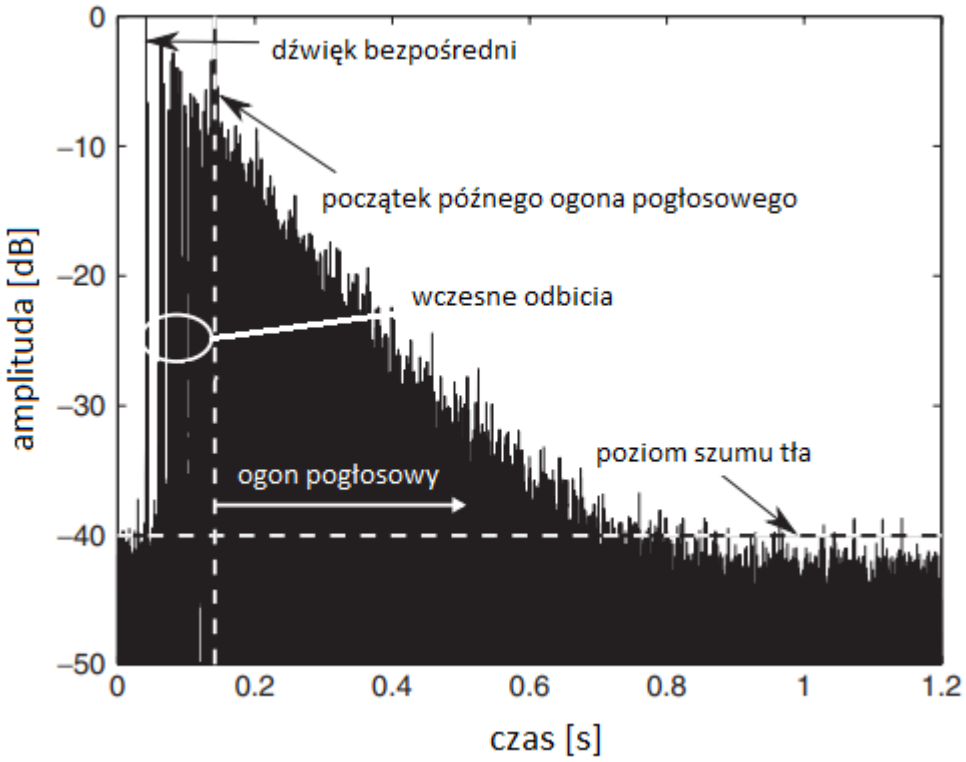
Rys. 2.2. Schemat podstawowej sieci pogłosowej Feedback Delay Network [9].

Badania nad rozwojem sieci Feedback Delay Network zostały rozwinięte w pracy Jota [1].

2.1.5. Cyfrowe symulacje pogłosu

Od wczesnych lat rozwoju prac nad realizacją sztucznego pogłosu najbardziej popularne jest rozgraniczenie go na dwie części: wczesne odbicia (pierwsze z nich to około 80 ms od dźwięku bezpośredniego, zależy to jednak od wielkości pomieszczenia) oraz późny pogłos (pozostała część pogłosu) [1, 2, 13] (Rys 2.3). Wczesne odbicia to część pogłosu złożona z dyskretnych odbić imitujących pierwsze odbicia od ścian lub innych elementów w pomieszczeniu. O rozpoznaniu kierunku, z którego dochodzi dźwięk decyduje pierwsza fala dźwiękowa. Czas opóźnienia w stosunku do dźwięku bezpośredniego oraz stosunki amplitud wczesnych odbić zależą w głównej mierze od kształtu pomieszczenia i pozycji źródła względem odbiornika. Odbicia te odgrywają główną rolę w subiektywnym odczuciu przestrzenności dźwięku, człowiek kojarzy

percepcyjnie właściwości sygnału z warunkami akustycznymi, jakie panowałyby w rzeczywistości. Pomagają więc w lokalizacji źródła dźwięku przez odbiorcę. Późny pogłos, w odróżnieniu od wczesnych odbić jest niezależny od położenia źródła dźwięku i odbiornika – jest silnie zależny od geometrii pomieszczenia oraz właściwości pochłaniających i rozpraszających materiałów elementów w nim się znajdujących. W celu zaprojektowania algorytmu sztucznego pogłosu Jot [13] proponuje procedurę przyjęcia odpowiednich parametrów, aby uzyskać wynikowy sygnał, a następnie rozważyć zjawiska binauralne oraz inne rozwiązania w celu nadania pogłosowi realizmu. Na podstawie badań wykazano, że pogłos oparty na sieci linii opóźniających nie jest subiektywnie rozróżnialny z pogłosem z rzeczywistych pomieszczeń.



Rys. 2.3. Schematyczne przedstawienie odpowiedzi impulsowej z wyszczególnieniem dźwięku bezpośredniego, wczesnych odbić oraz późnego ogona pogłosowego [2].

2.2. Sposoby implementacji elementów pogłosu

2.2.1. Wzmocnienia linii opóźniających i poszczególnych odbić

Barron [1] zwrócił uwagę, że czas pogłosu nie jest jedynym wyznacznikiem jakości dźwięku w pomieszczeniu. Przeprowadził badania mające na celu zrozumienie jak ważny przy modelowaniu sztucznego pogłosu jest wpływ pierwszych odbić. Badania te polegały na analizie subiektywnych wrażeń osób badanych, którym przedstawiane były próbki dźwiękowe zawierające pogłos o różnych parametrach. Został zbadany między innymi wpływ pierwszych odbić bocznych na subiektywne zjawisko „wrażenia przestrzenności" pogłosu. Zjawisko to miało wynikać z właściwości wczesnych odbić bocznych w pomieszczeniu (10 – 80 ms). Ze względu na najwyższy poziom dźwięku, wczesne odbicia są najbardziej znaczące w percypowaniu pogłosu. Zjawisko „wrażenia przestrzenności" odczuwane przez osoby badane pojawiało się dla symulacji odbić bocznych i powodowało wrażenie „poszerzenia się" źródła dźwięku. Przy zwiększeniu poziomu dźwięku odbicia bocznego wrażenie przestrzenności zwiększało się. W badaniach został przebadany także wpływ odbicia od sufitu na wrażenie przestrzenności – wpływ odbicia of sufitu ma negatywny (niewielki) wpływ na wrażenie przestrzenności.

Według Jota [13], jeśli poszczególne linie opóźniające mają niejednakowy czas zaniku, skutkuje to wyraźnie słyszalnymi składowymi i ujawnia obecność tych linii. W celu upewnienia się, że sytuacja taka nie będzie miała miejsca, należy upewnić się, że wszystkie linie mają jednakowy czas zaniknięcia dźwięku. Również w pracy [9] autorzy zwracają uwagę, iż jedną z fundamentalnych zasad przy tworzeniu sztucznego pogłosu jest zadbanie o to, aby amplituda wszystkich składowych zanikała prawie jednakowo szybko. Ma to na celu zapewnienie jednakowego opadania różnych składowych częstotliwościowych zanikającego dźwięku.

2.2.2. Czasy opóźnienia linii opóźniających

Dobór czasów opóźnienia linii opóźniających nie jest kluczowy, pod warunkiem, że liczby je reprezentujące nie są przez siebie podzielne [13]. Dobór nieskorelowanych wartości jest niezbędny, aby uniknąć zjawiska echa trzepoczącego i uzyskać płaską

charakterystykę częstotliwościową [12]. Aby uniknąć zmniejszenia gęstości odbić oraz superpozycji, pożądane jest użycie liczb nieproporcjonalnych do siebie [10].

Krótkie czasy opóźnienia skutkują większym ubarwieniem dźwięku – sytuacja taka ma miejsce w małych pomieszczeniach. Zjawisko takie byłoby nienaturalne w pomieszczeniach o wysokim czasie pogłosu. Według autorów [12] zawartość ubarwień w złożonej sieci nie jest łatwa do oceny, kontrola zawartości tych ubarwień wymaga używania metody prób i błędów. Prawdopodobnie zjawisko to występuje w mniejszym stopniu dla sieci zawierających choć niewielką liczbę długich linii opóźniających, jednak wtedy trudniej kontrolować gęstość pogłosu [12]. Zjawisko to ma wytłumaczenie w interferencji między dźwiękiem bezpośrednim, a opóźnionym o niewielkie wartości, co skutkuje pojawieniem się filtru grzebieniowego. Zjawisko ma miejsce dla opóźnień około 10–50 ms, a szczególnie w okolicach 20 ms [1].

Według Beranka [14] najważniejszym wyznacznikiem jakości pogłosu jest czas pierwszego odbicia w stosunku do dźwięku bezpośredniego i jest to nawet ważniejsze niż czas pogłosu w pomieszczeniu, a optymalna wartość tego parametru to czas poniżej 20ms.

Schroeder proponuje także dobranie czasów opóźnienia jako liczby zawierające się w przedziale liczb o stosunku 1:1,5 [10].

2.2.3. Filtracja linii opóźniających

Schroeder zaproponował powiązanie wartości tłumienia linii opóźniających z ich zawartością częstotliwościową. Miało to na celu odwzorowanie faktu dłuższego czasu pogłosu dla niskich częstotliwości [10]. Stautner i Puckette zaproponowali, aby stosować filtr dolnoprzepustowy na wyjściu każdej linii opóźniającej w celu imitacji tłumienia dźwięku przez powietrze. W ich podejściu częstotliwości odcięcia filtrów zależą od czasu opóźnienia konkretnej linii [12].

Na trudność obiektywnej oceny jakości pogłosu wskazują autorzy publikacji [12]. Autorzy zauważyli, że dobór parametrów filtrów stosowanych na pierwszych odbiciach mocno wpływa na jakość pogłosu, natomiast trudno znaleźć ilościowy parametr pozwalający na obiektywną, liczbową ich ocenę. Według publikacji [12],

przypuszczalnie może zostać odkryta pewna metoda statystyczna wyboru właściwości wczesnych odbić, z uwzględnieniem ograniczeń percepcyjnych.

Istnieją następujące subiektywne sposoby na identyfikację niewystarczającej gęstości widmowej późnego pogłosu [13]:

- odpowiedź na sygnał impulsowy zawierać będzie "dzwonienie" poszczególnych składowych,
- odpowiedź na quasi–stacjonarny sygnał będzie zawierała nadmierny poziom niektórych częstotliwości (na przykład na niektórych nutach przy grze na instrumencie).

W celu wyłączenia czynnika subiektywnego z oceny pogłosu, w rozdziale 3 znajduje się zaproponowany w ramach pracy sposób oceny jego jakości w zależności od rodzaju filtracji.

2.2.4. Późny ogon pogłosowy

Późny pogłos jako zjawisko akustyczne składa się z nakładających się na siebie kopii dźwięku bezpośredniego. Symulując pogłos, zwiększenie gęstości późnego pogłosu można uzyskać dzięki zastosowaniu sprzężenia zwrotnego. Sposobami na zwiększenie gęstości może być także zwiększenie liczby linii opóźniających, uniezależnienie od siebie długości tych linii [10], a także dobór ich odpowiednich wzmocnień [9].

W literaturze niejednokrotnie pojawiają się także wzmianki o możliwości wykorzystania szumu do realizacji późnego pogłosu. Przykładowo, zostało stwierdzone, iż do tego celu może być użyty szum Gaussa, za jego pomocą można uzyskać późny pogłos bez zakolorowań w barwie [15]. Na możliwość wykorzystania szumu wskazuje także publikacja [16]. W tym przypadku autorzy proponują podzielenie sygnału na krótkie fragmenty, a następnie wykonywanie na nich operacji splotu z szumem. Szum ten, w odróżnieniu do szumu Gaussa nie ma składać się z liczb losowych, a przyjmuje wartości dyskretne ze zbioru: {– 1,0,1}. Według publikacji, zaskakującą cechą tego szumu jest fakt, iż nawet, jeżeli 90% jego próbek przyjmuje wartość 0, brzmi bardziej gładko niż szum Gaussa.

2.3. Zjawiska psychoakustyczne przy percypowaniu pogłosu

Mechanizmy słyszenia przestrzennego pozwalają na odczucie kierunku, z którego dobiega dźwięk, a także w pewnej mierze odległości od źródła dźwięku. Teoria oparta na podstawie wyników subiektywnych badań słuchowych głosi, iż w sytuacji, w której do uszu słuchacza dochodzi ten sam sygnał, słuchacz lokalizuje sygnał jako będący z przodu. W przypadku słyszenia sygnału przez słuchawki słuchacz odczuwa dźwięk w środku głowy [17].

Dwie najważniejsze wielkości opisujące słyszenie przestrzenne oraz możliwość lokalizacji źródła dźwięku przez człowieka to międzyuszna różnica czasu (*Interaural Time Difference, ITD*) oraz międzyuszna różnica poziomu (*Interaural Level Difference, ILD)* [17]:

- międzyuszna różnica poziomu polega na różnicy w natężeniach dźwięku docierającego do obu uszu słuchacza w momencie, gdy źródło dźwięku nie znajduje się dokładnie naprzeciw słuchacza. Zjawisko jest związane z faktem istnienia cienia akustycznego generowanego przez głowę odbiorcy (cień akustyczny zaczyna występować dla częstotliwości, przy których połowa długości fali staje się mniejsza od rozmiarów głowy). Dzięki temu, na podstawie przeszłych doświadczeń słuchacz jest w stanie z dużą rozdzielczością stwierdzić, z jakiego kierunku w płaszczyźnie horyzontalnej dochodzi dźwięk,
- międzyuszna różnica czasu to różnica fazy biorąca się z różnicy czasów dotarcia sygnału do obu uszu. Na tym zjawisku oparta jest kierunkowość słyszenia niższych częstotliwości.

Zjawiska te i fakt działania obu dla różnych zakresów częstotliwości opisuje teoria dupleksowa zaproponowana przez Reyleigha w 1907 roku [18]. W warunkach rzeczywistych, odbiorca dźwięku ma także do dyspozycji możliwość ruchów głową, które w przypadku nieoczywistego kierunku dochodzenia dźwięku pozwalają na wykrycie różnic fazowych i poprawną lokalizację. W przypadku odbierania sygnału sztucznego, zjawisko występuje jedynie słuchając sygnału korzystając z głośników (nie występuje korzystając ze słuchawek) [17]. Zostało także stwierdzone, że stopień wrażenia przestrzenności jest związany ze stopniem

niekoherencji sygnałów z obu kanałów [19]. W zależności od rodzaju sygnału mogą zachodzić także różne zjawiska percepcyjne takie jak: dwuuszne odmaskowanie, dwuuszne dudnienia, efekt pierwszeństwa i inne [17].

3. Ocena jakości symulacji pogłosu

Obiektywna ocena jakości pogłosu wymaga odpowiedniego doboru parametrów statystycznych i ich zastosowania w celu uzyskania wartości liczbowych będących podstawą do uzasadnienia oceny danego wariantu na tle innego. Posłużą do tego zdefiniowane na potrzeby pracy parametry (podrozdział 3.2) – będą one wyznacznikiem jakości pogłosów. Do zdefiniowania tych parametrów zostaną wykorzystane istniejące, znane parametry statystyczne stosowane do analizy widmowej i czasowej sygnału oraz do oceny podobieństwa sygnałów. Parametry te opisane są w podrozdziale 3.1. W podrozdziale 3.2 opisane są także warianty pogłosów zaimplementowanych przez autora. Przedstawione są również wartości niektórych parametrów statystycznych – w formie graficznej w dziedzinie częstotliwości lub czasu, jak również wartości liczbowe stworzonych na potrzeby pracy parametrów (tabele). Testy te przeprowadzone będą dla trzech sygnałów: mowa ludzka, sygnał MLS oraz nagranie instrumentu – gitary klasycznej w warunkach bezechowych. Dokonana zostanie również interpretacja wyników oraz wysunięte wnioski mające wpływ na opisaną w rozdziale 4 implementację docelowego pogłosu.

3.1. Opis wybranych parametrów statystycznych do analizy sygnałów

Opisane w podrozdziale 2.2 sposoby implementacji poszczególnych elementów pogłosu zawierają wskazówki dotyczące pożądanej zawartości widmowej i czasowej pogłosu oraz cech sygnału wpływających na wrażenie przestrzenności. W celu oceny jakości pogłosu przedstawia się stosowane parametry statystyczne mogące być w tym pomocne.

3.1.1. Widmowa gęstość mocy

Wielkość ta określa, jak moc sygnału jest rozprowadzona w dziedzinie częstotliwości. W przypadku sygnałów cyfrowych moc definiuje się abstrakcyjnie jako kwadrat wartości sygnału. Ogólnie, średnia wartość mocy zdefiniowana jest jako [20]:

$$P = \lim_{T\to\infty} a\frac{1}{T}\int_0^T |x(t)|^2 dt \tag{3.1}$$

gdzie:

P – średnia wartość mocy sygnału,

T – okres sygnału.

Jeśli sygnał może być potraktowany jako sygnał stacjonarny, to widmowa gęstość mocy jest transformatą Fouriera (3.2)

$$\hat{x}(\omega) = \int_0^T x(t)e^{-i\omega t}dt \tag{3.2}$$

gdzie:

$\hat{x}(\omega)$ – transformata Fouriera,

T – okres sygnału,

x(t) – sygnał poddawany transformacji,

ω – częstość kołowa,

t – czas,

funkcji autokowariancji tego sygnału, co w rezultacie, po przełożeniu do dziedziny dyskretnej daje [21]:

$$\phi(\omega) = \sum_{k=-\infty}^{\infty} r(k)e^{-i\omega t} \tag{3.3}$$

gdzie:

ø(ω) – widmowa gęstość mocy,

r(k) – funkcja autokowariancji sygnału.

3.1.2. Płaskość widmowa

Parametr ten pozwala na rozróżnienie sygnału tonalnego od szumowego [22]. Wartości parametru przedstawiane są w decybelach, gdzie im widmo sygnału jest bardziej płaskie, tym wartość parametru wyższa. Wartość szczytowa (0 dB) oznacza, iż sygnał ma widmo stałe [23]. Definicja płaskości widmowej to stosunek średniej geometrycznej i średniej arytmetycznej mocy widma sygnału [22, 23]:

$$Flatness = \frac{exp(\frac{1}{N}\sum_{n=0}^{N-1}\ln x(n))}{\frac{1}{N}\sum_{n=0}^{N-1}x(n)} \tag{3.4}$$

gdzie:

N – liczba próbek sygnału,

x(n) – moc widma sygnału.

3.1.3. Koherencja sygnałów

Wielkość ta to miara relacji pomiędzy dwoma sygnałami i informuje o zgodności ich faz w badanej częstotliwości [24]. Zdefiniowana jest jako [25]:

$$C_{xy}(f) = \frac{\left|G_{xy}(f)\right|^2}{G_{xx}(f)G_{yy}(f)} \tag{3.5}$$

gdzie:

$C_{xy}(f)$ – koherencja sygnałów x i y,

$G_{xy}(f)$ – wzajemna gęstość widmowa [26] między sygnałami x i y,

$G_{xx}(f)$, $G_{yy}(f)$ – wzajemne gęstości widmowe sygnałów x i y samych z sobą.

Koherencja przyjmuje wartości z zakresu [0;1], gdzie 0 oznacza brak koherencji, a 1 silną koherencję między sygnałami w badanej częstotliwości [24].

3.2. Testy stosowanych rozwiązań implementacji pogłosu

3.2.1. Implementowane warianty pogłosu

Warianty pogłosów zaimplementowane przez autora:

A) W celu zbadania wpływu doboru czasów opóźnienia na brzmienie pogłosu, przetestowano następujące techniki symulacji zjawiska pogłosu, implementując:

- pogłos złożony z linii opóźniających o czasach opóźnienia jako liczby losowe,
- pogłos złożony z linii opóźniających o czasach opóźnienia jako liczby wzajemnie pierwsze (w nawiązaniu do [10, 12, 13]),
- pogłos złożony z linii opóźniających o czasach opóźnienia zawierających się w przedziale liczb o stosunku 1:1.5 (w nawiązaniu do [10]).

B) Filtracja linii opóźniających:

- filtracja dolnoprzepustowa każdej linii opóźniającej w jednakowy sposób,
- filtracja dolnoprzepustowa o częstotliwości odcięcia tym niższej, im wyższa wartość opóźnienia linii opóźniającej (w nawiązaniu do [10]),
- filtracja w zakresie pochłaniania materiałów (większość materiałów, z których składają się ściany mają największy stopień pochłaniania energii akustycznej w zakresie 500 Hz – 2 kHz) (w nawiązaniu do [12]).

C) Późny ogon pogłosowy (w nawiązaniu do podrozdziału 2.2.4):

- pogłos złożony z linii opóźniających ze sprzężeniem mnożony przez filtrowany szum biały,
- pogłos złożony z linii opóźniających ze sprzężeniem zagęszczony odbiciami w późnej fazie pogłosu przez dodanie tam większej liczby linii opóźniających.

W celu obiektywnego porównania zaimplementowanych wersji, na potrzeby niniejszej pracy zdefiniowano parametr będący podstawą do oceny wpływu doboru wartości czasów opóźnienia linii opóźniających, późnego pogłosu oraz sposobu filtracji na jakość pogłosu. Zdefiniowany parametr jest odpowiedzią na powtarzające się w literaturze [9, 10, 13] podkreślenia wpływu wysokiej gęstości oraz płaskości widma na jakość pogłosu. Zdefiniowany przez autora parametr przedstawia się jako zależność:

$$B_x := \frac{\frac{H_{xy} + H_{xz}}{2} E(\emptyset_x)}{\sigma(flatness_x) * \sigma(\emptyset_x)} \tag{3.6}$$

gdzie:

H_{xy} – współczynnik wyrażony liczbą z zakresu [0;1] będący stosunkiem liczby punktów, w których płaskość widma sygnału *x(n)* jest większa, niż płaskość widma sygnału *y(n)* do liczby wszystkich punktów wektora płaskości widma,

x, y, z – kolejne wersje pogłosu,

σ – średnie odchylenie standardowe,

E – średnia arytmetyczna,

$\emptyset_x$ – widmowa gęstość mocy sygnału *x(n)*,

$flatness_x$ – płaskość widmowa sygnału *x(n)*.

Wzór przyjmuje formę analogiczną dla sygnałów *y(n)* oraz *z(n)*.

D) Psychoakustyczne zjawiska przestrzenne

- pogłos z różnicą w poziomie pierwszych odbić bocznych – w celu potwierdzenia zaobserwowanej tezy o większej przestrzenności przy zwiększaniu amplitudy odbicia bocznego (w nawiązaniu do [1]),

- pogłos z zastosowaniem różnicy czasu opóźnienia między lewym i prawym kanałem dla linii opóźniających – w celu symulacji zjawiska międzyusznej różnicy czasu (w nawiązaniu do [17]),
- pogłos z zastosowaniem różnicy amplitudy lewego i prawego kanału dla linii opóźniających – w celu symulacji zjawiska międzyusznej różnicy poziomu (w nawiązaniu do [17]),
- pogłos z zastosowaniem różnicy pomiędzy filtracją linii opóźniających lewego i prawego kanału (w nawiązaniu do [19]).

Jak opisano w podrozdziale 2.3, stopień wrażenia przestrzenności związany jest ze stopniem niekoherencji sygnałów z obu kanałów. Dlatego też, w przypadku chęci spowodowania u słuchacza wrażenia przestrzenności, pożądane jest, aby koherencja sygnałów z obu kanałów była jak najmniejsza. Równie pożądana jest jednak wysoka widmowa gęstość mocy. Przyjęto założenie, iż im bardziej wartości koherencji są skupione wokół średniej tym lepiej. Podobnie, jak odchylenie od wartości średniej widmowej gęstości mocy, która pożądana jest na stałym poziomie. Założenia opisano zdefiniowanym na potrzeby pracy parametrem *Pr*. Zdefiniowanie parametru jest odpowiedzią na powtarzające się w literaturze podkreślanie istotności cech sygnału wpływające na jakość pogłosu [9, 10, 13] oraz jego przestrzenność [1, 19]. Cechy te to: wysoka gęstość widma oraz stopień niekoherencji sygnałów z obu kanałów. Zdefiniowany przez autora parametr *Pr* przedstawia się jako:

$$Pr := \frac{E(\emptyset_x)}{E(C_{x,LR}) * \sigma(\emptyset_x) * \sigma(C_{x,LR})} \tag{3.7}$$

gdzie:

σ – średnie odchylenie standardowe,

Fi – widmowa gęstość mocy sygnału,

$C_{x,LR}$ – koherencja lewego i prawego kanału sygnału,

E – średnia arytmetyczna.

Do wyznaczenia parametrów: koherencja, widmowa gęstość mocy oraz płaskość widma zostały wykorzystane funkcje z biblioteki *scipy* w języku Python, odpowiednio: *coherence* [27], *spectral_flatness* [28], *welch* [29].

3.2.2. Wyniki porównań wersji pogłosów oraz interpretacja wyników

Zaimplementowane wersje pogłosu porównywane są na podstawie wartości parametrów zdefiniowanych w ramach pracy. W podrozdziale tym przedstawione są także wykresy płaskości widma od czasu oraz, w przypadku badania przestrzenności sygnałów, wykresy koherencji między lewym i prawym kanałem od częstotliwości. Implementacje opisanych w podrozdziale 3.2.1 wersji pogłosu zostały zrealizowane korzystając z frameworku JUCE w języku C++ – dokładny opis implementacji znajduje się w rozdziale 4.

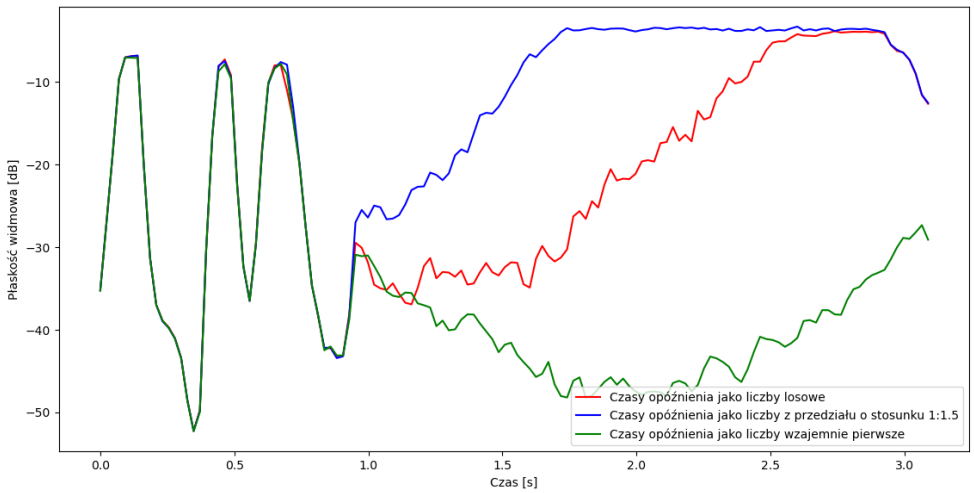
Wszystkie sygnały wykorzystane do badań zawierają fragment przed zakończeniem czasu trwania dźwięku bezpośredniego (widoczne jest to na wykresach ze zmienną czas na osi poziomej). Po jego zakończeniu parametry wyraźnie zaczynają się rozbiegać, tam ukazuje się ich odmienność. Podczas trwania sygnału przed wyłączeniem źródła wykresy pokrywają się, jednak niewielkie różnice między nimi istnieją, co wynika z faktu, iż podczas jego trwania sygnał posiada już w sobie sygnał pogłosowy.

Wszystkie badane sygnały różnią się od siebie w największym stopniu (z wyjątkiem badania dot. przestrzenności) parametrem płaskość widmowa. Dlatego też parametr ten ma największy wpływ na wartości parametrów zdefiniowanych w ramach pracy. W niniejszym podrozdziale przedstawiane są także przebiegi płaskości widmowej od czasu. Graficznej prezentacji tego parametru towarzyszy interpretacja otrzymanych wartości.

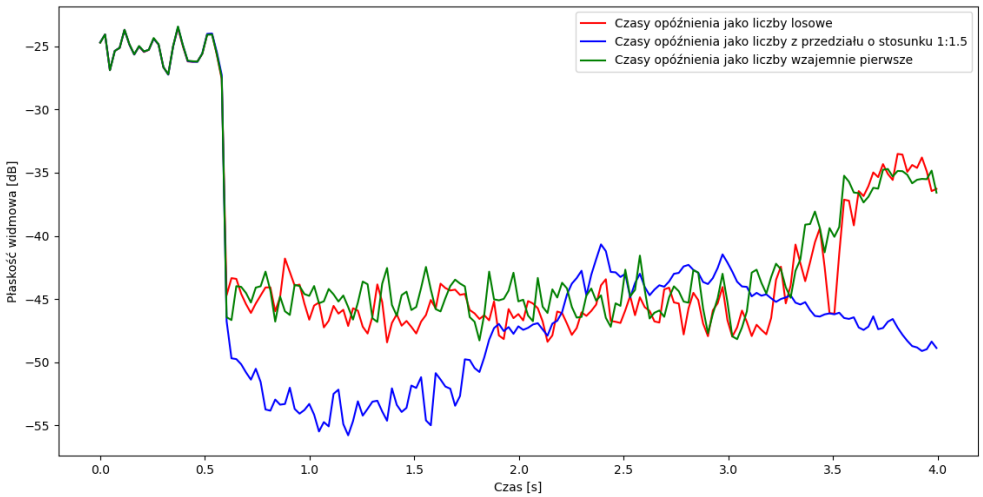
Wartości liczbowe wyznaczonych parametrów B (wzór 3.6) oraz Pr (wzór 3.7) będące wyznacznikami jakości pogłosu zostały znormalizowane do wartości 1 w obrębie sygnału. W ramach normalizacji wartość 1 przyjmuje wersja sygnału, która uzyskała najwyższą wartość parametru.

A) Dobór czasów opóźnienia:

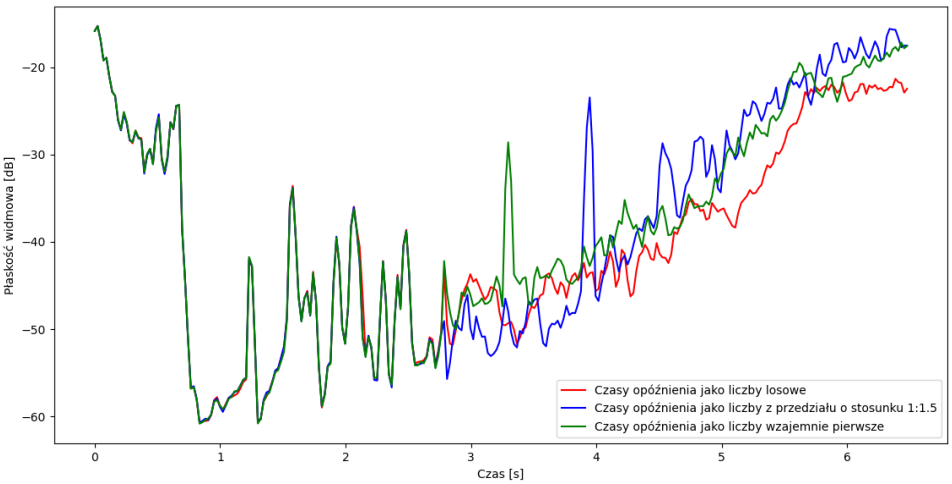
Płaskość widmowa sygnału mowy w wersji pogłosu z liniami opóźniającymi o czasach opóźnienia będących liczbami zawierającymi się z zakresie 1:1,5 przez większość czasu trwania sygnału jest większa niż tego parametru dla innych wersji (Rys. 3.1). Parametr ten ma bezpośredni wpływ na wartość parametru *B* będący wyznacznikiem jakości pogłosu. W przypadku tym, nawet bez stosowania jednoliczbowych parametrów można stwierdzić (na podstawie samego wykresu), która wersja sygnału najbardziej odpowiada oczekiwaniom. Parametry te są natomiast szczególnie przydatne w nieoczywistych sytuacjach, przykładowo w przypadku sygnału MLS lub gitary klasycznej (Rys. 3.2, 3.3), gdzie podczas trwania sygnału największą płaskość widmową przyjmuje każda z wersji sygnału, w zależności od momentu w czasie trwania. Wartości parametru dla czasów linii opóźniających przedstawiono w Tab. 3.1.



Rys. 3.1. Płaskość widmowa dla różnego rodzaju generowania czasów linii opóźniających – sygnał mowy.



Rys. 3.2. Płaskość widmowa dla różnego rodzaju generowania czasów linii opóźniających – sygnał MLS.



Rys. 3.3. Płaskość widmowa dla różnego rodzaju generowania czasów linii opóźniających – gitara klasyczna.

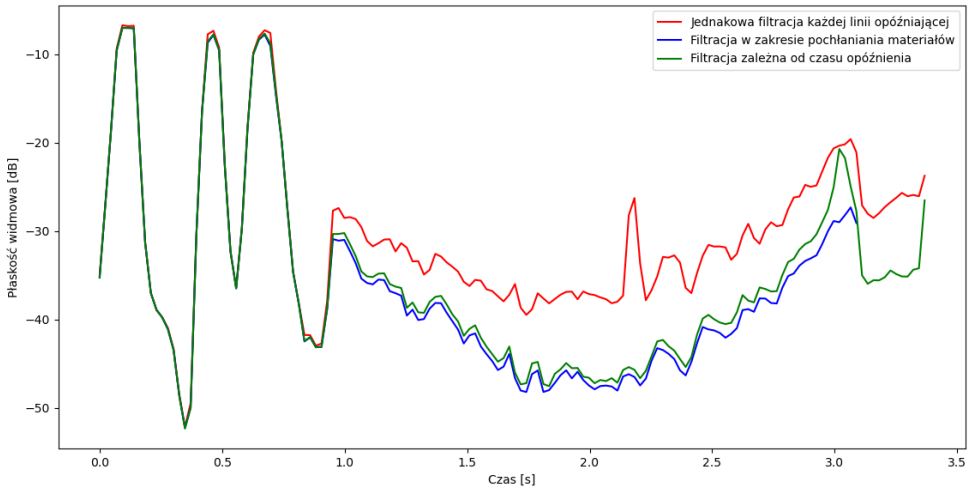
Tab. 3.1. Wartości parametru *B* dla różnego rodzaju generowania czasów linii opóźniających.

	Czasy opóźnienia		
	1. Losowe	**2. Jako liczby pierwsze**	**3. Jako liczby z zakresu o stosunku 1:1,5**
Sygnał mowy	0,57	0,22	1,0
MLS	0,64	1,0	0,39
Gitara klasyczna	0,80	1,0	0,98

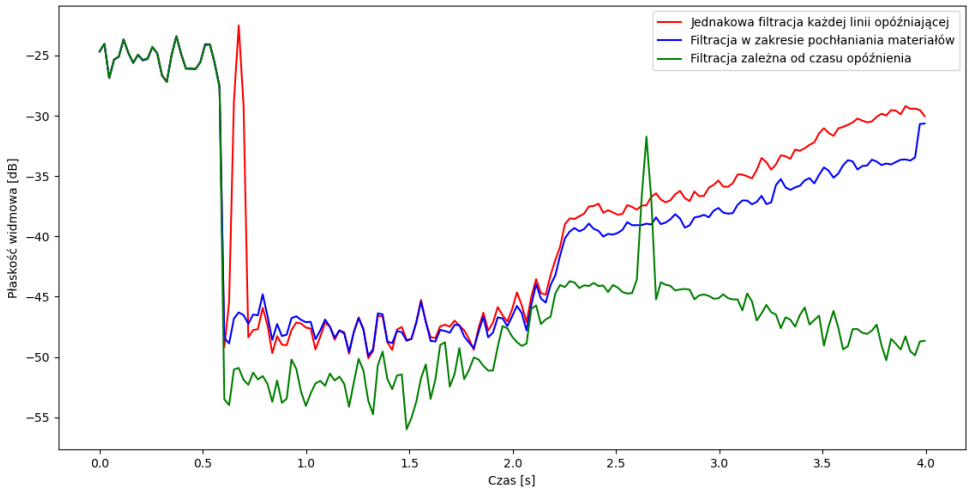
Zdefiniowany parametr *B* dla dwóch z trzech badanych sygnałów okazał się najwyższy dla wariantu pogłosu nr 2 (Tab. 3.1). Opcja nr 3 okazała się najlepszą wersją dla sygnału mowy, a nagranie gitary klasycznej osiągnęło również niemal najlepszy wynik. Wersja ta (nr 3) natomiast ograniczona jest wąskim zakresem czasów opóźnienia – w przypadku chęci użycia maksymalnego czasu opóźnienia jako przykładowo 1200 ms, najniższy czas opóźnienia wynosiłby w takim przypadku 800 ms. Ze względu na najwyższą wartość parametru dla dwóch sygnałów w wersji nr 2, oraz na to ograniczenie zakresu czasów opóźnienia, zdecydowano, iż najlepszym wyborem jest opcja z czasami opóźnienia będącymi liczbami pierwszymi.

B) Filtracja linii opóźniających:

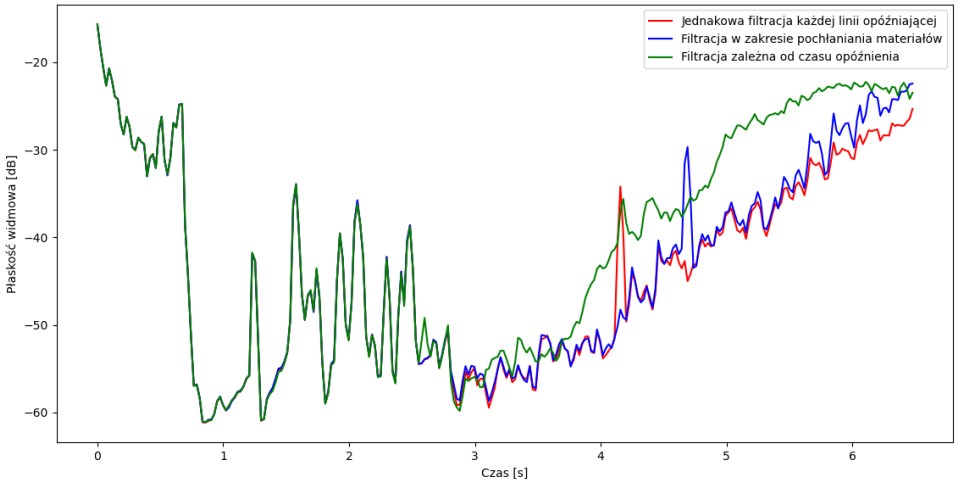
Założenia dotyczące pożądanych cech sygnału przyjęte przy definiowaniu parametru *B* (podrozdział 3.2.1) potwierdzają przebiegi czasowe płaskości widmowej, na których można zauważyć, iż we wszystkich przypadkach wersja sygnału, której płaskość widmowa przez większość czasu trwania sygnału jest największa (Rys. 3.4–3.6), przyjmuje największą wartość parametru *B* (Tab. 3.2).



Rys. 3.4. Płaskość widmowa symulacji różnych wariantów filtracji – sygnał mowy.



Rys. 3.5. Płaskość widmowa symulacji różnych wariantów filtracji – sygnał MLS.



Rys. 3.6. Płaskość widmowa symulacji różnych wariantów filtracji – gitara klasyczna.

Tab. 3.2. Wartości parametru B dla pogłosów z różnymi wariantami filtracji.

	Filtracja		
	1. Każdej linii w jednakowy sposób	**2. W zakresie pochłaniania materiałów**	**3. Tym niższa częstotliwość odcięcia filtru, im wyższy czas opóźnienia**
Sygnał mowy	1,0	0,41	0,07
MLS	1,0	0,79	0,11
Gitara klasyczna	0,47	0,92	1,0

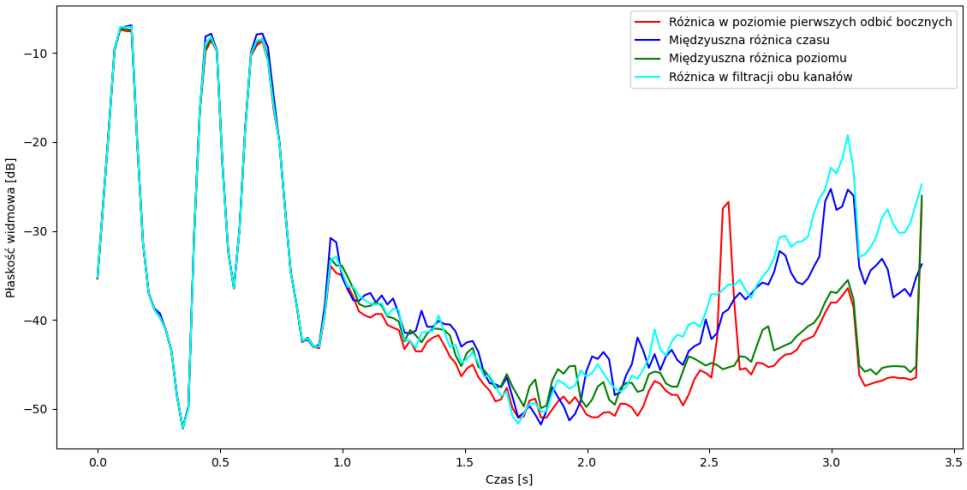
Dla dwóch sygnałów (sygnał mowy oraz MLS) wartość zdefiniowanego parametru przyjmuje największe wartości dla pierwszej wersji sygnału (Tab. 3.2) (sygnał z filtracją każdej linii opóźniającej jednakowym filtrem). Pokrywa się to z założeniem, iż im płaskość widmowa sygnału większa, tym wartość parametru wyższa (jednakowa filtracja każdej linii opóźniającej skutkuje podobną zawartością widmową w każdym momencie czasu trwania sygnału). Natomiast nie odzwierciedla to teorii dotyczącej zjawiska rzeczywistego pogłosu akustycznego, mówiącej, iż odbicia, które dochodzą do słuchacza później, są filtrowane mocniej (oprócz filtracji przy odbiciu od

przeszkody – również przez ośrodek. Fala akustyczna przebiega dłuższą drogę, dlatego jest mocniej filtrowana przez powietrze).

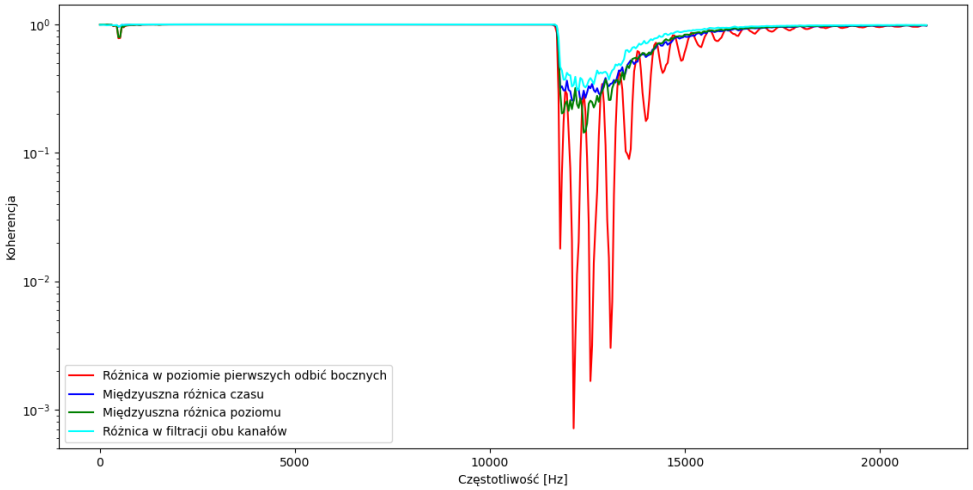
Zgodnie ze wskazaniami parametru *B*, wersja zastosowana do późniejszej docelowej implementacji to wersja nr 1 (Tab. 3.2). Natomiast, ze względu na fakt, iż wersja pogłosu z filtracją zależną od czasu opóźnienia silnie wiąże się z teorią, a więc być może z lepszymi wrażeniami słuchowymi, filtracja ta zostanie zastosowana w docelowej implementacji i dodana do interfejsu użytkownika jako wersja opcjonalna do wyboru (w celu samodzielnej oceny i subiektywnego wyboru, preferencje wyboru mogą również zależeć od typu sygnału, szczególnie, iż dla nagrania gitary klasycznej wersja ta uzyskała najlepszy wynik).

C) Zjawiska przestrzenne:

Wyznacznikiem przestrzenności wersji pogłosu w tym podrozdziale będzie stworzony w tym celu parametr *Pr* (podrozdział 3.2.1). Płaskość widmowa przy badaniu przestrzenności nie ma wpływu na zdefiniowany parametr, natomiast wykresy tej zmiennej od czasu zostały przedstawione poglądowo, aby można było zaobserwować na nich, iż w większości przypadków różnice między metodami nie miałyby dużego wpływu na ten parametr. W sposób graficzny przedstawiono także koherencję pomiędzy lewym i prawym kanałem od częstotliwości, dla wszystkich metod i sygnałów.

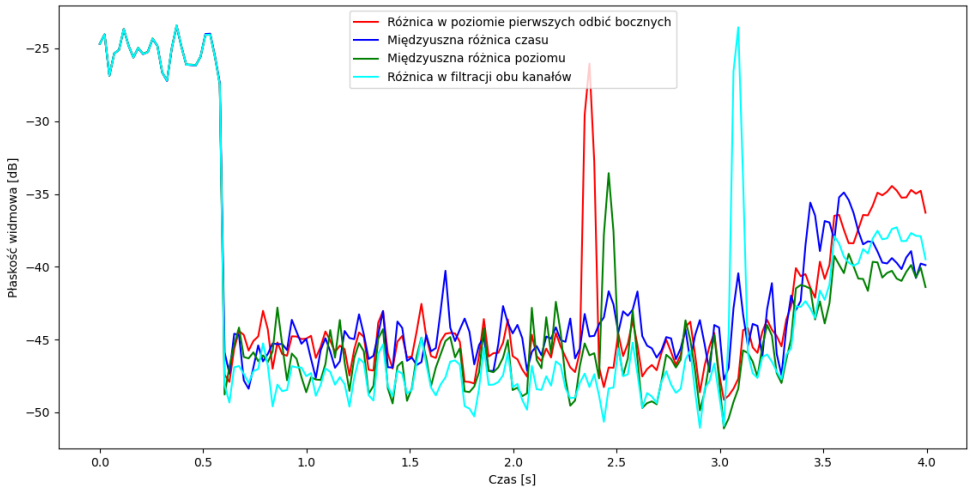


Rys. 3.7. Płaskość widmowa dla różnych zjawisk przestrzennych – sygnał mowy.

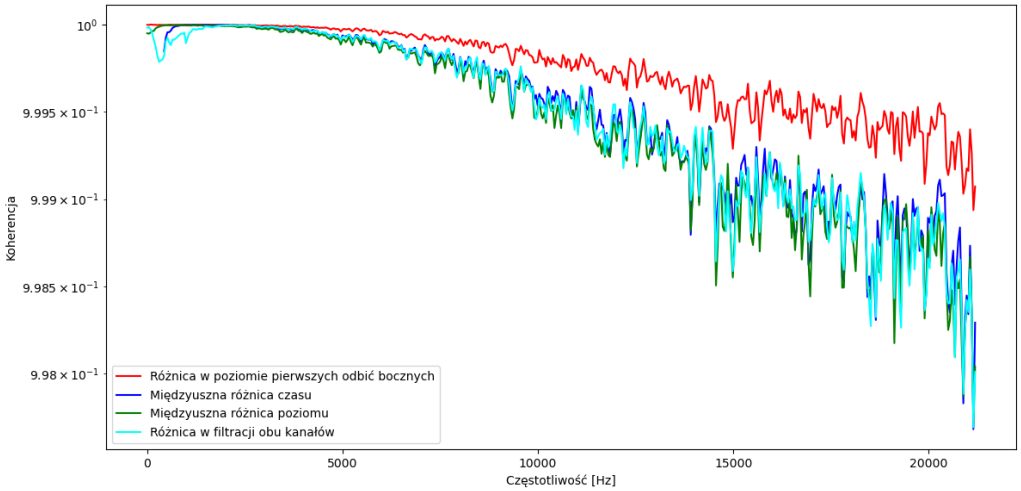


Rys. 3.8. Koherencja między kanałami dla różnych zjawisk przestrzennych – sygnał mowy.

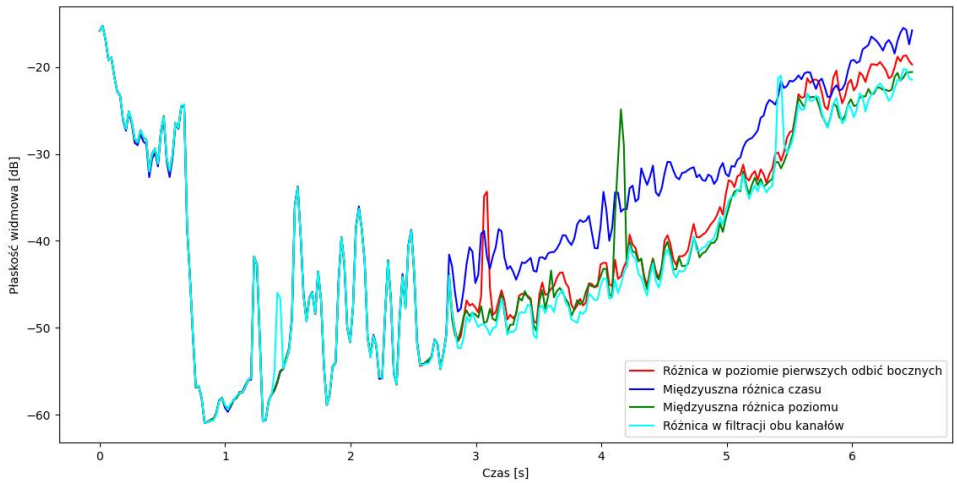
Na wartości liczbowe parametru *Pr* największy wpływ mają funkcje koherencji między lewym i prawym kanałem od częstotliwości. W przypadku sygnału mowy funkcja koherencji między kanałami od częstotliwości dla wersji pogłosu z różnicą w amplitudzie pierwszych odbić bocznych (Rys. 3.8, kolor czerwony) posiada przebieg odbiegający od pozostałych. Można łatwo zaobserwować, iż jego wartość średnia jest najmniejsza, co jest właściwością pożądaną, aczkolwiek jego odchylenie standardowe jest największe. Dlatego ostatecznie wartość parametru *Pr* dla tej wersji sygnału mowy przyjmuje najmniejszą wartość. Pozostałe wersje tego sygnału przyjmują wartości zbliżone do najwyższej w obrębie tego sygnału (Tab. 3.3).



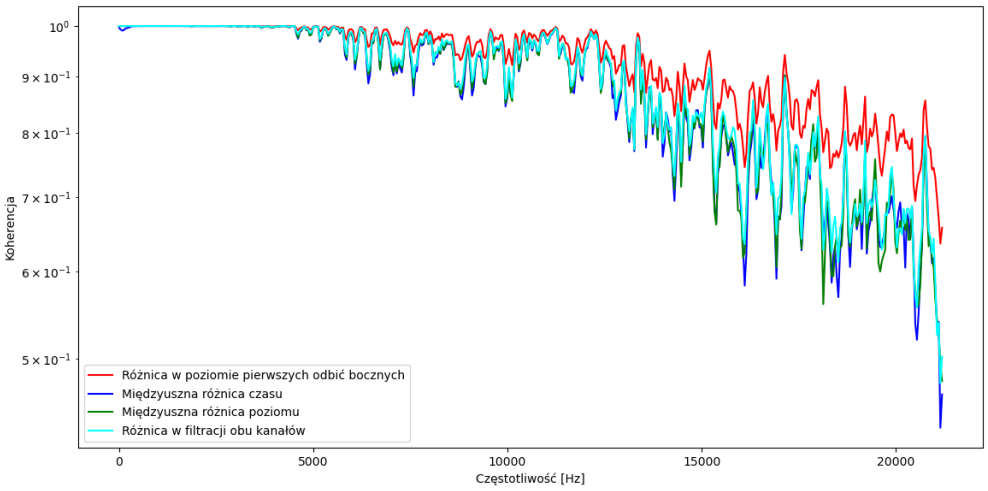
Rys. 3.9. Płaskość widmowa dla różnych zjawisk przestrzennych – sygnał MLS.



Rys. 3.10. Koherencja między kanałami dla różnych zjawisk przestrzennych – sygnał MLS.



Rys. 3.11. Płaskość widmowa dla różnych zjawisk przestrzennych – gitara klasyczna.



Rys. 3.12 Koherencja między kanałami dla różnych zjawisk przestrzennych – gitara klasyczna.

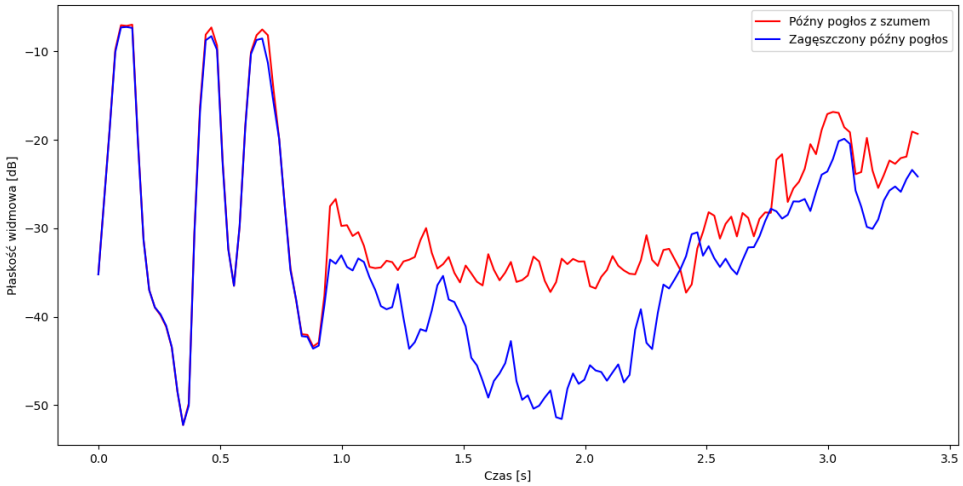
Tab. 3.3. Wartości parametru *Pr* dla symulacji różnych zjawisk przestrzennych.

	1. ITD	**2. Różna filtracja obu kanałów**	**3. ILD**	**4. Różnica w amplitudzie pierwszych odbić bocznych**
Sygnał mowy	0,88	1,0	0,83	0,68
MLS	0,51	0,48	0,47	1,0
Gitara klasyczna	0,67	0,69	0,66	1,0

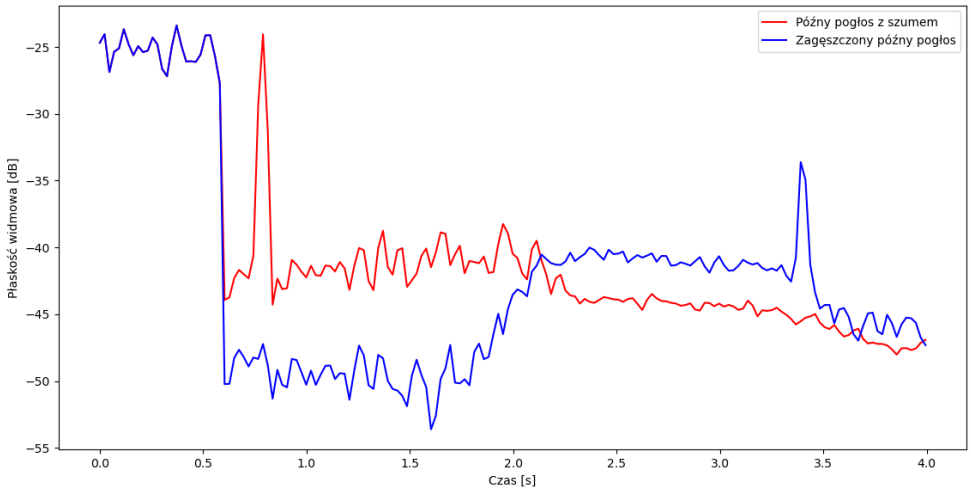
W przypadku dwóch pozostałych sygnałów (MLS oraz nagranie gitary klasycznej), wersją, której wartości koherencji odstają od pozostałych jest również wersja nr 4 (Tab. 3.3). W tym przypadku jednak działa to na korzyść tej wersji, według przyjętych założeń implementacja pogłosu z różnicą w amplitudzie pierwszych odbić bocznych ma największy wpływ na przestrzenność pogłosu. Pomimo wysokiej wartości średniej koherencji między kanałami (której pożądane są małe wartości) sygnałów MLS oraz nagrania gitary klasycznej, (Rys. 3.10, 3.12), parametr *Pr* przyjął dla nich największą wartość. Ze względu na rozbieżność w wynikach parametru dla różnych sygnałów, w tym wypadku również, oprócz wersji nr 4, do interfejsu użytkownika dodane zostaną opcje włączenia i wyłączenia symulacji innych zjawisk przestrzennych.

D) Późny ogon pogłosowy:

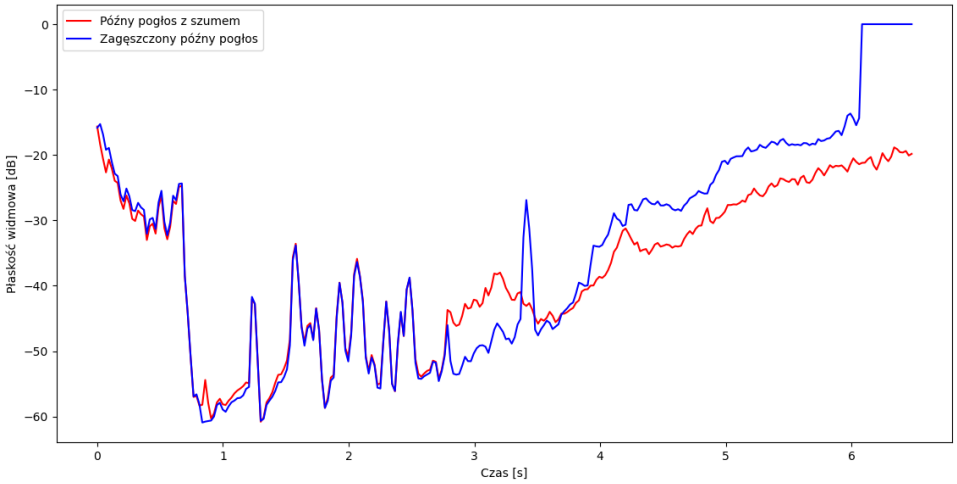
W podpunkcie tym porównywane są dwie wersje pogłosu, zgodnie z opisem w podrozdziale 3.2.1. Wyznacznikiem jakości pogłosu w tym przypadku ponownie jest parametr *B*.



Rys 3.13. Płaskość widmowa dla obu wersji implementacji późnego pogłosu – sygnał mowy.



Rys 3.14. Płaskość widmowa dla obu wersji implementacji późnego pogłosu – sygnał MLS.



Rys 3.15. Płaskość widmowa dla obu wersji implementacji późnego pogłosu – gitara klasyczna.

Tab. 3.4. Wartości parametru B dla pogłosów z różnymi wariantami generowania ogona pogłosowego.

	1. Późny pogłos z szumem	**2. Późny pogłos zagęszczony odbiciami**
Sygnał mowy	1,0	0,086
MLS	0,98	1,0
Gitara klasyczna	0,84	1,0

Zgodnie z definicją płaskości widmowej (wzór 3.4) przyjmuje ona tym wyższą wartość, im bardziej sygnał zbliżony jest do sygnału szumowego. Tak więc na wartości parametru *B* w przypadku sygnału *MLS* mógł mieć wpływ szumowy charakter tego sygnału. Natomiast obie wersje pogłosu w tym podpunkcie wykazały zbliżone, wysokie wartości zdefiniowanego parametru, z wyjątkiem wersji pogłosu nr 2 (Tab. 3.4) dla sygnału mowy. Odchylenie to jest stosunkowo duże, różni się od pozostałych o rząd wielkości. Ze względu na ten fakt, wersją implementowaną w docelowym pogłosie będzie późny pogłos mnożony z szumem białym.

4. Implementacja docelowego pogłosu

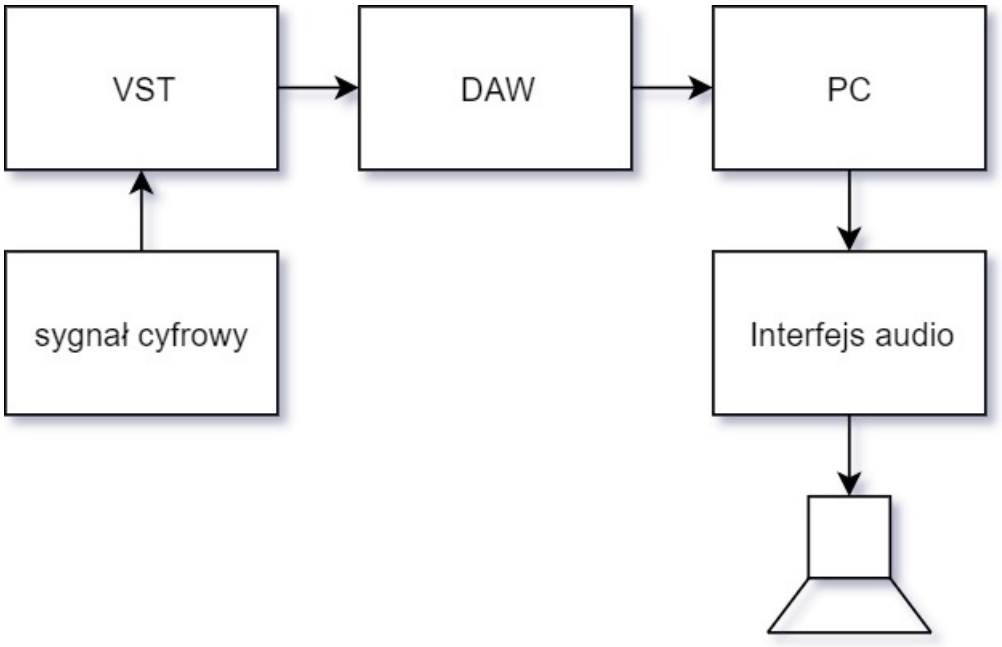
Jak wspomniano we Wstępie (1), możliwości obliczeniowe w obecnych czasach znacznie ułatwiają badania nad realizacją sztucznego pogłosu w stosunku do warunków w XX wieku. Realizacja zjawisk akustycznych w domenie cyfrowej wymaga jednak konieczności znajomości technik przetwarzania sygnałów cyfrowych (*Digital Signal Processing*) oraz języków programowania, a także sposobów implementacyjnych zapewniających wydajność obliczeniową, szczególnie w przypadku konieczności działania algorytmu w czasie rzeczywistym. Istnieje jednak coraz więcej gotowych narzędzi umożliwiających zaawansowane operacje związane z DSP.

Zarówno testowe implementacje elementów pogłosu (rozdział 3), jak i docelowej aplikacji zostały zrealizowane z wykorzystaniem frameworku[1] JUCE. Jest to wieloplatformowy otwarto–źródłowy framework pozwalający na realizację aplikacji m. in. w technologii VST w języku C++ [31]. Narzędzie to pozwala także na realizację interfejsu użytkownika. Połączenie tych funkcji daje efektywne narzędzie do tworzenia aplikacji związanych z dźwiękiem. Wszystkie klasy oraz funkcje występujące w opisie implementacji w tym rozdziale są częścią frameworku JUCE.

4.1. Koncepcja realizacji

Projekt zakłada stworzenie aplikacji w formacie VST pozwalającej na modyfikację sygnału przypisanego do ścieżki w oprogramowaniu typu DAW (*Digital Audio Workstation*). Modyfikacja ta to sztuczny pogłos w postaci efektu będącego wynikiem implementacji opracowanego w ramach niniejszej pracy algorytmu. Stworzona wtyczka VST posłuży jako narzędzie mogące być wykorzystane w celu nałożenia efektu na sygnał cyfrowy (Rys. 4.1).

[1] framework – platforma umożliwiająca tworzenie oprogramowania [30]
[2] na komputerze działającym pod kontrolą systemu Windows 10, wyposażonym w czterowątkowy, dwurdzeniowy procesor Intel Core i5-3230M o częstotliwości taktowania zegara 2,6 GHz RAM o

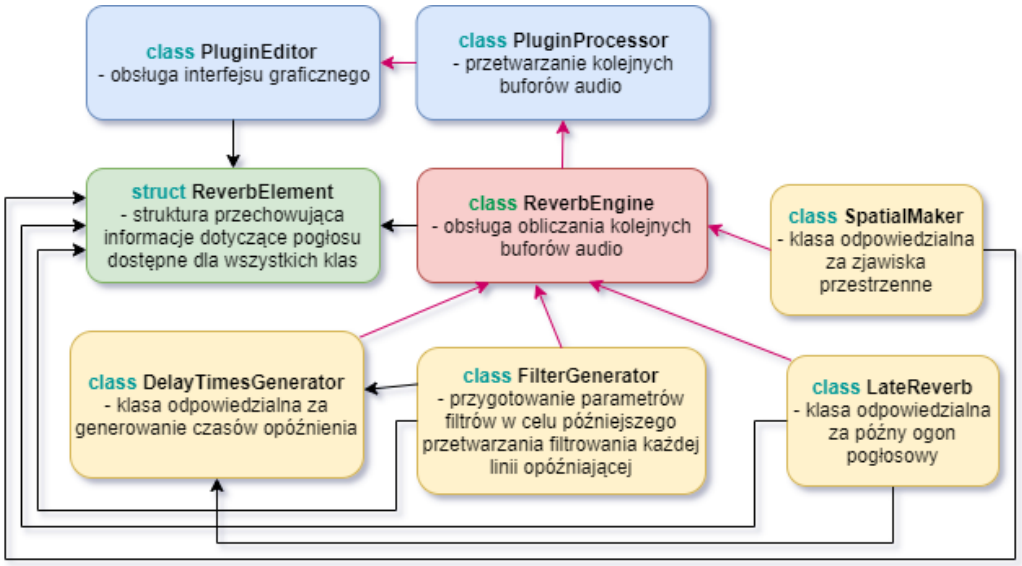


Rys. 4.1. Schemat toru przetwarzania sygnału.

Implementacja docelowego pogłosu zakłada uwzględnienie wyników badań opisanych w rozdziale 3 i na ich podstawie zastosowanie wybranych elementów w celu uzyskania hybrydowego rozwiązania.

4.2. Architektura aplikacji

W celu obsługi wszystkich elementów składających się na wynikowy pogłos, przy jednoczesnym zachowaniu czytelności kodu, stworzono szereg klas i struktur, z których każda odpowiada za inną część obliczeń realizujących przetworzenie sygnału w celu uzyskania sygnału wynikowego.



Rys. 4.2. Schemat przedstawiający hierarchię oraz opis klas w programie.

Na Rys. 4.2. przedstawiony jest schemat hierarchii klas w programie. Klasy oznaczone kolorem niebieskim są klasami bibliotecznymi, których istnienie jest niezbędne do prawidłowej kompilacji aplikacji. Reszta klas została stworzona na potrzeby pracy i w nich odbywają się wszystkie obliczenia prowadzące przygotowania bufora wyjściowego. Strzałki czarne oznaczają zastosowany mechanizm dziedziczenia, dzięki któremu klasy odpowiedzialne za obliczenia związane ze wszystkimi elementami pogłosu mają dostęp do wspólnych danych potrzebnych każdej z dziedziczących klas. Pozostałe strzałki oznaczają, iż polem klasy, do której skierowana jest strzałka jest obiekt klasy, z której strzałka wychodzi. Rys 4.3. przedstawia zastosowanie tego mechanizmu na przykładzie klasy *ReverbEngine*.

```
class ReverbEngine: public ReverbElement
{
public:
    ReverbEngine() {}

    FilterGenerator filterGenerator;
    SpatialMaker spatialMaker;
    DelayTimesGenerator delayTimes;
    LateReverb lateReverb;

    /*...*/
```

Rys. 4.3. Definicje zmiennych o typach stworzonych klas, w głównej klasie odpowiedzialnej za obliczenia dotyczące sygnału.

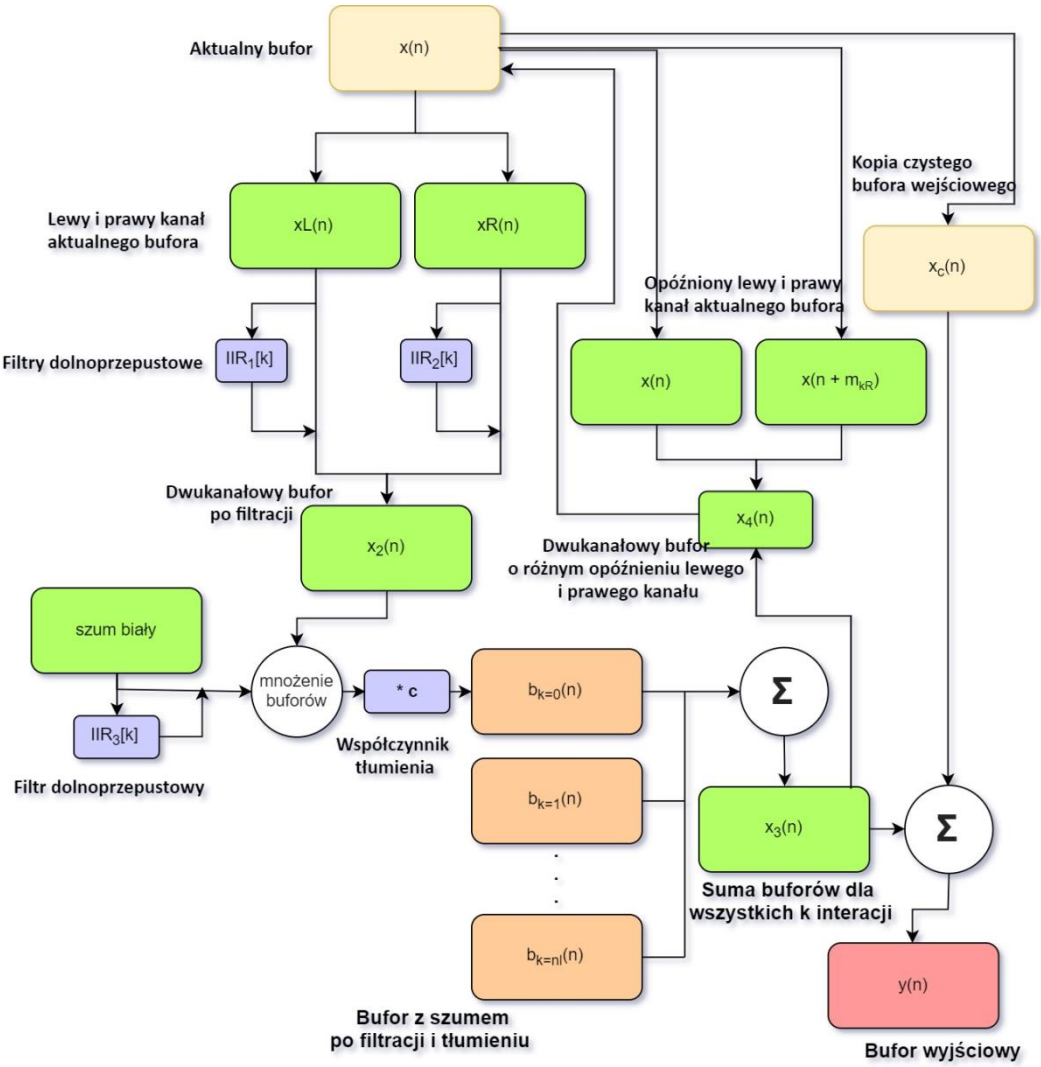
4.3. Połączenie wybranych rozwiązań w docelowy pogłos

Przetwarzanie sygnału w trakcie działania programu wiąże się z wymaganiem dobrej optymalizacji ze względu na czas obliczeń. Dlatego też, duża część obliczeń prowadzących do uzyskania wyjściowego sygnału wykonywana jest przed rozpoczęciem przetwarzania kolejnych buforów. Część danych liczbowych niezbędnych do uzyskania finalnego efektu jest predefiniowana i nie wpływa na szybkość obliczeń, a tym samym nie ma wpływu na potencjalne opóźnienia w czasie działania programu. Tak więc obliczenia w klasach odpowiedzialnych za m. in. przygotowanie tablicy czasów opóźnienia linii opóźniających, działania filtrów i zjawisk przestrzennych wykonywane są przed rozpoczęciem wykonywania obliczeń związanych z przetwarzaniem aktualnego bufora audio. Większość klas posiada metodę *prepare*, realizującą te jednorazowe obliczenia. Metody te wywoływane są z kolei w metodzie *prepare* klasy *ReverbEngine*, która wywoływana jest również jednorazowo (Rys. 4.4). Następnie informacje te przesyłane są do klasy bibliotecznej odpowiedzialnej za właściwe przetwarzanie sygnału (klasa *PluginProcessor*).

```
void ReverbEngine::prepare(double sampleRate, int samplesPerBlock, int numChannels)
{
    sampleRate_ = sampleRate;

    filterGenerator.prepare(sampleRate, samplesPerBlock, numChannels);
    spatialMaker.prepare();

    /*...*/
```

Rys. 4.4. Przygotowanie danych na przykładzie klas *filterGenerator* i *spatialMaker*.

W niniejszym podrozdziale znajduje się szczegółowy opis działań prowadzących do uzyskania sygnału wyjściowego. Ogólny schemat działań w czasie przetwarzania jednego bufora audio znajduje się na Rys. 4.5.



Rys. 4.5. Schemat przetwarzania sygnału w czasie trwania jednego bufora.

4.3.1. Linie opóźniające

A) Wariant pogłosu złożonego z linii opóźniających o losowych czasach opóźnienia:

Wariant ten wymagał użycia generatora liczb losowych, w celu utworzenia tablicy losowych liczb z zakresu pożądanych wartości opóźnienia. Operacja losowania wartości czasów opóźnienia należy do działań wykonywanych przed rozpoczęciem przetwarzania sygnału. Gdyby wartości te losowane były w czasie przetwarzania sygnału, mielibyśmy do czynienia z dużą ilością dodatkowych obliczeń w czasie przetwarzania jednego bufora. Przykładowo, w przypadku bufora o wielkości 512 próbek i częstotliwości próbkowania 44100 próbek na sekundę, czas przetwarzania takiego bufora wynosi ~12 milisekund. W tak krótkim czasie istniałaby potrzeba

wylosowania ok. 100 (w zależności od liczby linii opóźniających) nowych liczb, co skutkowałoby znacznie większą złożonością obliczeniową. Co więcej, podejście takie skutkuje niepożądanymi artefaktami i wymaga zastosowania dodatkowych obliczeń w celu ich wyeliminowania. Dlatego też pojawia się potrzeba predefinicji tych wartości.

Do wygenerowania liczb losowych została wykorzystana klasa biblioteczna *Random,* która realizuje generator liczb losowych oraz posiada szereg metod pozwalających na uzyskanie losowej wartości określonego typu liczbowego z zadanego zakresu [32]. Do wylosowania wartości czasów opóźnienia, wykorzystane zostały liczby z dziedziny liczb całkowitych reprezentujących milisekundy. Wartości minimalna i maksymalna zakresu losowania określone są przez pożądany zakres czasowy minimalnej i maksymalnej wartości opóźnienia kopii sygnału. Zakres liczb tych określony został za pomocą klasy *JUCE::Range<int>* [33]. Klasa ta jest przydatna, ponieważ dane w postaci zmiennej tego typu mogą być wykorzystane przez metodę klasy *Random,* zwracającą liczbę z zadanego zakresu. Operacje losowania, jak również tworzenia wektora czasów opóźnienia realizowane są przez stworzoną na potrzeby pracy klasę *DelayTimesGenerator*.

B) Wariant pogłosu złożonego z linii opóźniających o czasach opóźnienia będących liczbami pierwszymi:

W tym wariancie, do stworzenia wektora czasów opóźnienia niezbędna jest znajomość szeregu liczb pierwszych w zakresie pożądanych minimalnej i maksymalnej wartości czasu opóźnienia. W celu uzyskania jak największej gęstości pogłosu najlepiej, aby szereg zawierał wszystkie liczby pierwsze z zadanego zakresu. Mając wektor liczb pierwszych o odpowiedniej długości możliwe jest stworzenie wektora czasów opóźnienia. Długość tego wektora ograniczona jest przez maksymalną liczbę linii opóźniających, która z kolei ograniczona jest przez możliwości obliczeniowe pozwalające aplikacji na płynne przetwarzanie sygnału bez opóźnień i przerw w przetwarzaniu sygnału wynikających ze zbyt wielu obliczeń w jednostce czasu. Przy testowaniu aplikacji[2] znaleziona maksymalna liczba linii opóźniających pozwalająca na płynne przetwarzanie to ~150. Liczba ta zależy również od obecności innych obliczeń

[2] na komputerze działającym pod kontrolą systemu Windows 10, wyposażonym w czterowątkowy, dwurdzeniowy procesor Intel Core i5-3230M o częstotliwości taktowania zegara 2,6 GHz RAM o pojemności 8 GB

towarzyszących uzyskaniu finalnego efektu, takich jak: obliczenia związane z filtracją, uzyskaniem efektów przestrzennych oraz późnym pogłosem (podrozdziały 4.3.4 – 4.3.6). Największy wpływ na szybkość obliczeń, a tym samym na maksymalną możliwą do zastosowania liczbę linii opóźniających, ma zastosowanie filtrowanego szumu białego przy tworzeniu późnego ogona pogłosowego (podrozdział 4.3.5). W przypadku zastosowania tego ostatniego, maksymalna liczba linii nie powodująca negatywnych skutków to ~80. W przypadku ograniczenia do tej liczby, aby uzyskać odpowiednio wysoki maksymalny czas opóźnienia, niezbędne jest pomijanie pewnych liczb pierwszych z posortowanego wektora wszystkich liczb pierwszych (z zadanego zakresu) tak, aby pogłos był odpowiednio długi. Przykładowo, przy liczbie linii opóźniających ograniczonej do wspomnianych 80 i jednoczesnej chęci uzyskania odpowiednio długiego pogłosu, aby uzyskać maksymalny czas opóźnienia 1500 ms, konieczne jest pomijanie 2/3 wszystkich liczb, a więc branie jedynie co trzeciej wartości ze wszystkich liczb pierwszych z zadanego zakresu.

4.3.2. Sposób uzyskania linii opóźniających ze sprzężeniem

A) Utworzenie sygnału sprzężonego z jego opóźnioną kopią.

Realizowane podejście wymaga stworzenia dodatkowego bufora audio, który zapewniać będzie dodatkowy nośnik informacji niezbędny do wykonywania obliczeń wykorzystujących kopie pierwotnego sygnału – kopie te są głównym składnikiem pogłosu. W celu stworzenia takiego bufora (bufor ten będzie nazywany opóźniającym) skorzystano z klasy *AudioBuffer,* realizującej wielokanałowy bufor audio złożony z liczb zmiennoprzecinkowych reprezentujących próbki sygnału [34]. W obrębie klasy istnieją metody pozwalające na dodawanie i kopiowanie buforów, ustawianie wzmocnień oraz ustalanie wartości konkretnych próbek bufora, metody te okażą się przydatne w realizacji dalszych części algorytmu. Zastosowanie tej klasy pozwala również na wykorzystanie danych w tej postaci przez inne klasy wbudowane frameworku.

Aby uzyskać sygnał będący sumą sygnału pierwotnego z jego opóźnioną kopią, w pierwszej kolejności stworzono kopię sygnału pierwotnego o długości odpowiadającej długości bufora opóźniającego. Długość bufora opóźniającego została

ustalona na liczbę próbek odpowiadającą 4 sekundom (jest zależna od częstotliwości próbkowania). Następnie za pomocą funkcji bibliotecznej A*udioBuffer::copyFrom()* wykonane zostaje przekopiowanie bufora przygotowanego pierwotnie do wysłania na wyjście (zakładając brak przetwarzania – bufor sygnału oryginalnego) do bufora opóźniającego. Funkcja ta powoduje stworzenie kopii określonego fragmentu bufora. Długość fragmentu bufora, dokładne miejsce wklejenia, numer kanału oraz źródło kopiowania określane są w argumentach ww. funkcji. Podczas przetwarzania kolejnego bufora, zostaje on wklejony próbkę dalej w stosunku do ostatniej próbki bufora przetwarzanego przed nim.

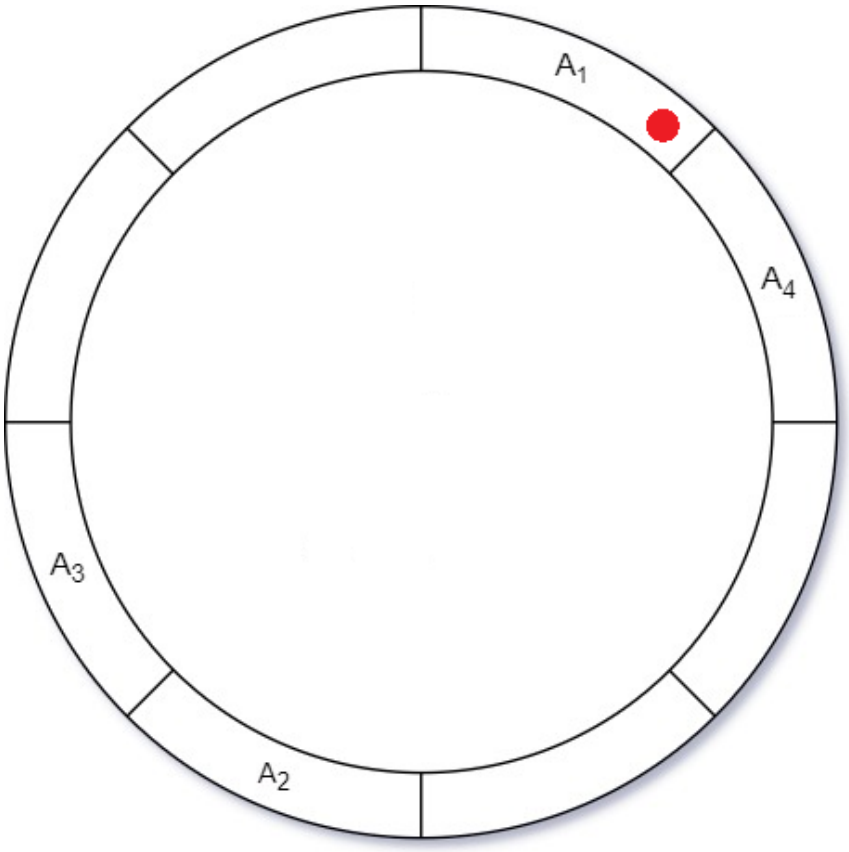
```
const float* delayBufferDataL = delayBuffer.getReadPointer(leftChannel);
const float* delayBufferDataR = delayBuffer.getReadPointer(rightChannel);

const float* bufferDataLa = buffer.getReadPointer(leftChannel);
const float* bufferDataRa = buffer.getReadPointer(rightChannel);
```

Rys. 4.6. Definicja zmiennych zawierających informacje o miejscu buforów w pamięci.

Wykonywanie operacji na buforach wymaga definicji zmiennych wskaźnikowych wskazujących na pierwsze elementy buforów aktualnego i opóźniającego dla obu ich kanałów (Rys. 4.6). Posiadanie informacji tych jest konieczne do poprawnego kopiowania i dodawania do siebie buforów, a stosowanie tych operacji jest niezbędne do uzyskania finalnego efektu.

Przy wypełnianiu bufora opóźniającego, zastosowano strukturę danych stosowaną szeroko szczególnie przy potrzebie przetwarzania strumienia danych – bufor kołowy [35]. W celu wizualizacji działania tej struktury danych, sytuacja implementacji z niniejszej pracy została sprowadzona do prostego przypadku (Rys 4.7). W tej prostej sytuacji bufor opóźniający ma 8 próbek, a bufor wejściowy sygnału 5 próbek. W momencie kopiowania pierwszego bufora do bufora opóźniającego, bufor opóźniający zostaje wypełniony liczbą próbek odpowiadającej długości bufora wejściowego. W przypadku rozpoczęcia od pierwszego elementu bufora opóźniającego, koniec bufora wejściowego nie wyjdzie poza zakres długości bufora opóźniającego (element A_1 – początek bufora wejściowego, A_2 – koniec bufora wejściowego, czerwony punkt – początek bufora opóźniającego). Następny bufor wejściowy wklejany

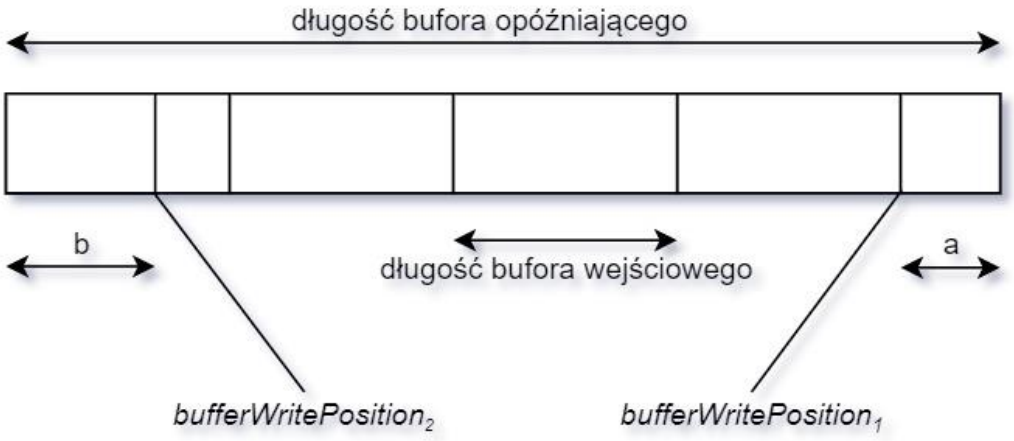
jest taki sposób, że pierwsza jego próbka umieszczana jest jedną próbkę dalej niż koniec poprzedniego bufora, a ostatnia liczbę próbek dalej odpowiadającą długości bufora (w tej sytuacji 4 próbki dalej). W tym przypadku jednak, w buforze opóźniającym nie ma wystarczająco dużo wolnych próbek, aby bufor wejściowy w całości się w nim zmieścił. W tej sytuacji do bufora opóźniającego zostaje wklejona taka ilość próbek, jaka pozostała wolna. Dalsza część bufora wejściowego zostaje wklejona na początek bufora opóźniającego – dane będące tam w tym momencie (w tym przypadku próbki poprzedniego bufora) zostają nadpisane nowymi danymi (A_3 – początek drugiego bufora wejściowego, A_4 – koniec drugiego bufora wejściowego). Jako że poprzedni bufor został już przetworzony, dane te nie są już potrzebne. Można wykorzystać ten fakt i skorzystać z miejsca w pamięci, które może być ponownie wykorzystane.



Rys. 4.7. Schematyczne przedstawienie działania bufora kołowego.

Założone podejście wymaga więc konieczności utworzenia zmiennej przechowującej informację o pozycji kopiowania i odczytywania danych w buforze

opóźniającym. Zmienna ta (*bufferWritePosition*, Rys. 4.8–4.9) zainicjalizowana jest wartością 0, a przy przetwarzaniu każdego kolejnego bufora jej wartość zwiększana jest o wartość długości bufora wejściowego. Na Rys. 4.8 przedstawiona jest sytuacja, w której wartość zmiennej *bufferWritePosition* przekracza wartość długości bufora opóźniającego. Przed zmianą wartości zmienna przyjmuje wartość $bufferWritePosition_1$, po zmianie jej wartość to $bufferWritePosition_2$. Na rysunku wartości a i b to liczby, których suma jest równa długości bufora wejściowego. Tak więc zmienna $bufferWritePosition_2$ jest liczbą powstałą przez odjęcie liczby *a* od długości bufora wejściowego (b to liczba próbek, które nie zmieściły się na końcu bufora opóźniającego. Operacja ta zostaje realizowana korzystając z działania modulo, dzięki temu po przekroczeniu zakresu bufora opóźniającego zmienna przyjmuje odpowiednią wartość (Rys. 4.9).



Rys. 4.8. Schemat przedstawiający sposób zmiany wartości zmiennej *bufferWritePosition*. Suma liczb *a* i *b* jest równa długości bufora wejściowego, $bufferWritePosition_2$ – liczba powstała przez odjęcie liczby *a* od długości bufora wejściowego, *b* – liczba próbek, które nie zmieściły się na końcu bufora opóźniającego.

```
bufferWritePosition += bufferLength;

bufferWritePosition = bufferWritePosition % delayBufferLength;
```

Rys. 4.9. Sposób zmiany wartości zmiennej *bufferWritePosition*.

Jak opisano powyżej, przy wypełnianiu bufora opóźniającego, w pewnym momencie może dojść do sytuacji, w której w buforze opóźniającym nie będzie wystarczająco miejsca, aby zmieścił się kolejny bufor wejściowy. W tej sytuacji bufor opóźniający zostaje wypełniony do końca (tyloma próbkami, ile się zmieści), a reszta próbek bufora wejściowego wklejona zostaje na jego początek. Każdemu kolejnemu wypełnieniu bufora opóźniającego towarzyszy ciągłe nadpisywanie jego wartości nowymi. W tym przypadku, przed skopiowaniem bufora wejściowego do opóźniającego, w pierwszej kolejności sprawdzane jest, czy w buforze opóźniającym jest wystarczająco dużo miejsca, aby bufor wejściowy się zmieścił, a więc czy długość bufora opóźniającego jest większa, niż aktualnej pozycji wpisania (wartości zmiennej *bufferWritePosition)* plus długości bufora wejściowego (Rys. 4.10). Jeśli tak, bufor wejściowy wklejany jest w miejsce aktualnej pozycji zmiennej *bufferWritePosition.* Jeśli nie, bufor wejściowy zostaje podzielony na dwie części: pierwsza o długości bufora opóźniającego minus pozycji wklejenia bufora, druga o długości bufora wejściowego minus długości fragmentu pierwszego (a więc w tym przypadku długość bufora wejściowego minus długość bufora opóźniającego plus pozycja wklejenia bufora). Pierwszy fragment zostaje wklejony w miejsce aktualnej pozycji wskaźnika, drugi na początek bufora opóźniającego (Rys. 4.10).

```
void ReverbEngine::copyBufferToDelayBuffer(int channel, const float* bufferData,
                const float* delayBufferData, const int bufferLength, const int delayBufferLength)
{

    if (delayBufferLength > bufferLength + bufferWritePosition)
    {
        delayBuffer.copyFrom(channel, bufferWritePosition, bufferData, bufferLength);
    }
    else
    {
        delayBuffer.copyFrom(channel, bufferWritePosition, bufferData,
                                        delayBufferLength - bufferWritePosition);

        delayBuffer.copyFrom(channel, 0, bufferData + delayBufferLength - bufferWritePosition,
                                        bufferLength - delayBufferLength + bufferWritePosition);
    }
}
```

Rys. 4.10. Sposób wypełniania bufora opóźniającego buforami wejściowymi.

Następnie z wypełnionego bufora opóźniającego, kopiowany jest bufor z pewnego miejsca wcześniej (z przeszłości, miejsce to zależy od aktualnego czasu opóźnienia – podrozdział 4.3.1) i podmieniany z aktualnym buforem (Rys. 4.11). Podobnie, jak

w przypadku wypełniania bufora opóźniającego, w tym przypadku ponownie wykorzystywana jest właściwość bufora kołowego. Miejsce w buforze opóźniającym, z którego ma odbywać się kopiowanie jest obliczane na podstawie czasu opóźnienia w aktualnej *k* iteracji (Rys. 4.5, iteracja, o której mowa opisania jest w podpunkcie B niniejszego podrozdziału) – aktualnej wartości w wektorze czasów opóźnienia (jest to liczba próbek odpowiadająca temu czasowi opóźnienia, Rys. 4.11). W przypadku, gdy kopiowanie ma odbywać się z miejsca, w którym koniec bufora, który chcemy skopiować wykracza poza rozmiar bufora opóźniającego, zostaje skopiowany jego fragment o długości odpowiadającej liczbie próbek pozostałych w buforze opóźniającym, a następnie jego dalsza część, znajdująca się na początku bufora opóźniającego.

```
void ReverbEngine::copyBackToCurrentBuffer(AudioBuffer<float>& buffer, int channel,
            const float* bufferData, const float* delayBufferData,const int bufferLength,
                                                 const int delayBufferLength, int delayTime)
{

    const int bufferReadPosition = int(delayBufferLength + bufferWritePosition -
                                    (sampleRate_ * delayTime / 1000)) % delayBufferLength;

    if (delayBufferLength > bufferLength + bufferReadPosition)
    {
        buffer.copyFrom(channel, 0, delayBufferData + bufferReadPosition, bufferLength);
    }
    else
    {
        buffer.copyFrom(channel, 0, delayBufferData + bufferReadPosition,
                                    delayBufferLength - bufferReadPosition);

        buffer.copyFrom(channel, delayBufferLength - bufferReadPosition, delayBufferData,
                                    bufferLength - delayBufferLength + bufferReadPosition);

    }
}
```

Rys. 4.11. Sposób podmiany bufora wejściowego na bufor z przeszłości.

Aby uzyskać sumę bufora aktualnego z jego opóźnioną kopią, w następnym kroku do bufora opóźniającego dodawany jest aktualny bufor – dzięki wcześniejszym krokom jest on opóźnioną wersją bufora wejściowego. Tym sposobem podczas przetwarzania kolejnego bufora, przy kopiowaniu fragmentu bufora opóźniającego do aktualnie przetwarzanego, fragment ten zawiera już sygnał wejściowy i opóźniony. Mając w buforze aktualnym sygnał z jego opóźnioną kopią, dalsze przetwarzanie zapewnia powstanie sprzężenia zwrotnego.

Każdy kolejny fragment bufora opóźniającego o długości bufora wejściowego jest sumowany z buforem aktualnym. Odbywa się to na zasadzie podobnej, co kopiowanie bufora wejściowego do opóźniającego. Jeżeli w buforze opóźniającym jest wystarczająca liczba próbek, które nie były jeszcze sumowane z buforem aktualnym, zostaje wykonane sumowanie aktualnego bufora z fragmentem bufora opóźniającego (początkowi tego fragmentu odpowiada aktualna pozycja wskaźnika *bufferWritePosition*). Natomiast, jeżeli w buforze opóźniającym nie ma już wystarczająco dużo próbek (które nie zostały jeszcze poddane operacji sumowania z próbkami aktualnego bufora), pozostałe w nim próbki zostają zsumowane z próbkami aktualnego bufora (ich liczba jest równa różnicy długości bufora opóźniającego i aktualnej pozycji wskaźnika *bufferWritePosition*). Reszta próbek bufora aktualnego dodawana jest do próbek będących na początku bufora opóźniającego (Rys. 4.12).

```
void ReverbEngine::addDelayWithCurrentBuffer(int channel, const int bufferLength,
            const int delayBufferLength, const float* bufferData, float amplitudeMultiplier)
{
    if (delayBufferLength > bufferLength + bufferWritePosition)
    {

        delayBuffer.addFromWithRamp(channel, bufferWritePosition, bufferData,
                        bufferLength, amplitudeMultiplier, amplitudeMultiplier);


    }
    else
    {
        delayBuffer.addFromWithRamp(channel, bufferWritePosition, bufferData,
                delayBufferLength - bufferWritePosition, amplitudeMultiplier, amplitudeMultiplier);

        delayBuffer.addFromWithRamp(channel, 0, bufferData, bufferLength -
                delayBufferLength + bufferWritePosition, amplitudeMultiplier, amplitudeMultiplier);


    }
}
```

Rys. 4.12. Sposób dodania aktualnego bufora do bufora opóźniającego.

B) Utworzenie sygnału z wieloma jego opóźnionymi kopiami.

Mając możliwość uzyskania na wyjściu sygnału z jego opóźnioną kopią (podpunkt A), możliwe jest dodawanie do aktualnie przygotowanego bufora audio na wyjście kolejnych kopii sygnału. Wektor liczb reprezentujących czasy opóźnienia w milisekundach zostaje w tym miejscu wykorzystany do przekazania informacji, z którego miejsca z bufora opóźniającego powinien być kopiowany fragment, aby uzyskać odcinek sygnału z przeszłości – sprzed liczby milisekund danej w wektorze.

Jak wspomniano w podpunkcie A, bufor opóźniający wypełniony szeregiem następujących po sobie buforów nieprzetworzonych posiada wszystkie niezbędne informacje do tego, aby móc z niego kopiować pożądane fragmenty sygnału (długość bufora opóźniającego przekracza maksymalny czas opóźnienia zawarty w wektorze czasów opóźnienia). Dodawanie do aktualnego bufora fragmentów sygnału z różnych miejsc z przeszłości odbywa się w następujący sposób:

- podmiana bufora wejściowego z fragmentem sygnału z przeszłości (miejsce, z którego odbywa się kopiowanie zależy od aktualnego indeksu wektora czasów opóźnienia i jest różne dla każdej iteracji),
- dodanie bufora aktualnego do bufora opóźniającego (podpunkt A).

Iteracja, o która mowa, wykonywana jest liczbę razy równą długości wektora czasów opóźnienia, a więc równa liczbie linii opóźniających – przy każdej iteracji miejsce kopiowania z przeszłości jest inne (wektor czasów opóźnienia składa się z wartości unikatowych). Indeks iteracji oznaczony jest jako *line* w zamieszczonych w kolejnych podrozdziałach fragmentach kodu, a literą *k* na Rys. 4.5. Po wykonaniu wypunktowanych w niniejszym podpunkcie operacji, każdy kolejny fragment w buforze opóźniającym o długości bufora wejściowego jest sumą wszystkich fragmentów sygnału z przeszłości, które były uprzednio kopiowane do bufora wejściowego (suma ta oznaczona jest jako $x_3(n)$ na Rys. 4.5). Towarzyszące przetwarzaniu każdego kolejnego bufora wysyłanie sumy tej na wejście zapewnia obecność sprzężenia zwrotnego, dodatkowo zwiększając zawartość symulacji odbić.

Pierwsza podmiana bufora wejściowego na bufor z przeszłości powoduje, iż w buforze aktualnym chwilowo nie ma bufora reprezentującego dźwięk bezpośredni. Aby przywrócić jego obecność, ostatni element wektora czasów opóźnienia zostaje uprzednio podmieniony na wartość 0, dzięki czemu przy dodawaniu do bufora opóźniającego kopii sygnału z wielu miejsc w czasie, zostaje dodany także bufor opóźniony o 0 ms, a więc dźwięk bezpośredni.

4.3.3. Tłumienie odbić

Symulacja tłumienia kolejnych powtórzeń sygnału wykonywana jest bezpośrednio przed dodaniem aktualnego bufora do bufora opóźniającego (podrozdział

4.3.2). Zastosowana przy tym dodawaniu funkcja biblioteczna *addFromWithRamp()* [36] oprócz dodawania buforów umożliwia tłumienie amplitudy sygnału dodawanego – bezpośrednio przed jego dodaniem do bufora, na którym wywoływana jest funkcja. Dzięki jednemu z argumentów tej funkcji bibliotecznej można przekazać do funkcji informację o tym, jak bardzo ograniczona będzie amplituda bufora dodawanego, przed jego dodaniem. Na Rys. 4.12 argument ten to *amplitudeMultiplier* i jest zmienną mogącą przyjmować inne wartości przy każdym wywołaniu funkcji *addBufferWithDelayBuffer*. Przy jej wywoływaniu w każdej iteracji (poprzedni podrozdział, podpunkt B) zmienna *amplitudeMultiplier* jest liczbą będącą stosunkiem aktualnego tłumienia (docelowo możliwego do poddania modyfikacji przez użytkownika) i liczby linii opóźniających (Rys. 4.13). Dzielenie przez liczbę linii opóźniających jest niezbędne, ponieważ funkcja *addBufferWithDelayBuffer* jest wywoływana (w czasie przetwarzania jednego bufora) liczbę razy równą liczbie linii opóźniających. Amplituda sumy wszystkich kopii sygnału (z wszystkich miejsc w czasie) nie może przekraczać wartości 1 – po osiągnięciu tej wartości następuje jego zniekształcenie (próbki poprawnie zapisanego sygnału cyfrowego zawierają się w przedziale [0;1]).

```
addDelayWithCurrentBuffer(leftChannel, bufferLength, delayBufferLength,
                          bufferDataLa, mWetDry / (float)numberDelayLines);

addDelayWithCurrentBuffer(rightChannel, bufferLength, delayBufferLength,
                          bufferDataRa, mWetDry / (float)numberDelayLines);
```

Rys. 4.13. Sposób tłumienia amplitudy.

4.3.4. Filtracja linii opóźniających

Podobnie jak w przypadku wersji pogłosu z losowymi wartościami czasu opóźnienia linii opóźniających, w tym przypadku również predefinicja pewnej części obliczeń pozwala na ograniczenie złożoności obliczeniowej i tym samym przyspieszenie działania programu.

Filtrami zastosowanymi do ograniczenia pasma częstotliwości poszczególnych linii opóźniających były filtry z nieskończoną odpowiedzią impulsową (IIR). Do

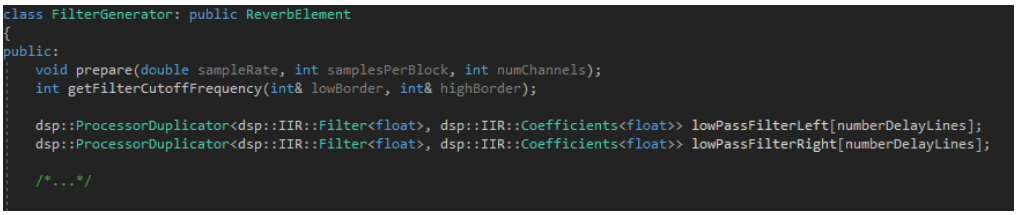
implementacji filtrów wykorzystane zostały klasy biblioteczne realizujące większość obliczeń związanych z projektowaniem filtrów i samą filtracją.

W celu przygotowania filtrów do późniejszego zastosowania przy przetwarzaniu sygnału, skorzystano z bibliotecznych klas i struktur:

- *dsp::ProcessorDuplicator<MonoProcessorType, StateType>* – klasa pozwalająca na przetwarzanie wielokanałowego bufora audio i przetwarzanie sygnału w połączeniu z innymi klasami [37],
- *dsp::IIR::Filter<SampleType>* – klasa realizująca przetwarzanie sygnału filtrem IIR [38],
- *dsp::IIR::Coefficients<NumericType>* – klasa realizująca przechowywanie współczynników filtru IIR w celu późniejszego wykorzystania przez klasę *dsp::IIR::Filter<SampleType>* [39],
- *dsp::ProcessSpec* – struktura niezbędna do prawidłowego przetwarzania sygnału, współpracująca z innymi klasami [40].

Ze względu na konieczność filtracji każdej linii opóźniającej w inny sposób w jednej z wersji pogłosu, stworzono kontener filtrów o różnej częstotliwości odcięcia. Długość kontenera jest równa liczbie linii opóźniających. Tym sposobem przy dodawaniu kolejnych kopii sygnału z różnych momentów przeszłości (podpunkt B, podrozdział 4.3.2) można wykonać filtrację fragmentu bufora z danego miejsca w czasie). Generowanie częstotliwości odcięcia filtrów odbywa się losując wartości liczbowe (w Hz) z zadanego zakresu (Rys. 4.17).

Indeks *numberDelayLines* (Rys. 4.14) to stała typu *static const int* i jest polem stworzonej struktury *ReverbElement.* Stała ta przechowuje liczbę linii opóźniających niezmienną w trakcie działania programu (liczba linii aktywnych w danym momencie może się różnić, w zależności od ustawienia pokrętła dostępnego dla użytkownika, określającego wielkość pogłosu). W celu filtracji lewego i prawego kanału w różny sposób (konieczność taka istniała przy implementacji w jednej z wersji pogłosu), konieczne jest stworzenie dwóch kontenerów, wykorzystane później osobno do filtracji lewego i prawego kanału.



Rys. 4.14. Definicja klasy *FilterGenerator* i ich najważniejszych metod, oraz sposób tworzenia tablicy typu *dsp::ProcessorDuplicator*.

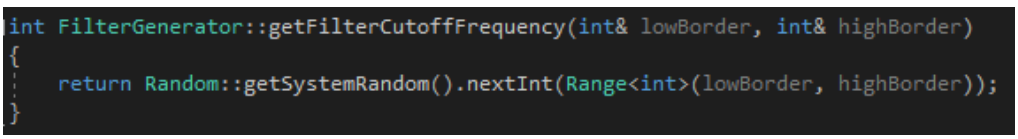
```
std::sort(delayTimes.begin(), delayTimes.end());
std::reverse(delayTimes.begin(), delayTimes.end());
```

Rys. 4.15. Sortowanie wektora czasów opóźnienia linii opóźniających i jego odwrócenie.

```
sort(lowPassCutoffFrequenciesLeft.begin(),
                lowPassCutoffFrequenciesLeft.end());
sort(lowPassCutoffFrequenciesRight.begin(),
                lowPassCutoffFrequenciesRight.end());
```

Rys. 4.16. Sortowanie wektorów częstotliwości odcięcia filtrów dla lewego i prawego kanału.

Chcąc filtrować dolnoprzepustowo tym mocniej (z niższą częstotliwością odcięcia) im fragment kopii sygnału pochodzi z dalszego momentu wstecz w czasie, wykonano sortowanie wartości czasów opóźnienia, przy jednoczesnym sortowaniu częstotliwości odcięcia filtrów (Rys. 4.16). Aby najdłuższy czas opóźnienia odpowiadał najniższej częstotliwości odcięcia, jeden z wektorów musi zostać odwrócony (Rys. 4.15).



Rys. 4.17. Funkcja realizująca generowanie częstotliwości odcięcia filtru dolnoprzepustowego.

```
void FilterGenerator::prepare(double sampleRate, int samplesPerBlock, int numChannels)
{
    dsp::ProcessSpec spec;
    spec.sampleRate = sampleRate;
    spec.maximumBlockSize = samplesPerBlock;
    spec.numChannels = numChannels;

    filtersNumber = numberDelayLines;
    lowBorderFilterFrequency = 100;
    highBorderFilterFrequency = 3000;
    for (int filter = 0; filter < filtersNumber; ++filter)
    {
        lowPassCutoffFrequenciesLeft.push_back(getFilterCutoffFrequency(
                            lowBorderFilterFrequency, highBorderFilterFrequency));
    }

    for (int filter = 0; filter < filtersNumber; ++filter)
    {
        lowPassFilterLeft[filter].prepare(spec);
        lowPassFilterLeft[filter].reset();
        *(lowPassFilterLeft[filter]).state = *dsp::IIR::Coefficients<float>::
                    makeLowPass(sampleRate, lowPassCutoffFrequenciesLeft[filter], 1.0f);
    }

    /*...*/
```

Rys. 4.18. Sposób przygotowania obiektu struktury *dsp::ProcessSpec*, wygenerowanie wektora częstotliwości oraz przygotowania filtrów do dalszego przetwarzania (kanał lewy).

```
dsp::AudioBlock<float> block(buffer);
filterGenerator.lowPassFilterRight[line].process(dsp::ProcessContextReplacing<float>(block));
```

Rys 4.19. Przetwarzanie prawego kanału aktualnego bufora audio przygotowanymi filtrami.

Przy dodawaniu do siebie kopii sygnału z różnych miejsc w czasie (podpunkt B, podrozdział 4.3.2) towarzyszącemu generowaniu wyjściowego bufora, wykonywana jest jego filtracja – ma to miejsce po wykonaniu kopii fragmentu sygnału o długości bufora wejściowego z pewnego miejsca w czasie. Dopiero po filtracji, fragment ten dodawany jest do tworzonej sumy. Poprawność przyporządkowania filtra odpowiedniemu fragmentowi sygnału (z odpowiedniego miejsca w czasie) zapewniona jest przez sortowanie wartości czasów opóźnienia oraz częstotliwości odcięcia filtrów.

Ostateczny krok, a więc sama filtracja bufora wymaga zastosowania klasy bibliotecznej *dsp::AudioBlock* [41] i przypisania do jej obiektu pożądanego do filtracji bufora. To, jaką zawartość posiada bufor w momencie przypisywania klasie *dsp::AudioBlock* (Rys. 4.19), zależy od miejsca tego przypisania, a więc od aktualnego

etapu przetwarzania. Następnie dane w tej postaci zostają wykorzystane przez klasę *dsp::ProcessContextReplacing.* W wyniku operacji tych, pożądany bufor filtrowany jest filtrem zdefiniowanym w opisanych wcześniej działaniach (Rys. 4.18, indeks *filter* to numer filtru w kolejnej iteracji i odpowiada iteratorowi *k* na Rys. 4.5, filtr w danej iteracji oznaczono tam jako IIR_1 dla kanału lewego i IIR_2 dla kanału prawego).

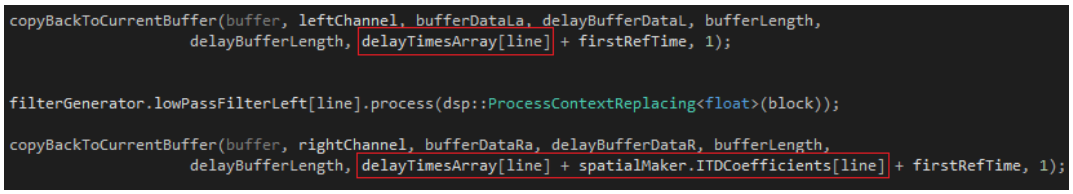
4.3.5. Uprzestrzennienie pogłosu

A) Międzyuszna różnica czasu

Podobnie jak w przypadku tworzenia kontenera filtrów, którego długość odpowiadała długości wektora czasów opóźnienia (w celu wykonania innej filtracji dla każdej linii opóźniającej), w przypadku chęci symulacji zjawiska międzyusznej różnicy czasu konieczne jest stworzenie danych w wektorze o tej samej długości. Dzięki temu każda linia opóźniająca będzie mogła być modyfikowana w inny sposób. Stworzony wektor zawiera dane liczbowe z informacją o różnicy czasu opóźnienia między lewym i prawym kanałem dla każdej linii opóźniającej (Rys. 4.20). Generowanie wektora odbywa się takiej samej zasadzie, co wektora częstotliwości odcięcia filtrów, z tą różnicą, iż zakres losowania wartości jest przedziałem [-40;40] ms (podrozdział 2.2.2).

```
void SpatialMaker::createITDarray()
{
    ITDCoefficients.clear();
    for (int line = 0; line < numberDelayLines; ++line)
    {
        ITDCoefficients.push_back(Random::getSystemRandom().
                                  nextInt(Range<int>(-40, 40)));

    }
}
```

Rys. 4.20. Sposób tworzenia wektora różnicy czasów opóźnienia.



Rys. 4.21. Wykorzystanie wektora różnicy czasów opóźnienia w celu symulacji zjawiska międzyusznej różnicy czasu.

Przekazywana do funkcji *copyBackToCurrentBuffer* (ciało funkcji na Rys. 4.2) aktualna wartość wektora *delayTimesArray* (Rys. 4.21) w danej iteracji jest liczbą milisekund odpowiadającą miejscu, z którego kopiowany jest bufor (podrozdział 4.3.2). Dodając do tej wartości aktualną wartość wektora różnicy czasów opóźnienia, a więc liczbę z zakresu [-40;40] ms, wywołania funkcji powodują kopiowanie buforów z różnych miejsc w czasie dla lewego i prawego kanału. Miejsce kopiowania różni się maksymalnie o 40 ms. Zakres losowania uwzględnia liczby ujemne – dzięki temu wystarczające jest dodanie wylosowanej liczby do czasu opóźnienia tylko jednego z kanałów. W przypadku wylosowania liczby ujemnej, opóźniony jest bufor aktualnej (w danej iteracji) kopii kanału lewego sygnału, w przypadku liczby dodatniej – kanału prawego. Liczba próbek odpowiadająca wylosowanej liczbie oznaczona jest jako m_{kR} na Rys. 4.5.

Tworzenie wektora różnicy czasów opóźnienia należy do części obliczeń wykonywanych przed rozpoczęciem przetwarzania sygnału, dzięki temu wartości losowane są tylko raz, co wpływa pozytywnie na szybkość obliczeń. Dodatkowo, losowanie liczb na bieżąco (przy przetwarzaniu kolejnych buforów) skutkuje negatywnymi, słyszalnymi artefaktami w sygnale, tak jak w przypadku losowania czasów opóźnienia (podrozdział 4.3.1).

B) Międzyuszna różnica poziomu

Symulacja zjawiska zakłada ograniczenie amplitudy jednego z kanałów w każdej linii opóźniającej. W tym przypadku predefinicja wartości wpływających na tłumienie amplitudy nie jest konieczna – manipulacja amplitudą podczas działania programu nie powoduje żadnych negatywnych skutków (ograniczanie amplitudy polega jedynie na mnożeniu wartości próbek przez liczby). Amplituda ograniczana jest bezpośrednio

przed dodaniem aktualnego bufora do bufora opóźniającego (w ciele funkcji *addDelayWithCurrentBuffer*, podrozdział 4.3.2) – jest to miejsce przesyłania do funkcji argumentu odpowiedzialnego za tłumienie (Rys. 4.22–4.23). Aby zjawisko zachodziło w obie strony (aby w niektórych liniach opóźniających głośniejszy był sygnał w lewym kanale, w innych w prawym), zbiór wartości liczb losowanych zawiera liczby dodatnie i ujemne (jest to przedział [-0,5;0,5]). Wylosowana liczba dodawana jest do aktualnej wartości amplitudy jednego z kanałów i dzielona przez liczbę linii opóźniających (podrozdział 4.3.3). Dodatkowo za pomocą zmiennej typu logicznego (zmienna *ILD_on*) sprawdzane jest, czy użytkownik włączył opcję działania symulacji tego zjawiska przestrzennego (Rys. 4.22).

```
if (ILD_on)
{
    float ILD = Random::getSystemRandom().nextFloat() - 0.5f;
    addDelayWithCurrentBuffer(leftChannel, bufferLength, delayBufferLength,
                    bufferDataLa, (mWetDry + ILD) / (float)numberDelayLines);
}
else
    addDelayWithCurrentBuffer(leftChannel, bufferLength, delayBufferLength,
                        bufferDataLa, mWetDry / (float)numberDelayLines);

addDelayWithCurrentBuffer(rightChannel, bufferLength, delayBufferLength,
                        bufferDataRa, mWetDry / (float)numberDelayLines);
```

Rys. 4.22. Sposób ograniczania amplitudy w celu symulacji zjawiska międzyusznej różnicy poziomu.

```
void ReverbEngine::addDelayWithCurrentBuffer(int channel, const int bufferLength,
            const int delayBufferLength, const float* bufferData, float amplitudeMultiplier)
{
    if (delayBufferLength > bufferLength + bufferWritePosition)
    {

        delayBuffer.addFromWithRamp(channel, bufferWritePosition, bufferData,
                        bufferLength, amplitudeMultiplier, amplitudeMultiplier);

    }

    /*...*/
```

Rys. 4.23. Sposób wykorzystania przesłanej do funkcji zmiennej odpowiedzialnej za tłumienie.

C) Różna filtracja obu kanałów

Symulacja przestrzenności z wykorzystaniem filtrów zakłada filtrację lewego i prawego kanału w inny sposób, dzięki czemu słuchacz odczuwać będzie większe wrażenie przestrzenności.

W realizowanym podejściu różna filtracja lewego i prawego kanału wymaga stworzenia dwóch kontenerów (każdy dla jednego kanału) zawierających informacje o filtrach dla każdej linii opóźniającej w ramach obu kanałów. W odróżnieniu jednak do wykonania jednego losowania częstotliwości odcięcia, jak miało to miejsce w filtracji opisanej w podrozdziale 4.3.4, są one losowane dwukrotnie – dzięki temu dla lewego i prawego kanału przyjmują inne wartości. Dzięki osobnemu losowaniu dla obu kanałów istnieje duża szansa, iż wylosowane częstotliwości w obu wektorach będą różne. Dodatkowo, zakresy losowania dla obu wektorów częstotliwości odcięcia różnią się o niewielkie wartości. Zakresy mają jednak zbliżoną wartość średnią, co zapewnia uniknięcie sytuacji, w której jeden kanał jest znacznie mocniej filtrowany niż drugi, a co za tym idzie ma słyszalnie niższy poziom. Operacje losowania oraz przygotowania filtrów zostały wykonane podobnie jak w poprzednim podrozdziale (Rys. 4.24– 4.25).

```
filtersNumber = numberDelayLines;
lowBorderFilterFrequency = 100;
highBorderFilterFrequency = 3000;
for (int filter = 0; filter < filtersNumber; ++filter)
{
    lowPassCutoffFrequenciesLeft.push_back(getFilterCutoffFrequency(
                        lowBorderFilterFrequency, highBorderFilterFrequency));
}

for (int filter = 0; filter < filtersNumber; ++filter)
{
    lowPassFilterLeft[filter].prepare(spec);
    lowPassFilterLeft[filter].reset();
    *(lowPassFilterLeft[filter]).state = *dsp::IIR::Coefficients<float>::
                makeLowPass(sampleRate, lowPassCutoffFrequenciesLeft[filter], 1.0f);
}
```

Rys. 4.24. Sposób przygotowania filtrów dolnoprzepustowych dla lewego kanału, zakres losowania częstotliwości odcięcia różni się od losowania dla kanału prawego.

```
int rightLowHighpassFreq = 80;
int rightHighHighpassFreq = 3800;
for (int filter = 0; filter < filtersNumber; ++filter)
{
    lowPassCutoffFrequenciesRight.push_back(getFilterCutoffFrequency(
                            rightLowHighpassFreq, rightHighHighpassFreq));

}

for (int filter = 0; filter < filtersNumber; ++filter)
{
    lowPassFilterRight[filter].prepare(spec);
    lowPassFilterRight[filter].reset();
    *(lowPassFilterRight[filter]).state = *dsp::IIR::Coefficients<float>::
            makeLowPass(sampleRate, lowPassCutoffFrequenciesRight[filter], 1.0f);
}
```

Rys. 4.25. Sposób przygotowania filtrów dolnoprzepustowych dla prawego kanału, zakres losowania częstotliwości odcięcia różni się od losowania dla kanału lewego.

Mając wszystkie niezbędne dane przygotowane do wykonania filtracji, możliwe jest poddanie tej operacji pożądanych buforów. Jak wspomniano w podrozdziale 4.3.4 – to, jaką zawartość posiada bufor w momencie przypisywania obiektowi klasy *dsp::AudioBlock* zależy od miejsca tego przypisania w kodzie, a więc od aktualnego etapu przetwarzania. Chcąc więc poddawać filtracji sygnał tylko z jednego z kanałów, konieczne jest dokonanie tego przypisania w momencie, gdy dany bufor zawiera tylko jeden z kanałów. Tak więc w przypadku chęci filtracji kanału lewego, funkcja *process* klasy *dsp::IIR::Filter* musi zostać wywołana pomiędzy skopiowaniem lewego kanału fragmentu bufora opóźniającego do aktualnego (aktualny bufor w tym momencie zawiera kopię sygnału wejściowego z przeszłości), a prawym. Do wykonania tej operacji używany jest kontener filtrów przygotowanych do filtracji lewego kanału Rys. 4.26). Tym sposobem filtrowany jest tylko bufor kanału lewego. Analogicznie, w przypadku chęci filtracji bufora prawego kanału konieczne jest przypisanie do obiektu klasy *AudioBlock* w momencie, kiedy w aktualnym buforze znajduje się tylko prawy kanał.

```
copyBackToCurrentBuffer(buffer, leftChannel, bufferDataLa, delayBufferDataL,
            bufferLength, delayBufferLength,delayTimesArray[line] + firstRefTime);

dsp::AudioBlock<float> blockLeftChannel(buffer);
filterGenerator.lowPassFilterLeft[line].process(dsp::
                        ProcessContextReplacing<float>(blockLeftChannel));
```

Rys. 4.26. Sposób filtracji bufora pojedynczego kanału.

D) Symulacja różnicy w poziomie pierwszych odbić bocznych

Podobnie jak w przypadku implementacji międzyusznej różnicy poziomu, wykonana została modyfikacja amplitudy w celu uzyskania jej różnicy między lewym i prawym kanałem. W tym przypadku jednak, ograniczenie amplitudy konieczne jest tylko dla jednego odbicia – dla kopii sygnału o czasie odbicia pierwszego, a więc najkrótszym czasie opóźnienia.

```
if (line == numberDelayLines - 2)
{
    addDelayWithCurrentBuffer(leftChannel, bufferLength, delayBufferLength, bufferDataLa, NULL,
                                    mWetDry / (float)numberDelayLines, dry);
    addDelayWithCurrentBuffer(rightChannel, bufferLength, delayBufferLength, bufferDataRa, NULL,
                                    (mWetDry - lateralAmplitudeDifference) / (float)numberDelayLines, dry);

}
```

Rys. 4.27. Realizacja różnicy poziomu między kanałami pierwszych odbić bocznych.

Bezpośrednio przed dodaniem skopiowanego i przefiltrowanego bufora do sumy buforów z wszystkich założonych miejsc w czasie, wykonywana jest modyfikacja amplitudy pierwszego odbicia – a więc bufora skopiowanego sprzed liczby próbek odpowiadającej liczbie milisekund pierwszego odbicia. W posortowanej tablicy *numberDelayLines* (podrozdział 4.3.4) wartość pierwszego odbicia znajduje się w przedostatnim elemencie tablicy (wartość najmniejsza w wektorze nie licząc dźwięku bezpośredniego). Na Rys. 4.27 przedstawiono sposób modyfikacji amplitudy w przedstawionym podejściu (*mWetDry* – zmienna reprezentująca aktualną amplitudę sygnału przetworzonego, przyjmuje wartości z zakresu [0,1]), *lateralAmplitudeDifference* – zmienna reprezentująca różnicę amplitudy. Pierwsze wywołanie funkcji *addDelayWithCurrentBuffer* (podrozdział 4.3.2) dotyczy kanału lewego, drugie – prawego. Fragmenty oznaczone czerwonymi obramowaniami przekazują do funkcji informację, jak bardzo tłumione będzie pierwsze odbicie.

Zmienna *lateralAmplitudeDifference,* a więc tłumienie amplitudy prawego kanału, może być modyfikowana przez użytkownika z zakresie [0;1] (podrozdział 4.4)).

4.3.6. Późny ogon pogłosowy

A) Późny pogłos zagęszczony odbiciami:

Implementacja tej wersji pogłosu polegała na zastosowaniu większej liczby linii opóźniających o dłuższym czasie pogłosu niż o czasie pogłosu krótszym. W tym celu, generowanie wektora zawierającego czasy opóźnienia nastąpiło w inny sposób niż w przypadku standardowej wersji pogłosu (podrozdział 4.3.1).

```
std::vector<int> DelayTimesGenerator::getDelayTimes(int & delayTimesNumber,
                                int& lowTime, int& highTime, int& firstRefTime)
{
    std::vector<int>delayTimes;
    delayTimes.clear();

    for (int i = 0; i < delayTimesNumber/4; ++i)
    {
        delayTimes.push_back(this->delayTimesPrime_[i * 4]);

    }

    for (int i = delayTimesNumber / 4; i < delayTimesNumber; ++i)
    {
        delayTimes.push_back(this->delayTimesPrime_[i]);

    }

    /*...*/
```

Rys. 4.28. Sposób zagęszczenia późnego ogona pogłosowego.

Zmienna *delayTimesPrime_* to pole klasy realizującej generowanie wartości czasów opóźnienia (stworzona klasa *DelayTimesGenerator*) i zawiera wszystkie liczby pierwsze z obejmującego pożądane czasy opóźnienia. Pierwsza ćwiartka elementów wartości czasów opóźnienia przygotowanych do późniejszego przetwarzania to fragment wektora złożony z co czwartej liczby pierwszej zaczynając od najmniejszej (w tym przypadku 2 ms). Reszta wektora, a więc jego trzy czwarte, jest złożona z wszystkich liczb pierwszych występujących po sobie po kolei od wartości

maksymalnej pierwszej ćwiartki wektora (Rys. 4.28). Tym sposobem w drugiej części wektora znajduje się o wiele więcej liczb, w przedziale liczbowym o podobnej długości.

B) Późny pogłos z wykorzystaniem szumu:

Do zagęszczenia późnego pogłosu w tym wariancie został wykorzystany szum biały, wygenerowany za pomocą klasy bibliotecznej *Random*, wykorzystując możliwość wygenerowania pseudolosowej liczby zmiennoprzecinkowej z zakresu [0;1] o rozkładzie Gaussa. Po wypełnieniu bufora szumem, zostaje wykonana jego filtracja dolnoprzepustowa, a następnie bufor szumowy mnożony jest z buforem sygnału w sposób opisany poniżej.

```
noiseBuffer.setSize(2, bufferLength);
noiseBuffer.clear();
for (int sample = 0; sample < bufferLength; ++sample)
{
    noiseBuffer.addSample(leftChannel, sample,
                Random::getSystemRandom().nextFloat());
    noiseBuffer.addSample(rightChannel, sample,
                Random::getSystemRandom().nextFloat());

}
```

Rys. 4.29. Sposób wypełnienia lewego i prawego kanału bufora szumem białym.

W pierwszej kolejności konieczne jest wypełnienie lewego i prawego kanału bufora pseudolosowymi liczbami zmiennoprzecinkowymi. Sposób wypełnienia bufora przedstawiony jest na Rys. 4.29.

```
void FilterGenerator::prepareNoiseFilters(double sampleRate,
           int samplesPerBlock, int numChannels, dsp::ProcessSpec spec)
{

    int lowBandFreq = 40, highBandFreq = 750;

    for (int filter = 0; filter < numberDelayLines; ++filter)
        noiseFiltersFrequencies.push_back(getFilterCutoffFrequency
                                              (lowBandFreq, highBandFreq));


    for (int filter = 0; filter < filtersNumber; ++filter)
    {
        noiseFilters[filter].prepare(spec);
        noiseFilters[filter].reset();
        *(noiseFilters[filter]).state = *dsp::IIR::Coefficients<float>::
                    makeLowPass(sampleRate, noiseFiltersFrequencies[filter], 1.0f);
    }

    std::sort(noiseFiltersFrequencies.begin(), noiseFiltersFrequencies.end());
    std::reverse(noiseFiltersFrequencies.begin(), noiseFiltersFrequencies.end());

}
```

Rys. 4.30. Metoda realizacji przygotowania filtrów do filtracji buforów szumowych.

Przygotowanie filtrów do filtracji buforów szumowych zostało wykonane w sposób analogiczny, jak filtrów w podrozdziale 4.3.4. W tym przypadku jednak, wektor częstotliwości odcięcia filtrów dolnoprzepustowych jest posortowany odwrotnie – ostatnia wartość wektora to częstotliwość najwyższa, a pierwsza – najniższa (Rys. 4.30). A więc, przy przetwarzaniu sygnału bufor przefiltrowany filtrem o najwyższej częstotliwości odcięcia mnożony jest buforem sygnału przefiltrowanym filtrem o najniższej częstotliwości odcięcia. Działania te powodują, iż im dalsza część pogłosu, tym zawiera on więcej szumu, a mniej filtrowanych kopii sygnału. Sama filtracja odbywa się analogicznie do filtracji w podrozdziale 4.3.4 (Rys. 4.31).

```
dsp::AudioBlock<float>noiseBlock(noiseBuffer);
filterGenerator.noiseFilters[line].process(dsp::
            ProcessContextReplacing<float>(noiseBlock));
```

Rys. 4.31. Filtracja bufora wypełnionego szumem.

```
if (noiseOn)
{
    for (int sample = 0; sample < bufferLength; ++sample)
    {
        bufferWriteL[sample] *= (noiseBufferDataL[sample]);
        bufferWriteR[sample] *= (noiseBufferDataR[sample]);

    }
}
```

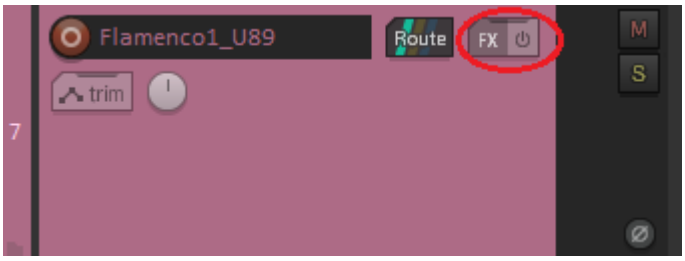
Rys. 4.32. Mnożenie bufora sygnału z szumem filtrowanym szumem białym.

Bufor szumowy i aktualny bufor sygnału mają taką samą długość – każda kolejna próbka wyjściowego sygnału to próbka bufora aktualnego sygnału pomnożona z odpowiadającą jej próbką bufora szumowego (Rys. 4.32).

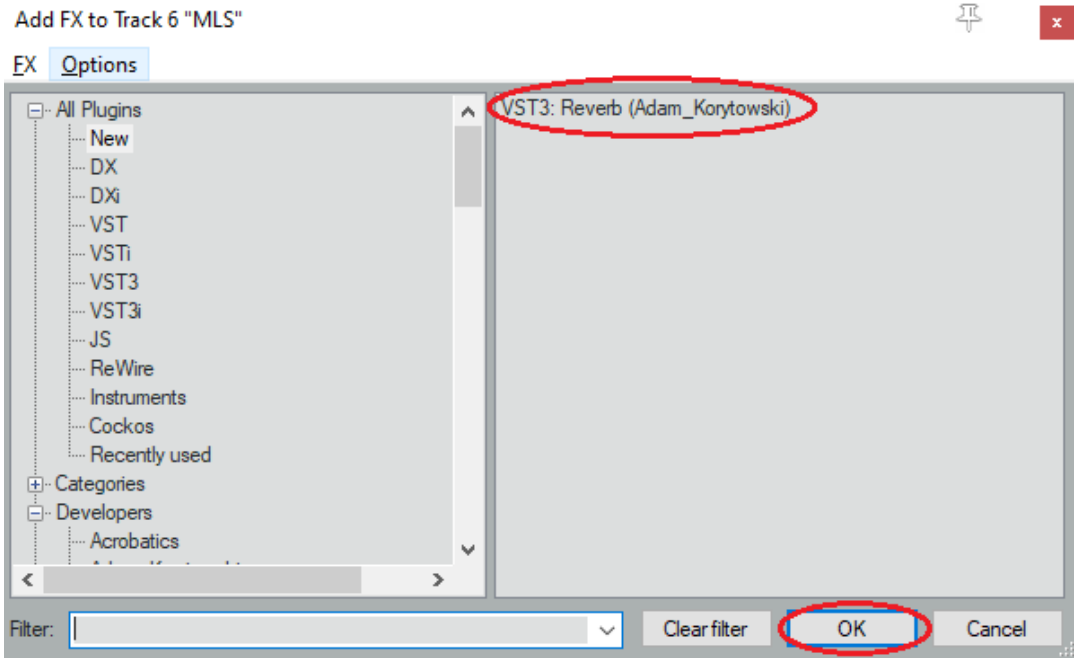
4.4. Część funkcjonalna aplikacji

4.4.1. Przygotowanie wtyczki do działania

Format aplikacji (VST3) pozwala na korzystanie z aplikacji w oprogramowaniu typu DAW (*ang. Digital Audio Workstation*). Aby wtyczka była widziana przez oprogramowanie, niezbędne jest umieszczenie pliku .vst3 w odpowiednim katalogu. W przypadku oprogramowania DAW Reaper jest to katalog *\REAPER\Plugins* znajdujący się w folderze z zainstalowanym programem. W celu wczytania wtyczki na daną ścieżkę w programie, należy skorzystać z odpowiedniego dla programu sposobu umożliwiającego tą operację. W programie Reaper możliwe jest to naciskając przycisk *FX* na pożądanej ścieżce, a następnie, po pojawieniu się okna z możliwością znalezienia odpowiedniej wtyczki, zaznaczenie jej przez kliknięcie lewego klawisza myszy i kliknięcia *OK* (Rys. 4.33– 4.34).



Rys. 4.33. Przycisk umożliwiający przypisanie wtyczki do ścieżki w programie Reaper 6.



Rys. 4.34. Sposób załadowania wtyczki VST3 do ścieżki w programie Reaper 6.

Po poprawnym załadowaniu wtyczki do ścieżki, sygnał przypisany do tej ścieżki będzie sygnałem przetworzonym przez operacje, które wykonuje wtyczka podczas jej działania.

4.4.2. Interfejs użytkownika aplikacji

Dokładny sposób przetworzenia sygnału przez wtyczkę zależy od ustawień elementów interfejsu danych użytkownikowi do modyfikacji. Interfejs użytkownika posiada 4 pokrętła, 3 włączniki oraz listę rozwijaną (Rys. 4.35) – elementy te pozwalają na zmianę parametrów pogłosu, a co za tym idzie na jego brzmienie.

Elementy interfejsu użytkownika to (Rys. 4.35):

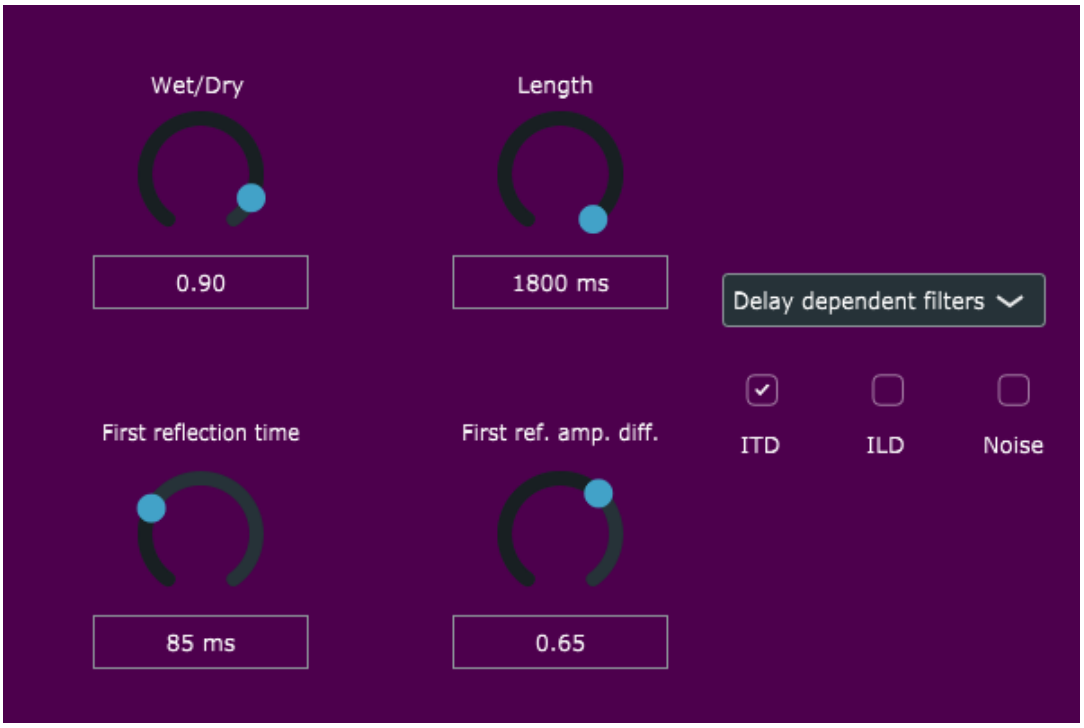
A) Pokrętła:

- *Wet/Dry* – umożliwia dostosowanie zawartości sygnału czystego z sygnałem przetworzonym (zakres: 0 – 1, gdzie 0 oznacza sam sygnał nieprzetworzony, a 1 – maksymalną wartość sygnału przetworzonego i czystego),
- *Length* – umożliwia dostosowanie długości pogłosu. Długość w tym przypadku oznacza wartość maksymalnego opóźnienia linii opóźniającej (zakres: 0–1800 ms),
- *First reflection time* – umożliwia ustawienie czasu pierwszego odbicia (zakres: 0– 300 ms),
- *First reflection amplitude difference* – umożliwia ustawienie różnicy w amplitudzie pomiędzy lewym i prawym kanałem pierwszego odbicia (zakres: [0;1], 0 – brak różnicy w amplitudzie, a 1 maksymalną różnicę – lewy kanał ma amplitudę 0).

B) Włączniki:

- ITD – włączenie symulacji międzyusznej różnicy czasu,
- ILD – włączenie symulacji międzyusznej różnicy poziomu,
- Noise – włączenie opcji mnożenia sygnału pogłosowego z szumem białym (podrozdział 4.3.6).

C) Lista rozwijalna – umożliwia zmianę wersji filtracji linii opóźniających: wersja z jednakową filtracją lub filtracją zależną od czasu opóźnienia linii opóźniających (podrozdział 4.3.4).



Rys. 4.35. Interfejs użytkownika aplikacji.

5. Podsumowanie i wnioski

Celem niniejszej pracy było stworzenie cyfrowej symulacji zjawiska pogłosu akustycznego w oparciu o analizę istniejących rozwiązań implementacyjnych oraz świadomość istnienia percepcyjnych zjawisk akustycznych. Aplikacja docelowo została zrealizowana w formacie VST. W rozdziale 2 przedstawione zostały najpopularniejsze podejścia do realizacji sztucznego pogłosu oraz opis znajdujących się w literaturze sposobów implementacyjnych elementów pogłosu. Realizacja docelowej aplikacji poprzedzona była implementacją różnych wersji pogłosu na podstawie znajdujących się w literaturze sposobów realizacji opisanych elementów. W rozdziale 3 dokonano oceny tych wersji stworzonymi na potrzeby pracy parametrami, których wartości liczbowe były wyznacznikiem ich jakości. Zdefiniowanie parametrów tych było odpowiedzią na powtarzające się w literaturze podkreślenia cech sygnału pogłosowego wpływających na jego jakość oraz odczucie wrażenia przestrzenności; do ich stworzenia wykorzystano znane parametry statystyczne stosowane do analizy widmowej i czasowej sygnału oraz do oceny podobieństwa sygnałów. Ocena ta miała na celu wyłonienie najlepszych sposobów implementacyjnych w obrębie badanych elementów: sposobu generowania czasów opóźnienia linii opóźniających, filtracji, przestrzenności oraz późnego ogona pogłosowego. Na podstawie tych wartości dokonany został wybór rozwiązań implementacyjnych w celu późniejszego ich połączenia przy realizacji docelowego pogłosu i tym samym uzyskania hybrydowego rozwiązania.

Ostateczny pogłos utworzony z wersji, które uzyskały najlepsze wyniki liczbowe stworzonych parametrów jest złożony z: linii opóźniających o długościach będących liczbami wzajemnie pierwszymi, filtrowanymi jednakowo, z symulacją wrażenia przestrzenności za pomocą różnicy amplitudy pierwszych odbić bocznych, z późnym ogonem pogłosowym mnożonym z filtrowanym szumem białym. Przy wyborze wersji pogłosu, w większości przypadków wartości parametrów nie pozwalały na jednoznaczne wskazanie. Wartości te często zależały od rodzaju sygnału, dlatego w niektórych przypadkach oprócz wybranej wersji, do interfejsu użytkownika dodano także opcję alternatywną, opcjonalną dla użytkownika. W przypadku filtracji, do interfejsu została dodana opcja zmiany filtracji linii opóźniających na wersję, w której im linia opóźniająca jest dłuższa, tym filtr dolnoprzepustowy, którym jest filtrowana ma niższą częstotliwość odcięcia. Dodatkowo, umożliwiono użytkownikowi włączenie

symulacji zjawisk przestrzennych: międzyusznej różnicy poziomu oraz czasu. Stworzony interfejs użytkownika pozwala na manipulację parametrami pogłosu w czasie działania aplikacji, w celu zmiany jego brzmienia wedle preferencji lub z zależności od rodzaju sygnału.

W rozdziale 4 przedstawiono zaprojektowaną procedurę realizacji docelowego pogłosu wraz ze szczegółami implementacyjnymi. Docelowa aplikacja została zrealizowana przy użyciu frameworku JUCE w języku C++. Narzędzie to pozwoliło aplikacji na komunikację z zewnętrznym oprogramowaniem typu DAW. Dzięki temu wtyczka przyporządkowana do ścieżki zawierającej sygnał cyfrowy przetwarza go zgodnie z operacjami zaimplementowanymi przy tworzeniu aplikacji.

Należy pamiętać, iż zjawiska zastosowane w implementacji dotyczące wrażeń przestrzennych związane są jedynie z lokalizacją źródła dźwięku w płaszczyźnie poziomej. Istnieje więcej czynników wpływających na wrażenie przestrzenności, takie jak np. wpływ małżowin usznych lub geometrii głowy i ciała słuchacza ze szczególnym wskazaniem na Funkcję Transmitancji Głowy *(Head Related Transfer Function).* A także innych zjawisk, dzięki którym możliwe jest lokalizowanie źródła poza płaszczyzną poziomą. Dokładna identyfikacja i parametryzacja tych zjawisk wymaga dalszych badań, pozwoli to na dokładniejsze odwzorowanie wrażenia przestrzenności w przypadku chęci ich stosowania w podobnych projektach. Ponadto, ze względu na fakt działania różnych zjawisk psychoakustycznych w różnych zakresach częstotliwości, wrażenie przestrzenności silnie zależy od rodzaju sygnału i zawartości jego widma. Tak więc faktyczna sytuacja związana z propagacją fali akustycznej w pomieszczeniu jest znacznie bardziej skomplikowana niż symulacja będąca wynikiem zaimplementowanego rozwiązania. Istnieje ponadto o wiele więcej zjawisk akustycznych, możliwych do uwzględnienia przy realizacji pogłosu – tak, aby uzyskać większą jakość i realność pogłosu. Należy jednak pamiętać, iż przedstawione rozwiązanie miało na celu realizację pogłosu w formie efektu, w odróżnieniu do uzyskania jak najbardziej realistycznego zjawiska pogłosu – dlatego też podejście do realizacji projektu zakładało wydajność oraz praktyczność użytkowania aplikacji.

Należy pamiętać również o tym, iż duża liczba zmiennych przy badaniach (m. in. minimalna i maksymalna wartość czasu opóźnienia linii opóźniającej, zakresy częstotliwości odcięcia filtrów, różnica w amplitudzie odbić bocznych) powoduje, iż

wszystkie możliwe przypadki są praktycznie niemożliwe to zbadania (ze względu na ich liczbę), ponadto w zależności od ich kombinacji wyniki mogą się różnić. Dodatkowo, manipulacja zmiennych przez użytkownika podczas działania programu także może wpływać na zdefiniowane parametry.

Historia realizacji tego typu projektów pokazuje, iż czynnik oceny subiektywnej jest nieodłącznym jej elementem. Ma to związek z niewielkimi możliwościami obliczeniowymi w czasach publikacji kluczowych dla tematyki prac, jak również z powtarzalnością ocen przy testach odsłuchowych. Przez powtarzalność opinii osób badanych na temat sygnałów, metodę badawczą uznawano za wiarygodną. Stworzone na potrzeby niniejszej pracy parametry i sposób ich zastosowania należy traktować jako próbę wskazania drogi, która docelowo ma prowadzić do pokrywania się wartości parametrów oceniających pogłos z subiektywnymi wrażeniami słuchaczy. Nasuwa się więc pomysł zweryfikowania, czy jakość pogłosu wyrażona otrzymanymi wartościami pokryje się z jakością wskazaną przez słuchaczy, lub zaproponowania nowych parametrów i ich weryfikacji również porównując otrzymane wartości ze wskazaniami subiektywnymi. W przypadku znalezienia parametrów zbieżnych co do oceny ze wskazaniami subiektywnymi zostałaby wyeliminowana konieczność przeprowadzania subiektywnych badań, które są bardziej czasochłonne, a ponadto niezbędne jest wykorzystanie do nich dużej grupy osób. Będzie bowiem istniała możliwość oceny jakości pogłosu z dużą szansą, iż wskazania parametrów pokrywają się z wrażeniami słuchowymi.

Bibliografia

[1] Barron M.: *The subjective effects of first reflections in concert halls – the need for lateral reflections*, Kwiecień, 1969

[2] Zölzer U., *DAFX: Digital Audio Effects. Second Edition*, John Wiley & Sons Ltd, Chichester 2011

[3] Drobner M.: *Instrumentoznawstwo I akustyka*, Podręcznik dla szkół muzycznych II stopnia, Polskie wydawnictwo muzyczne, Kraków, 1980

[4] Oleszkowicz J.: *Muzyka Elektronika Informatyka*, Centrum Edukacji Artystycznej, Warszawa 2010

[5] Drobner M., Golachowski S.: *Akustyka Muzyczna*, Polskie wydawnictwo muzyczne, Kraków, 1953

[6] Davis G., Jones R.: *The sound reinforcement handbook* (2nd ed.). Hal Leonard, USA, 1987

[7] Drabek P., Zalesak M.: *Reverberation chamber and its verification for acoustic measurements*, Tomas Bata University in Zlin, Faculty of Applied Informatics, Department of Automation and Control Engineering, Czechy, 2016

[8] Charakterystyka splotu dyskretnego. Dostępny: https://pages.jh.edu/~signals/discreteconv2/index.html (odwiedzona 25.08.20)

[9] Schroeder M. R., Logan B. F. – *Colorless Sounding Artificial Reverberation*, Journal of the Audio Engineering Society, Bell Telephone Laboratories, Incorporated, Murray Hill, vol. 9, lipiec 1961

[10] Schroeder M. R.: *Natural Sounding Artificial Reverberation*, Journal of the Audio Engineering Society, Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey, Vol. 10, No. 3, lipiec 1962

[11] Charakterystyka filtra wszechprzepustowego. Dostępny: https://ccrma.stanford.edu/~jos/pasp/Schroeder_Allpass_Sections.html (odwiedzona 25.08.20)

[12] Stautner J., Puckette M.: *Designing Multi–Channel Reverberators*, Computer Music Journal Vol. 6, No. 1, wiosna 1982

[13] Jot J. – M., Chaigne A.: *Digital Delay Networks for Designing Artificial Reverberators*, 90th Convetion of Audio Engineering Society, Paryż, luty 1991

[14] Beranek L.: *Aspects of concert halls acoustics*, Richard C. Heyser Memorial Lecture, Audio Engineering Society 123rd AES Convention, Nowy Jork, 2007

[15] Karjalainen M., J¨arvel¨ainen H.: *More about this reverberation science: Perceptually good late reverberation*, 111th Convention of Audio Engineering Society, Nowy Jork, Wrzesień 2001

[16] Välimäki V., Holm– Rasmussen B., Alary B., Lehtonen H.*: Late Reverberation Synthesis Using Filtered Velvet Noise,* Acoustic Lab, Department of Signal Processing and Acoustics, Aalto University, Finland, Maj 2017

[17] Kleczkowski P.: *Percepcja Dźwięku,* Wydawnictwa AGH, Kraków, 2013

[18] *Rayleigh Duplex Theory*, Binaural Sound Localization – Lewis Scott Diamond. Dostępny: http://diamonddissertation.blogspot.com/2010/05/rayleigh–duplextheory.html (odwiedzona 25.08.20).

[19] Smyth T.: *Time and Space*, Department of Music, University of California, San Diego (UCSD), Kwiecień 2019. Dostępny: http://musicweb.ucsd.edu/~trsmyth/space175/space175.pdf (odwiedzona 25.08.20).

[20] Priemer R.: *Introductory Signal Processing,* Advanced Series in Electrical and Computer Engineering – Vol. 6, Chicago, USA, 1991

[21] Stoica P., Moses R.: *Spectral Analysis of Signals,* Parentice Hall, Upper Saddle River, New Jersey, 2005. Dostępny: http://user.it.uu.se/~ps/SAS– new.pdf (odwiedzona 26.08.20

[22] Tsihrintzis G. A., Sotiropoulos D. N., Jain L. C.: *Machine Learning Paradigms,* Springer, Szwajcaria, 2019

[23] Tobias J. V., Broen P., Carey S. W., Casper J., Chapman R., Costello J. M. et al.: *Proceedings of the Conference on the Assessment of Vocal Pathology*, Bethesda, Maryland, Kwiecień 1979

[24] Kramer M. A.: *An Introduction to Field Analysis Techniques: The Power Spectrum and Coherence*, Department of Mathematics and Statistics, Boston University, Boston, 2013

[25] Lessard C. S., Ph.D.: *Signal processing of random physiological signals*, Texas A&M University, College Station, Morgan & Claypool, USA, 2006

[26] Preumont A.: *Random Vibration and Spectral Analysis*, Kluwer academic publishers, Netherlands, 1994

[27] Dokumentacja funkcji *coherence* z biblioteki *scipy w języku Python.* Dostępny: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.coherence.html (odwiedzona 02.05.20)

[28] Dokumentacja funkcji spectral_flatness z biblioteki *scipy* w języku Python. Dostępny: https://librosa.org/librosa/generated/librosa.feature.spectral_flatness.html (odwiedzona 02.05.20)

[29] Dokumentacja funkcji *welch* z biblioteki *scipy* w języku Python. Dostępny: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.welch.html#scipy.signal.welch (odwiedzona 27.08.20)

[30] Wyjaśnienie terminu *framework*. Dostępny: https://techterms.com/definition/framework (odwiedzona 06.09.20)

[31] Kod źródłowy oraz opis frameworku JUCE: Dostępny: https://github.com/WeAreROLI/JUCE (odwiedzona 25.08.20).

[32] Dokumentacja klasy *Random* frameworku JUCE. Dostępny: https://docs.juce.com/master/classRandom.html (odwiedzona 25.08.20)

[33] Dokumentacja klasy *Range* frameworku JUCE. Dostępny: https://docs.juce.com/master/classRange.html (odwiedzona 25.08.20)

[34] Dokumentacja klasy AudioBuffer frameworku JUCE. Dostępny: https://docs.juce.com/master/classAudioBuffer.html (odwiedzona 25.08.20)

[35] *Implementing Circular Buffer in C*. Dostępny: https://embedjournal.com/implementing–circular–buffer–embedded–c/ (odwiedzona 25.08.20)

[36] Dokumentacja funkcji *addFromWithRamp()* klasy *AudioBuffer* frameworku JUCE. Dostępny: https://docs.juce.com/blocks/classAudioBuffer.html#a1853da41f2a3d87fa6291a2bf2adca05 (odwiedzona 25.08.20)

[37] Dokumentacja klasy *dsp::ProcessorDuplicator* frameworku JUCE. Dostępny: ://docs.juce.com/master/structdsp_1_1ProcessorDuplicator.html (odwiedzona 25.08.20)

[38] Dokumentacja klasy *dsp::Filter::IIR* frameworku JUCE. Dostępny: http://charette.noip.com:81/programming/doxygen/juce/classjuce_1_1dsp_1_1IIR_1_1Filter.html#details (odwiedzona 25.08.20)

[39] Dokumentacja klasy *dsp::IIR::Coefficients* frameworku JUCE. Dostępny: https://docs.juce.com/master/structdsp_1_1IIR_1_1Coefficients.html (odwiedzona 25.08.20)

[40] Dokumentacja struktury *dsp::ProcessSpec* frameworku JUCE. Dostępny: https://docs.juce.com/master/structdsp_1_1ProcessSpec.html (odwiedzona 25.08.20)

[41] Dokumentacja klasy *dsp::AudioBlock* frameworku JUCE. Dostępny: https://docs.juce.com/master/classdsp_1_1AudioBlock.html (odwiedzona 25.08.20)