Spis treści:

1. Wstęp

1.1. Cel i zakres pracy

2. Symulacja i percepcja zjawiska pogłosu

2.1. Główne metody realizacji

2.1.1. Pogłos splotowy

2.1.2. Rozwiązania elektromechaniczne

2.1.3. Cyfrowe symulacje pogłosu

2.2. Sposoby implementacji elementów pogłosu

2.3. Zjawiska psychoakustyczne przy percypowaniu pogłosu

3. Implementacja i właściwości istniejących rozwiązań

3.1. Opis wybranych parametrów statystycznych do analizy sygnałów

3.2. Testy wybranych sposobów realizacji elementów pogłosu

4. Propozycja i implementacja metody hybrydowej

4.1. Koncepcja realizacji

4.2. Architektura aplikacji

4.3. Połączenie testowanych rozwiązań w docelowy pogłos

4.3.1. Linie opóźniające i sposób obliczania aktualnego bufora audio

4.3.2. Tłumienie odbić

4.3.3. Filtracja linii opóźniających

4.3.4. Uprzestrzennienie pogłosu

4.3.5. Późny ogon pogłosowy

4.4. Część funkcjonalna aplikacji

5. Podsumowanie i wnioski

Bibliografia

**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 prowadzone pod koniec XX wieku wymagały skomplikowanego toru sygnałowego oraz zwykle potrzeby wykorzystania pola swobodnego i dyfuzyjnego do wykonywania testów lub badań [2]. Współczesne możliwości znacznie to ułatwiają. Przede wszystkim możliwość odwzorowania większości zjawisk w domenie cyfrowej, jak również większe możliwości obliczeniowe. W literaturze z tego okresu można zaobserwować obszerne opisy [2][12]. W odróżnieniu od tej sytuacji, 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, więkobliczeniowe pozwalają na większą ilość testów w tym samym czasie, a co za tym idzie możliwość przetestowania większej liczby rozwiązań.

W związku z powyższym otwierają się szersze możliwości eksperymentowania – w niniejszej pracy możliwość ta zostaje wykorzystana 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 zaproponowanych w ramach pracy parametrów.

Praca obejmuje przegląd głównych idei realizacji sztucznego pogłosu oraz percepcyjnych zjawisk przestrzennych, testy wybranych rozwiązań realizacji poszczególnych elementów pogłosu i ich ocena na podstawie zaproponowanych parametrów w sposób obiektywny, jak również implementacja docelowej aplikacji w formacie VST. Docelowa aplikacja będzie pogłosem w postaci efektu, 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**

Pogłos akustyczny składa się z opóźnionych i osłabionych kopii dźwięku bezpośredniego. Zawartość częstotliwościowa każdego z odbić zależy od kierunkowości źródła oraz pochłaniania odbijających powierzchni [dafx]. Zjawisko to występuje naturalnie w większości środowisk i ma mocniejszy efekt przy powierzchniach odbijających.

Pogłos to czas, po którym poziom natężenia dźwięku spadnie o 60 dB od wyłączenia źródła dźwięku [Drobner nowszy]. Fale akustyczne odbijają się od elementów wnętrza pomieszczenia z coraz to większą stratą energii [Drobner]. Fale emitowane ze źródła dźwięku zwane są dźwiękiem bezpośrednim, odbite od elementów pomieszczenia tworzą fale wtórne, które nakładają się na fale dźwięku bezpośredniego. Po każdym odbiciu dźwięku bezpośredniego następują odbicia wtórne – dzięki temu podczas emisji wzrasta energia akustyczna dźwięku odbitego. Po wyłączeniu źródła następuje faza zaniku dźwięku odbitego, wskutek zmniejszania się ciśnienia akustycznego [Oleszkowicz]. 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 [Drobner]:

T = 0,164 \* V/A

V – objętość pomieszczenia,

A – średni współczynnik pochłaniania wszystkich materiałów w pomieszczeniu,

T – czas pogłosu

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 [Drobner]. Skorygowany wzór nazywany jest wzorem Eyringa i przyjmuje następującą postać [Drobner nowszy]:

T = –0,0713\* V/(P\*log(1–alfa))

alfa – współczynnik pochłaniania

Pogłos może być także wytworzony w sposób sztuczny na wiele sposobów [28].

**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 odbijających, nierównoległych ścianach 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.

Jednym z najwcześniejszych rozwiązań pozwalających na zastosowanie pogłosu w nagraniach był pogłos płytowy. Jego konstrukcja opierała się na prostokątnej, stalowej, obramowanej płycie z przymocowanym przetwornikiem elektromechanicznym. Przetwornik rejestrował sygnał dźwiękowy, którego drgania były przenoszone na płytę. Następnie drgania płyty były odbierane przez przetwornik elektroakustyczny, tym sposobem był 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 [dafx].

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 drganiu 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 [dafx].

**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 pogłos 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 [dafx]. Definicja splotu przedstawia się jako zależność [29]:



gdzie:

f, g – sygnały poddawane operacji splotu

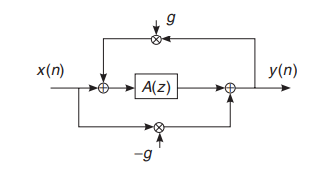
Ze względu na małą wydajność obliczeniową operacji splotu w dziedzinie czasu, stosuje się podejście wykorzystujące tożsamość operacji splotu w dziedzinie czasu z operacją mnożenia w dziedzinie częstotliwości. Wykonywanie obliczeń w dziedzinie częstotliwości jest znacznie bardziej wydajne obliczeniowo. Często jest to jedyne możliwe rozwiązanie w przypadku konieczności działania algorytmu w czasie rzeczywistym [dafx].

**2.1.2. Rozwiązanie oparte o filtry wszechprzepustowe i grzebieniowe**

Jednymi z najważniejszych, pionierskich prac dotyczących realizacji sztucznego pogłosu są prace M. Schroeder'a z lat 60 XX wieku [3][4]. Zaproponował on podejście wykorzystujące filtry wszechprzepustowe i grzebieniowe jako narzędzie umożliwiające symulację pogłosu. Filtr wszechprzepustowy przedstawia się jako zależność [dafx]:



gdzie m oznacza długość opóźnienia w próbkach.



Rys. 2.2. Struktura filtru wszechprzepustowego [dafx].

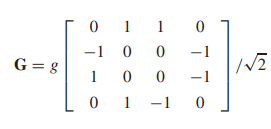
W schemacie filtru wszechprzepustowego przedstawionego na Rys. 2.2., współczynnik g oznacza wzmocnienie, a A(z) linię opóźniającą. Zastosowanie szeregu takich filtrów pozwala na uzyskanie odpowiedzi impulsowej o płaskiej charakterystyce [30]. Podejście oparte o filtry wszechprzepustowe krótkim czasie stało się standardem w niemal wszystkich stosowanych algorytmach sztucznego pogłosu [dafx].

**2.1.2. Feedback Delay Network**

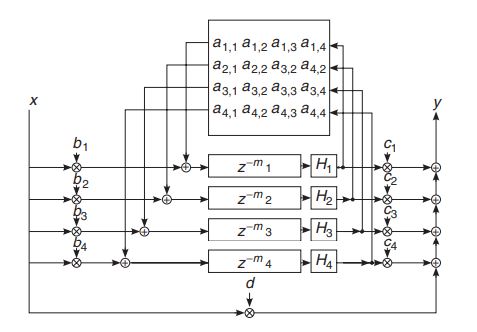
W 1982 roku J. Stautner i M. Puckette jako strukturę realizującą pogłos zaproponowali sieć linii opóźniających połączonych w pętli sprzężenia zwrotnego [5]. W późniejszym czasie struktura taka została nazwana siecią Feedback Delay Network. Pierwotnie zaproponowana sieć miała postać [dafx]:



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



Dzięki zastosowaniu sieci pogłosowej FDN uzyskać można wysoką gęstość późnego ogona pogłosowego [dafx]. 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].



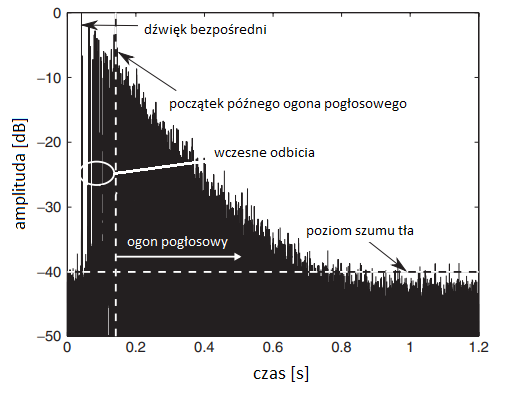
Rys. 2.3. Schemat podstawowej sieci pogłosowej FDN.

Badania nad rozwojem sieci FDN zostały rozwinięte w pracy Jota [1].

**2.1.3. 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][dafx] (Rys 2.1.). Wczesne odbicia to część pogłosu złożona z dyskretnych odbić imitujących wczesne odbicia od ścian lub innych elementów w pomieszczeniu. O rozpoznaniu kierunku, z którego dochodzi dźwięk decyduje pierwsza fala dźwiękowa. Odbicia dźwięku przychodzące w ciągu pierwszych 30 ms od pierwszej fali wpływają słabo na odczucie kierunku [11]. 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 warunki akustycznymi, jakie panowałyby w rzeczywistych warunkach. 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 [1] 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. Badania wskazały, iż zjawisko zachodzi dla nie tylko sygnałów takich jak mowa czy muzyka, ale także dla bardzo krótkich odcinków szerokopasmowego szumu Gaussa [?].



Rys. 2.1. Schematyczne przedstawienie odpowiedzi impulsowej z wyszczególnieniem dźwięku bezpośredniego, wczesnych odbić oraz późnego ogona pogłosowego [dafx].

**2.2. Sposoby implementacji elementów pogłosu**

A). Wzmocnienia linii opóźniających i poszczególnych odbić

Barron [2] 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. To zaobserwowane zjawisko miało wynikać z właściwości wczesnych odbić bocznych w pomieszczeniu (10 – 80 ms). Na podstawie jego badań został także wysunięty wniosek, iż wrażenie przestrzenności jest tym większe, im stosunek docierających do odbiorcy bocznych do pozostałych. Ze względu na najwyższy poziom dźwięku, odbicia te są najbardziej znaczące w percypowaniu pogłosu. Zjawisko „wrażenia przestrzenności” odczuwane przez osoby badane pojawiało się dla symulacji odbić bocznych w zakresie powyżej „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 [1], 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 [3] zwraca uwagę, iż jedną z fundamentalnych zasad przy tworzeniu sztucznego pogłosu jest zadbanie o to, aby amplituda wszystkich składowych zanikała jednakowo prawie jednakowo szybko. Ma to na celu zapewnienie jednakowego opadania różnych składowych częstotliwościowych zanikającego dźwięku.

B). Czasy opóźnienia linii opóźniających

Dobór czasów opóźnienia linii opóźniających nie jest kluczowy, pod warunkiem, że liczby je reprezentujące nie są przez siebie podzielne [1]. Dobór nieskorelowanych wartości jest niezbędny, aby uniknąć zjawiska echa trzepoczącego i uzyskać płaską charakterystykę częstotliwościową [5]. Aby uniknąć zmniejszenia gęstości odbić oraz superpozycji, pożądane jest użycie liczb nieproporcjonalnych do siebie [4].

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 [5] 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 ilość długich linii opóźniających, jednak wtedy trudniej kontrolować gęstość pogłosu [5]. Zjawisko to ma wytłumaczenie w interferencji między dźwiękiem bezpośrednim, a opóźnionym o niewielkie wartości, co skutkuje pojawieniem się filtru grzebieniowego. Wskazuje na fakt, iż zjawisko ma miejsce dla opóźnień około 10–50ms, a szczególnie w okolicach 20ms [2].

C). Filtracja linii opóźniających

Schroeder zaproponował [4] powiązanie wartości tłumienia linii opóźniających z ich zawartością częstotliwościową. Miało to na celu odwzorowanie faktu dłuższego czasu pogłosu dla niskich częstotliwości. Dodatkowo, liczba powtórzeń dźwięku przy powstaniu filtru grzebieniowego powstałego przy zsumowaniu dźwięku bezpośredniego z jego kopią przesuniętą o 40ms to 25. Sytuacja ta powoduje znaczne zagęszczenie ilości odbić w jednostce czasu.

J. Stautner i M. 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 [5].

Aby pogłos brzmiał naturalnie, pożądane jest, aby charakterystyka częstotliwościowa była płaska, lecz nie zbyt płaska [8].

Na trudność obiektywnej oceny jakości pogłosu wskazują [5]. 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.

"Perhaps a statistical method of choosing the early reflection properties, combined with some perceptually meaningful constraints, can be found."– [5].

Istnieją następujące subiektywne sposoby na identyfikację niewystarczającej gęstości widmowej późnego pogłosu [1]:

* odpowiedź na sygnał impulsowy sygnał 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 jakości w zależności od rodzaju filtracji.

D). Późny ogon pogłosowy

**2.4. 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 [11].

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 [11].

Dwie najważniejsze wielkości opisujące słyszenie przestrzenne oraz możliwość lokalizacji źródła dźwięku przez człowieka to [11]:

* ITD. (*Interaural Time Difference*) oraz ILD (*Interaural Level Difference).* Zjawisko ILD 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.
* ITD to międzyuszna 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 ich obu dla różnych zakresów częstotliwości opisuje teoria *Duplex Theory* i została zaproponowana przez Reyleigha w 1907 roku [17].

W warunkach rzeczywistych, odbiorca dźwięku ma także do dyspozycji możliwość ruchów głową, dzięki którym 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) [11].

*Większość badań naukowych dotyczących pogłosu, zarówno teraźniejszych jak i przeszłych, jest badaniami subiektywnymi – są przeprowadzane na grupie ludzi, którzy zostają poddani testom słuchowym. Ich zadaniem jest podanie badającemu informacji np. o tym, w którym momencie dźwięk zaczął być słyszalny, lub czy słyszą różnicę między dwoma sygnałami.*

Według Beranek [6] najważniejszym wyznacznikiem jakości pogłosu jest czas pierwszego odbicia w stosunku do dźwięku bezpośredniego i jest to nawet ważniejsze, czas pogłosu w pomieszczeniu, a optymalna wartość tego parametru to czas poniżej 20ms.

Zostało także stwierdzone, że stopień wrażenia przestrzenności jest związany ze stopniem niekoherencji sygnałów z obu kanałów [7].

Efekt związany z lokalizacją dźwięku przez czas pierwszych odbić nie występuje, jeżeli odstęp czasowy między pierwszą falą, a pierwszym odbiciem jest krótszy niż 1 ms, w takim przypadku źródło dźwięku jest lokalizowane w położeniu pomiędzy kierunkami obu tych dźwięków [?].

W zależności od rodzaju sygnału mogą zachodzić różne zjawiska percepcyjne takie jak: dwuuszne odmaskowanie, dwuuszne dudnienia, efekt pierwszeństwa i inne [11].

**3. Ocena jakości symulacji pogłosu**

Obiektywna ocena jakości pogłosów 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 (w podrozdziale 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. Następnie dokonana zostanie 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 rozdziale 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.

A). Widmowa gęstość mocy – 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 [19]:



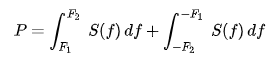
Jeśli sygnał może być potraktowany jako sygnał stacjonarny, to widmowa gęstość mocy jest transformatą Fouriera



funkcji autokorelacji tego sygnału, co w rezultacie daje [20]:

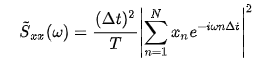


Moc sygnału dla danego pasma częstotliwości można wyliczyć wykonując całkowanie widmowej gęstości mocy po dodatnich i ujemnych częstotliwościach [20]:



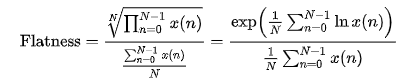
gdzie:  
P – moc sygnału,  
[F1, F2] – przedział częstotliwości   
S(f) – widmowa gęstość mocy sygnału

Definicja widmowej gęstości mocy może być przełożona do dziedziny dyskretnej, wtedy wzór przyjmuje formę [19]:



gdzie:  
Sxx(ω)–widmowa gęstość mocy,  
T = NΔt,  
n – kolejna próbka sygnału,  
N – liczba próbek sygnału,   
t – czas.

B). Płaskość widmowa – parametr określający, jak bardzo sygnał zbliżony jest do sygnału szumowego, w opozycji do sygnału tonalnego. Wysoka wartość płaskości widmowej (bliska 1.0 dla szumu białego) oznacza, że widmo ma podobną moc we wszystkich pasmach częstotliwości. Płaskość widmowa jest zdefiniowana jako stosunek średniej geometrycznej widma mocy sygnału do średniej arytmetycznej widma mocy sygnału [21]:



Wielkość ta często jest przedstawiana w decybelach, gdzie wartość szczytową parametru przyjmuje się jako 0 dB [21].

C). Koherencja sygnałów:

,

gdzie Gxy(f) jest wzajemną gęstością widmową [22] między sygnałami x i y, a Gxx(f) i Gyy(f) są gęstościami widmowymi sygnałów samych z sobą. Wielkość ta opisuje relację pomiędzy dwoma sygnałami … [24].

**3.2. Testy stosowanych rozwiązań implementacji pogłosu**

**3.2.1. Opis implementowanych w ramach testów wersji pogłosu**

Implementacje poniższych symulacji pogłosu zostały zrealizowane korzystając z frameworku JUCE w języku C++ – dokładny opis implementacji oraz uzasadnienie wyboru narzędzia znajdują się w rozdziale 4.

Warianty pogłosów zaimplementowane przez autora:

A). W celu zbadania wpływu doboru czasów opóźnienia na brzmienie pogłosu, dokonano symulacji zjawisk następujących pogłosów:

* 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 [1][4][5]),
* pogłos złożony z linii opóźniających o czasach opóźnienia zawierających się w zakresie 1:1.5 (w nawiązaniu do [4]).

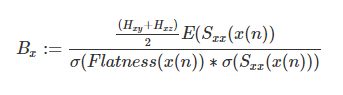
B). Filtracja linii opóźniających:

* filtracja dolnoprzepustowa każdej linii opóźniającej w jednakowy sposób,
* filtracja dolnoprzepustowa o częstotliwości odcięcia tym niższej, im wyższa wartość opóźnienia linii opóźniającej (w nawiązaniu do [4]),
* 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 [5]).

C). Późny ogon pogłosowy:

* 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 [1][3][4] podkreślenia wpływu wysokiej gęstości oraz płaskości widma na jakość pogłosu. Zdefiniowany przez autora parametr przedstawia się jako zależność:

 (5)

gdzie:

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

*x* – sygnał pogłosowy zawierający linie opóźniające o czasach opóźnienia będącymi liczbami wzajemnie pierwszymi

*y* – sygnał pogłosowy zawierający linie opóźniające o czasach opóźnienia będącymi liczbami losowymi

*z* – sygnał pogłosowy zawierający linie opóźniające o czasach opóźnienia będącymi liczbami losowymi z zakresu liczb o stosunku 1:1.5

*σ –* średnie odchylenie standardowe

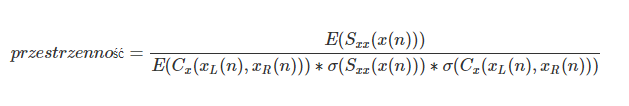
*E* – średnia arytmetyczna

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 w tezy o większej przestrzenności przy wyższym poziomie tych odbić (w nawiązaniu do [2]),
* pogłos z zastosowaniem różnicy czasu opóźnienia między kanałami dla linii opóźniających (w nawiązaniu do [11]),
* pogłos z zastosowaniem różnicy amplitudy pomiędzy kanałami dla linii opóźniających (w nawiązaniu do [11]),
* pogłos z zastosowaniem różnicy pomiędzy filtracją linii opóźniających obu kanałów (w nawiązaniu do [7]).

Jak opisano w podrozdziale 2.2, stopień wrażenia przestrzenności związany jest ze stopniem niekoherencji sygnałów z obu kanałów. Dlatego też, 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 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 *przestrzenność*. Zdefiniowanie parametru jest odpowiedzią na powtarzające się w literaturze podkreślanie istotności cech sygnału wpływające na jakość pogłosu [1][3][4] oraz jego przestrzenność [2][7]. Cechy te to: wysoka gęstość widma oraz stopień niekoherencji sygnałów z obu kanałów. Zdefiniowany przez autora parametr *Pr* przedstawia się jako:



gdzie:

*σ* – średnie odchylenie standardowe  
Sxx – widmowa gęstość mocy   
Cxy – koherencja lewego i prawego kanału sygnału   
E– średnia arytmetyczna  
xL(n)–lewy kanał sygnału x(n)  
xR(n)–prawy kanał sygnału x(n)

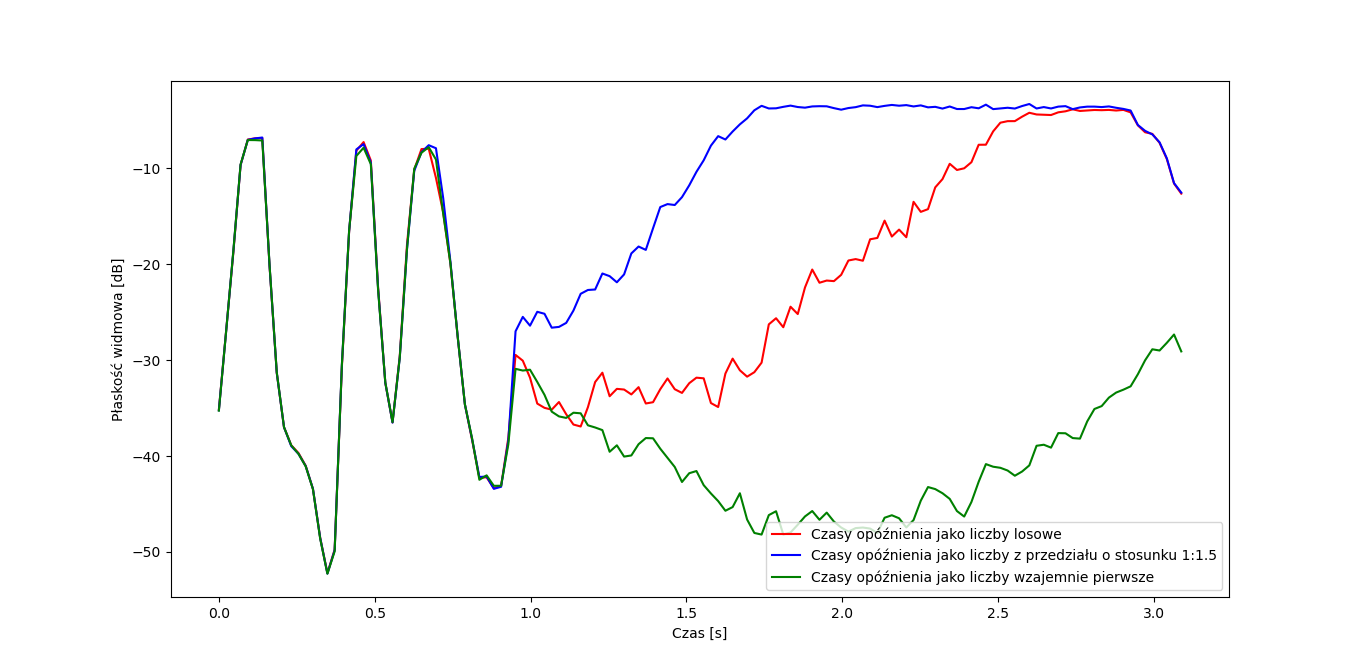
Do wyznaczenia parametrów: koherencja, widmowa gęstość mocy oraz płaskość widma zostały wykorzystane funkcje z biblioteki *scipy* w języku Python, odpowiednio: *coherence* [25], *spectral\_flatness* [26], *welch* [27].  
//tutaj będzie opisana szczegółowo metodologia wyznaczania wartości parametrów (parametr B oraz Pr)

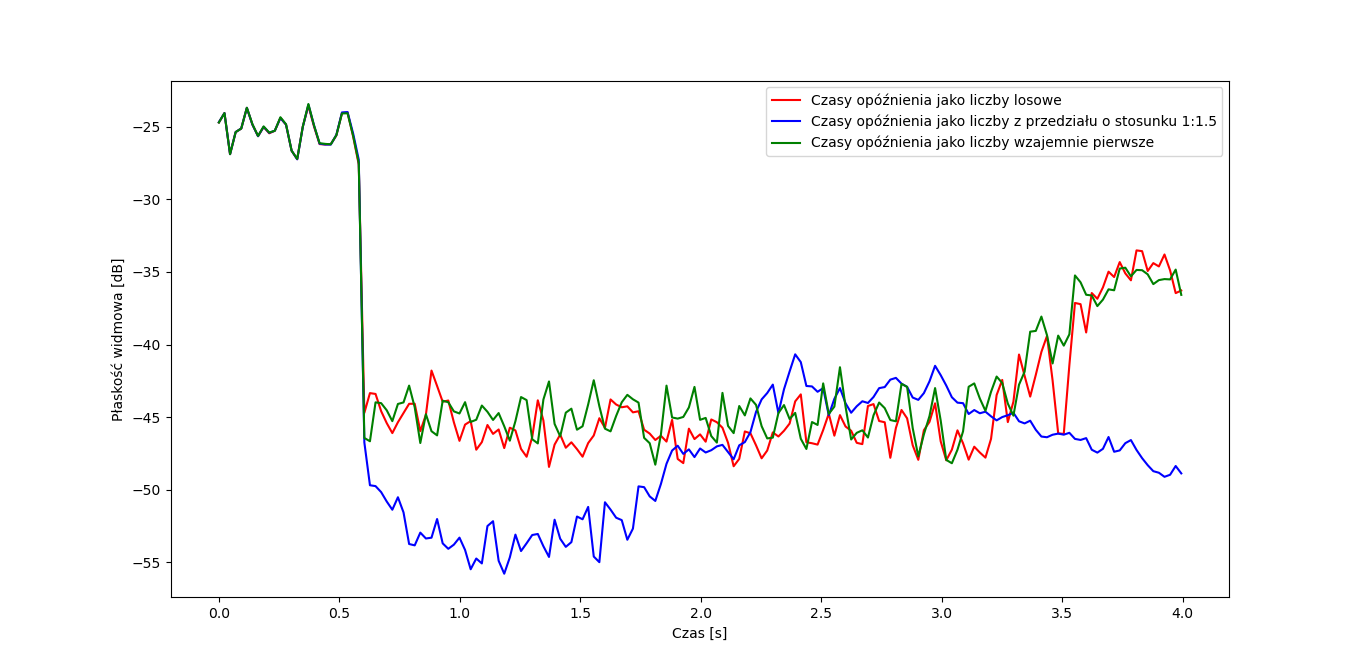
**3.2.2. Wyniki porównań wersji pogłosów oraz interpretacja wyników.**

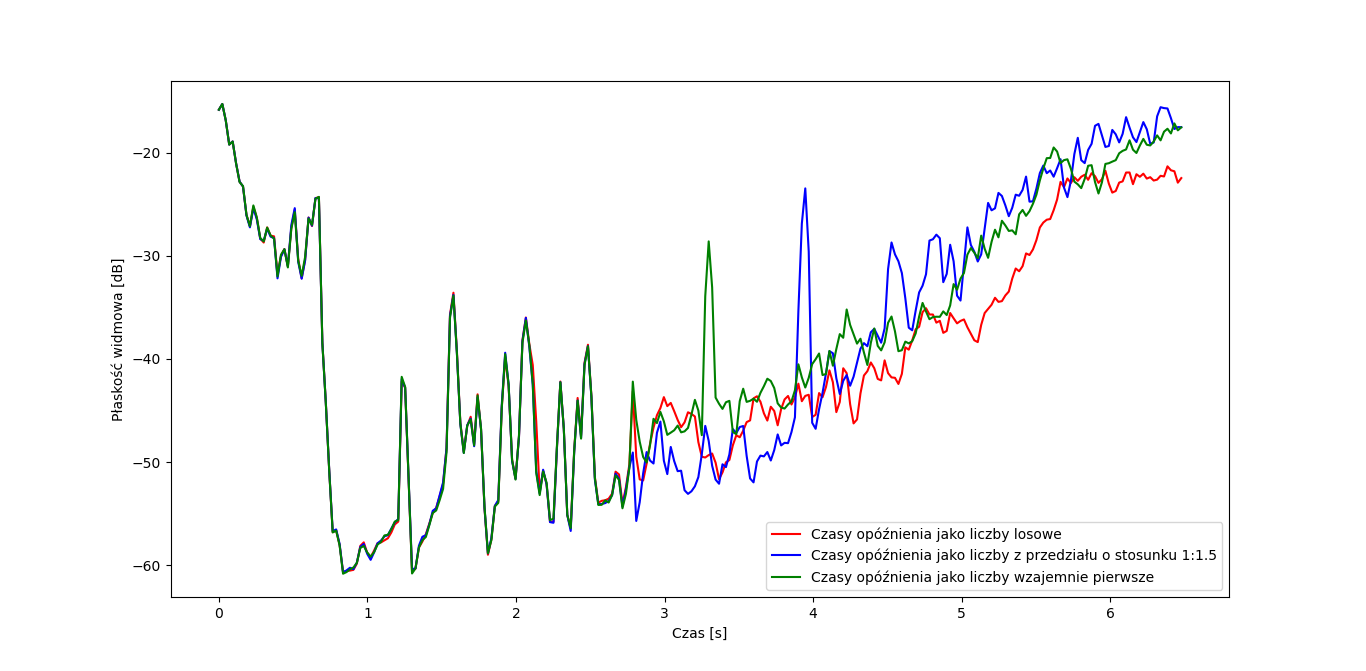
Wszystkie poniższe wykresy płaskości widma od czasu zawierają fragment sygnału przed zakończeniem czasu trwania dźwięku bezpośredniego. Po jego zakończeniu sygnały wyraźnie zaczynają się rozbiegać, co ukazuje ich odmienność. Podczas trwania sygnału przed wyłączeniem źródła wykresy pokrywają się. Jednak niewielkie różnice między nimi istnieją, co wynika z faktu, iż podczas jego trwania sygnał posiada już w sobie sygnał pogłosowy. Najbardziej użyteczne do oceny dane znajdują się już po wyłączeniu źródła, gdzie znajduje się sam sygnał pogłosowy.

A). Dobór czasów opóźnienia.

Na rys. 3.1. można zauważyć, iż dla sygnału mowy z liniami opóźniającymi o czasach opóźnienia będącymi liczbami zawierającymi się z zakresie 1:1.5, płaskość widma przez większość czasu trwania sygnału jest największa. Parametr ten ma bezpośredni wpływ na wartość parametru B będący wyznacznikiem jakości pogłosu (wzór 5). Wartości parametru B przy czasów linii opóźniających przedstawiono w tab. 3.1. W przypadku tym, nawet bez stosowania jednoliczbowych parametrów można stwierdzić, która wersja sygnału najbardziej odpowiada oczekiwaniom. Parametry te są natomiast szczególnie przydatne w nieoczywistych przypadkach, 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.

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

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

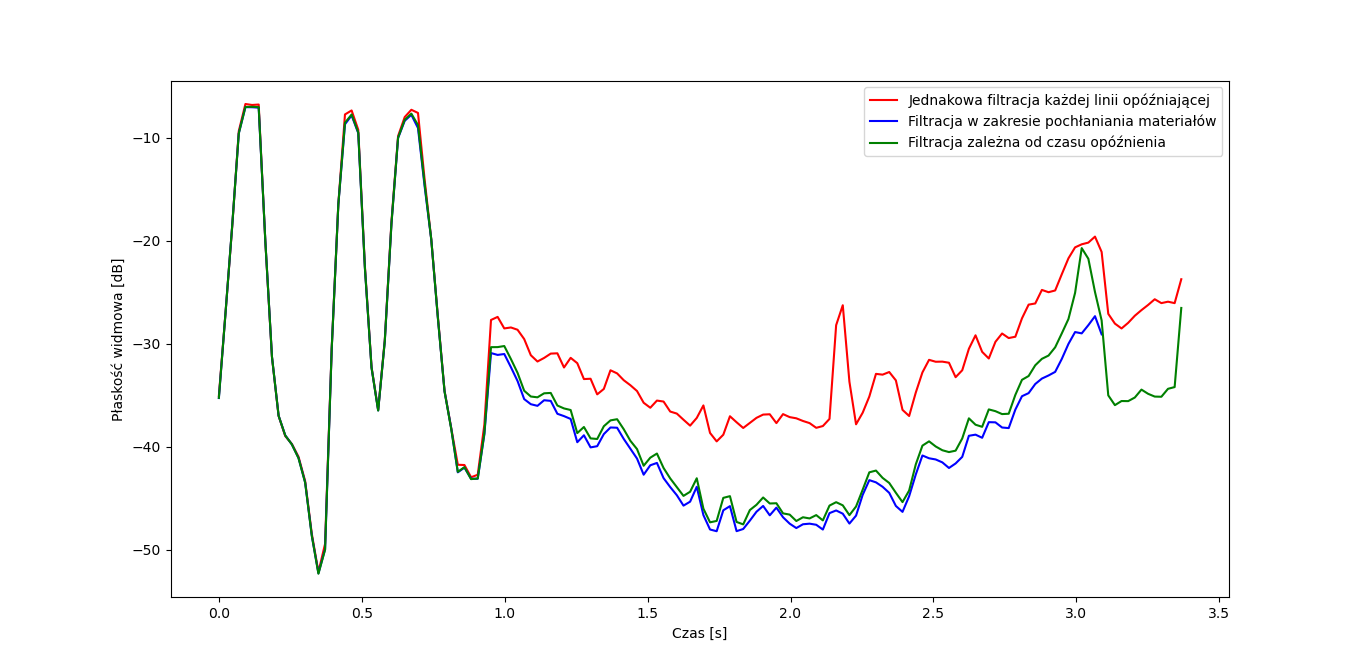
Rys. 3.3. Płaskość widmowa dla różnego rodzaju generowania linii opóźniających – gitara klasyczna.

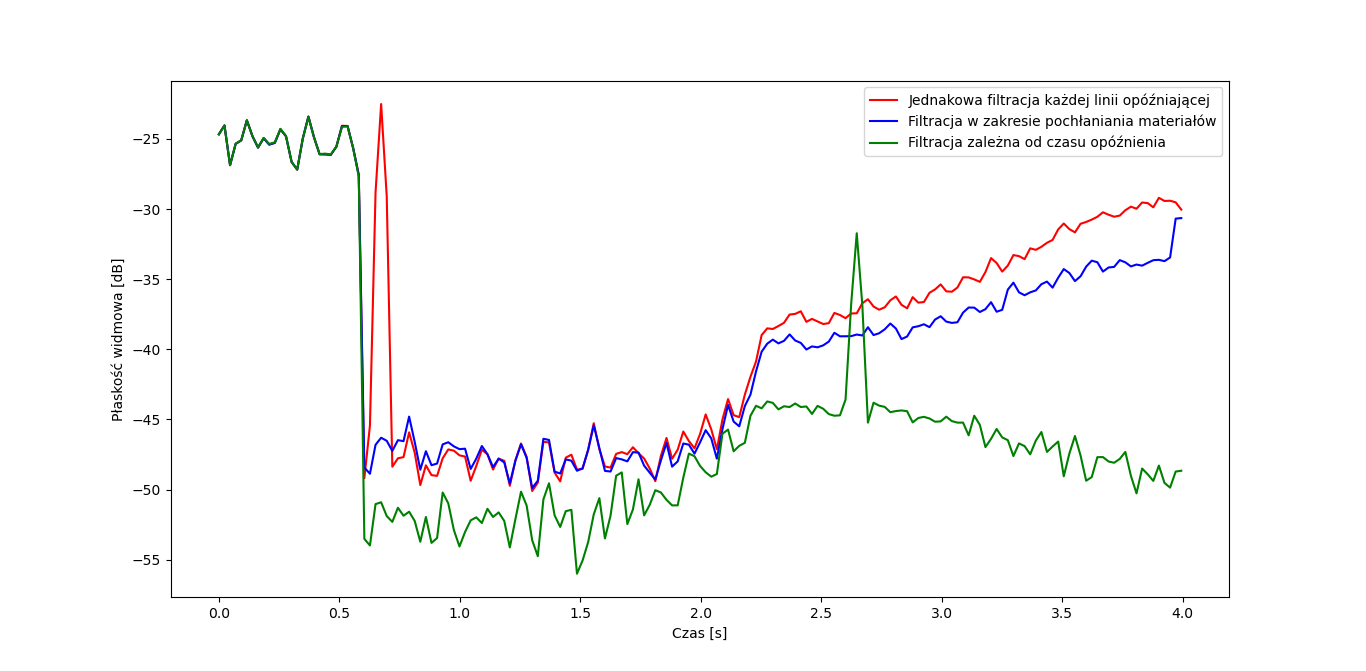
Tab. 3.1. Wartości parametru B dla różnego rodzaju generowania czasów linii opóźniających.

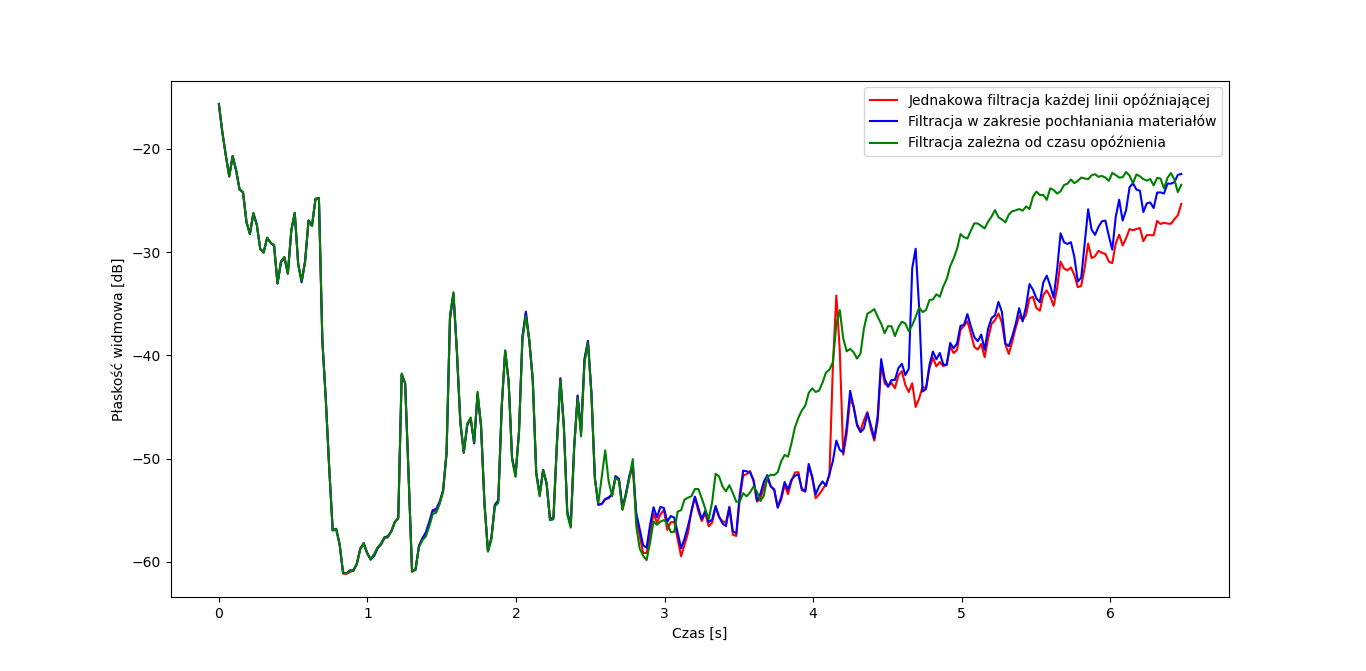
|  |  |  |  |
| --- | --- | --- | --- |
| B | Losowe czasy opóźnienia | Czasy opóźnienia jako liczby pierwsze | Czasy opóźnienia jako liczby zawierające się w zakresie liczb w stosunku 1:1.5 |
| Sygnał mowy | 0,57 | 0,22 | 1,0 |
| MLS | 0,64 | 1,0 | 0,39 |
| Gitara klasyczna | 0,80 | 1,0 | 0,98 |

B). Filtracja linii opóźniających:

Porównania te

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

Rys. 3.4. Płaskość widma symulacji różnych wariantów filtracji – sygnał MLS

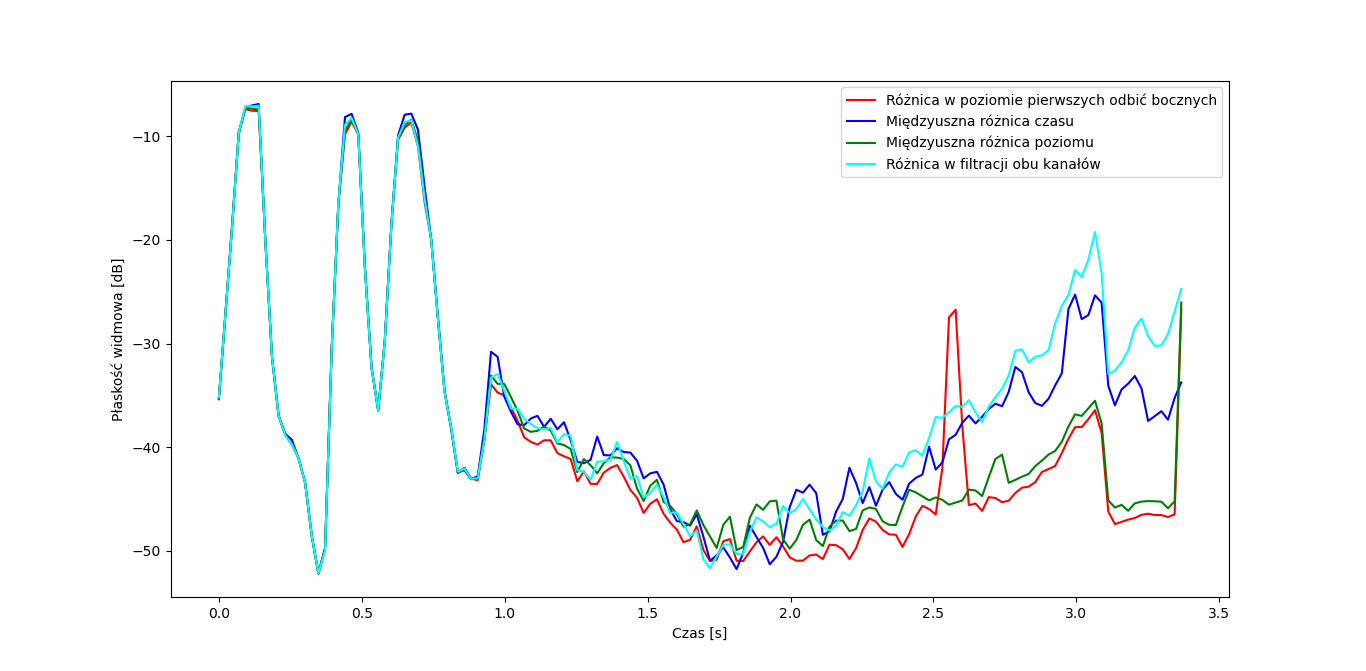
Rys. 3.4. Płaskość widma symulacji różnych wariantów filtracji – gitara klasyczna

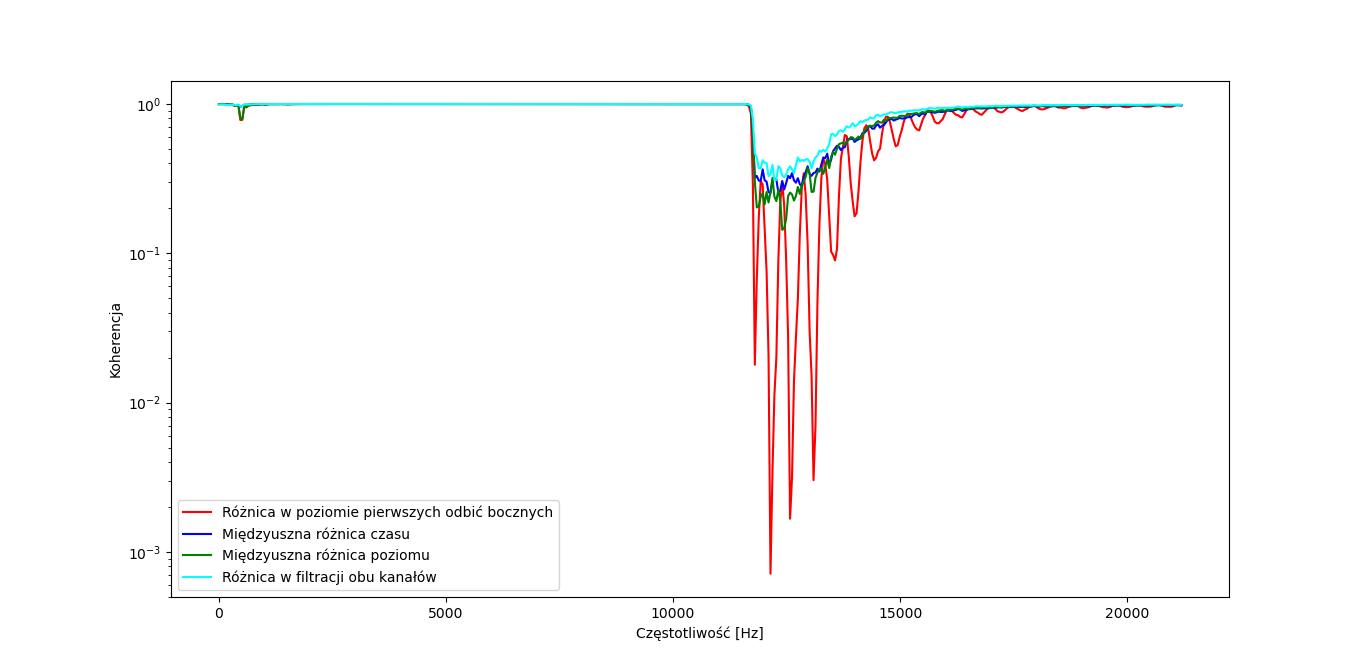
Tab. 3.2. Wartości parametru B dla pogłosów z różnymi wariantami filtracji

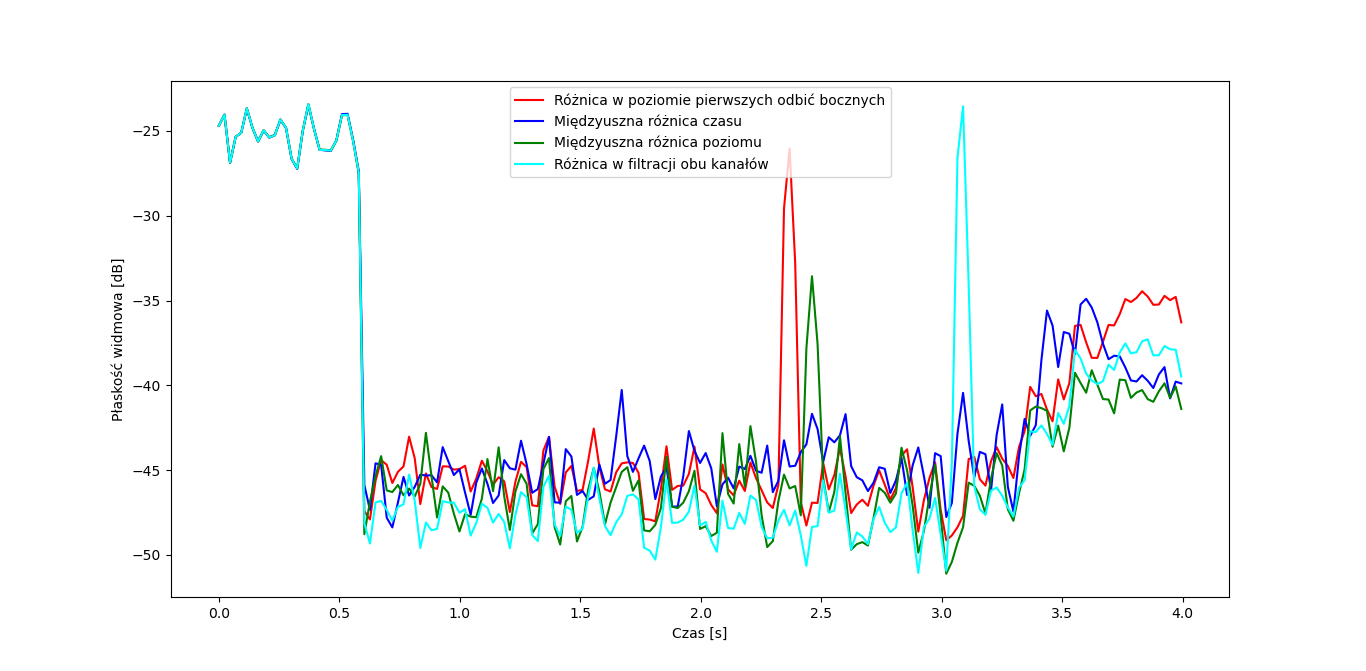
|  |  |  |  |
| --- | --- | --- | --- |
| B | Filtracja każdej linii w jednakowy sposób | Filtracja w zakresie pochłaniania materiałów | Tym niższa częstotliwość odcięcia filtru, im wyższy czas opóźnienia |
| Sygnał mowy | 1,0 | 0,41 | 0,07 |
| MLS | 1,0 | 0,79 | 0,11 |
| Gitara klasyczna | 0,47 | 0,92 | 1,0 |

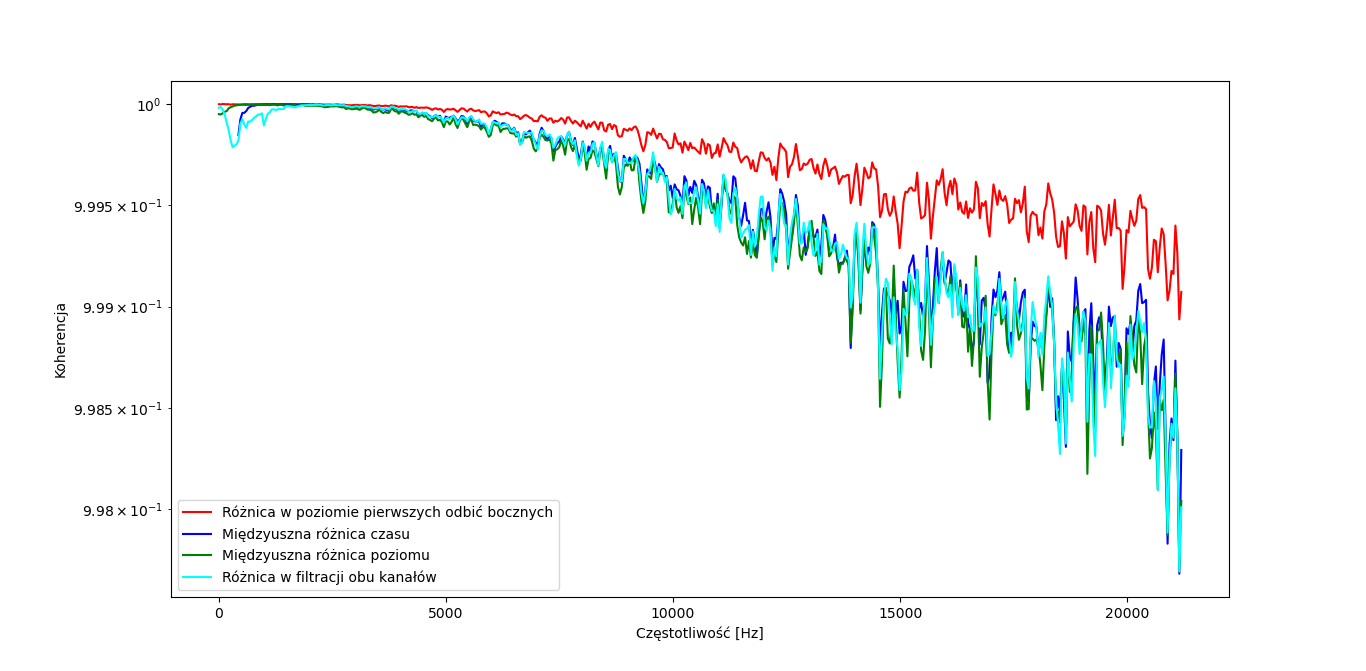
C). Zjawiska przestrzenne:

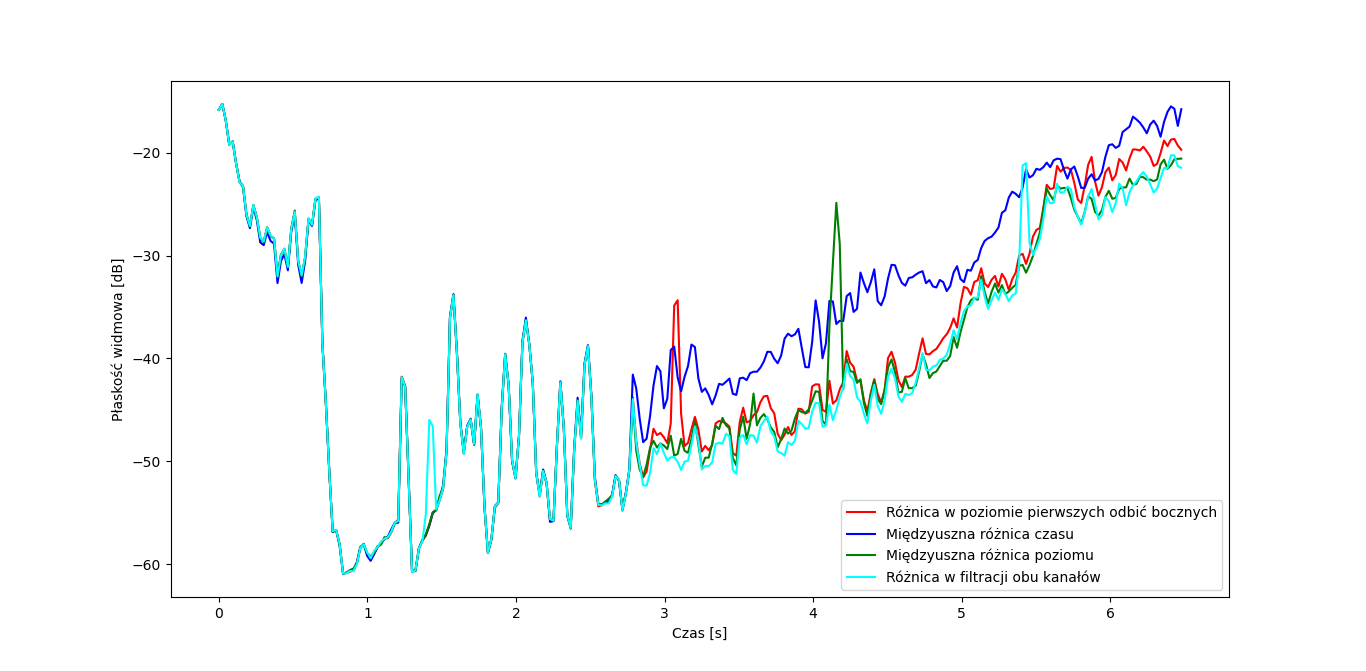
asd

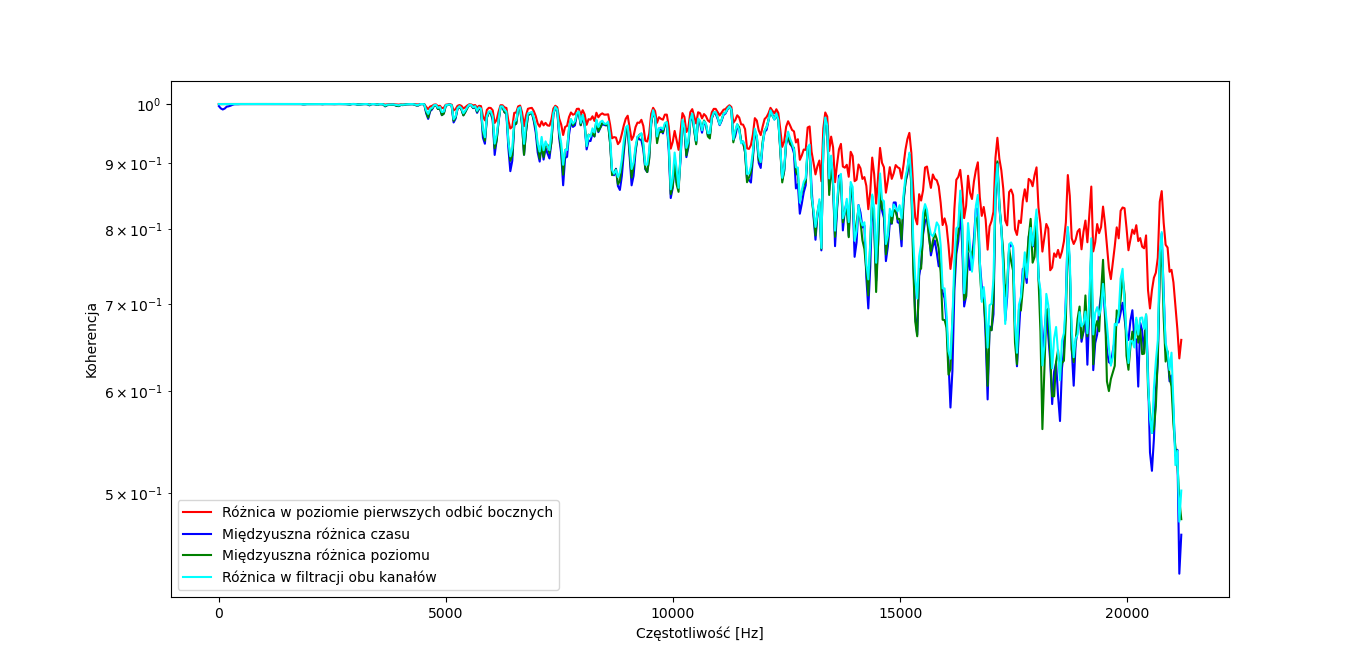
Rys. 3.7 Płaskość widmowa dla różnych zjawisk przestrzennych – sygnał mowy.

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

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

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

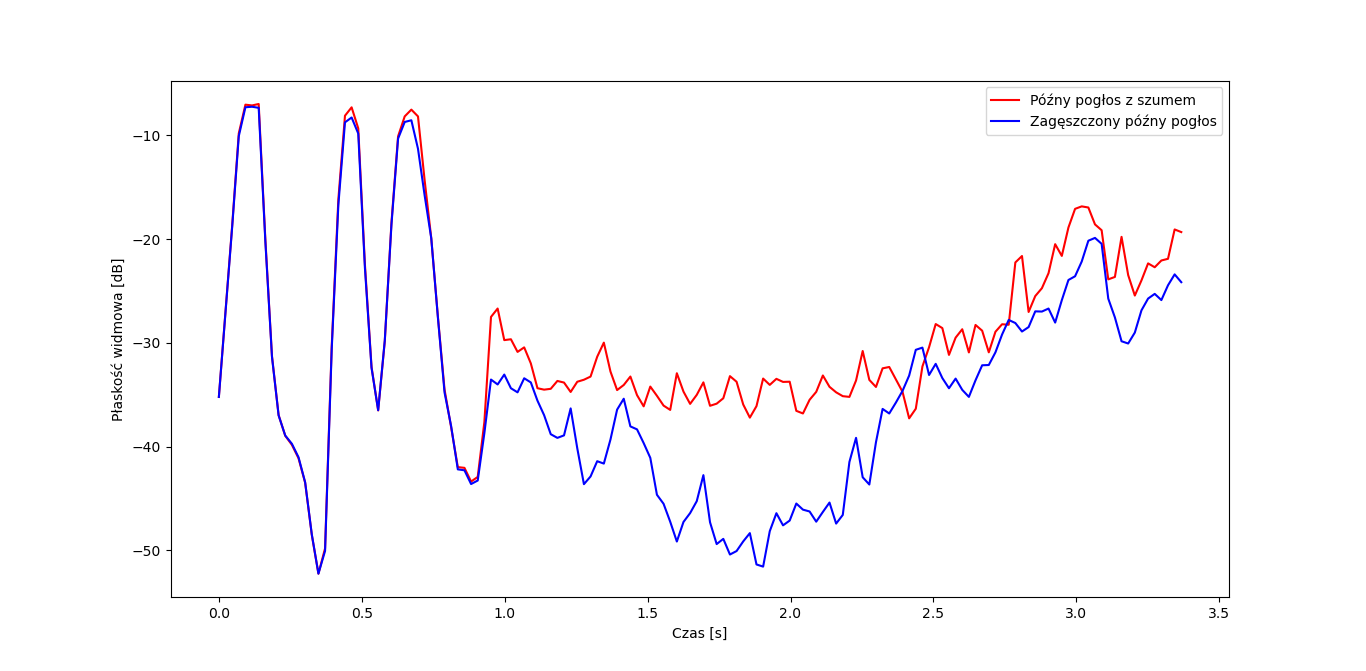
Rys. 3.7 Płaskość widmowa dla różnych zjawisk przestrzennych – gitara klasyczna.

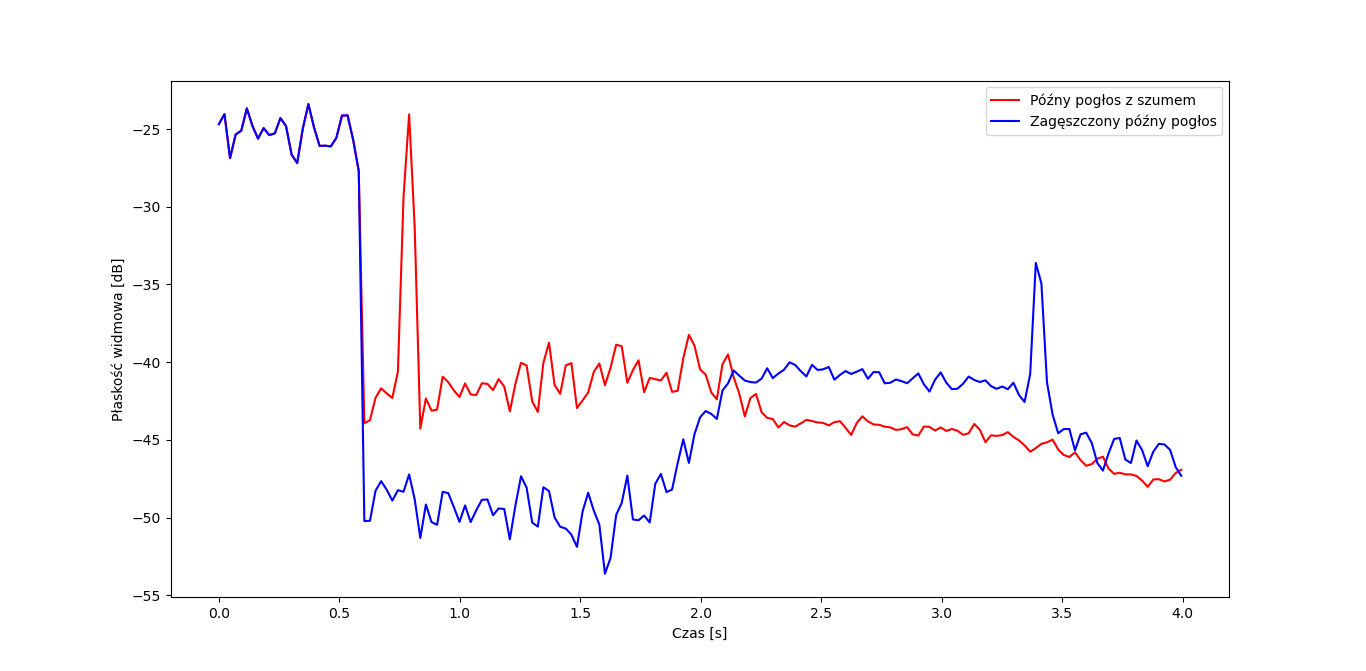
Rys. 3.10 Koherencja między kanałami dla różnych zjawisk przestrzennych – gitara klasyczna.

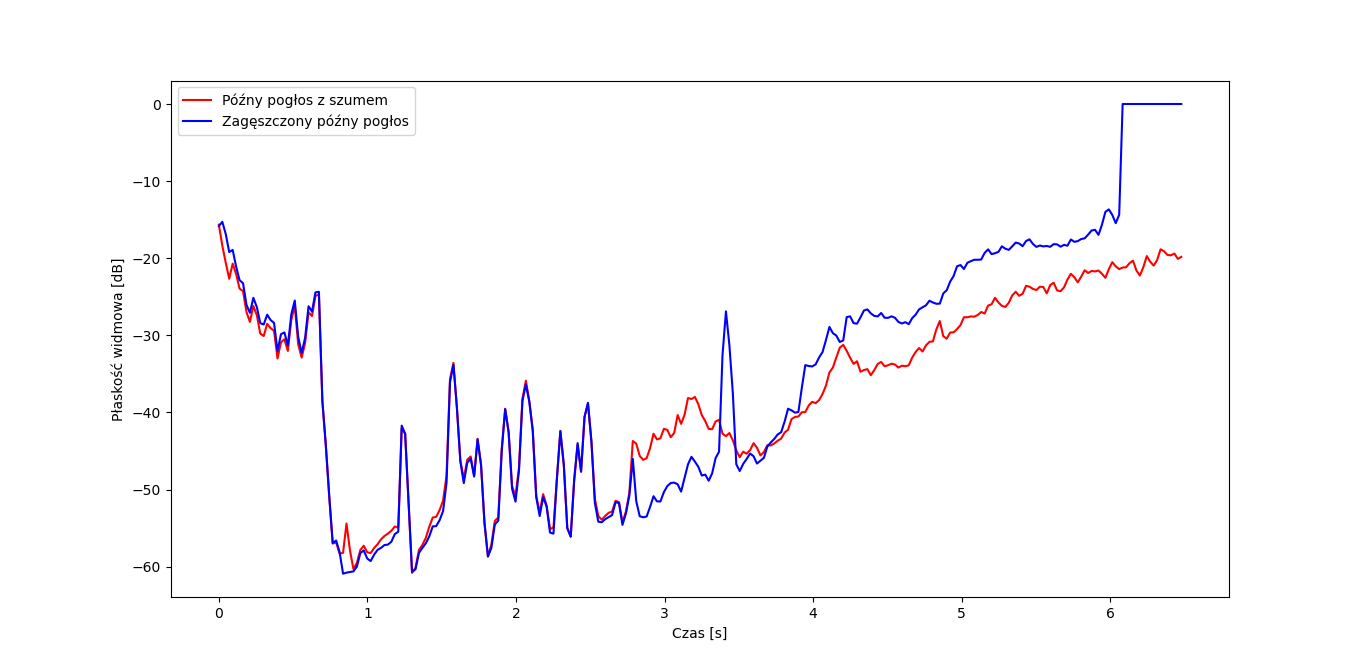
Tab. 3.3 Wartości parametru Pr dla symulacji różnych zjawisk przestrzennych

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Pr | ITD | Różna filtracja obu kanałów | ILD | 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 |

D). Późny ogon pogłosowy:

  
Rys 3.7 Płaskość widmowa dla obu wersji implementacji późnego pogłosu – sygnał mowy.

Rys 3.7 Płaskość widmowa dla obu wersji implementacji późnego pogłosu – sygnał MLS.

Rys 3.7 Płaskość widmowa dla obu wersji implementacji późnego pogłosu – gitara klasyczna

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

|  |  |  |
| --- | --- | --- |
| B | Późny pogłos z szumem | Późny pogłos zagęszczony odbiciami |
| Sygnał mowy | 1,0 | 0,086 |
| MLS | 0,98 | 1,0 |
| Gitara klasyczna | 0,84 | 1,0 |

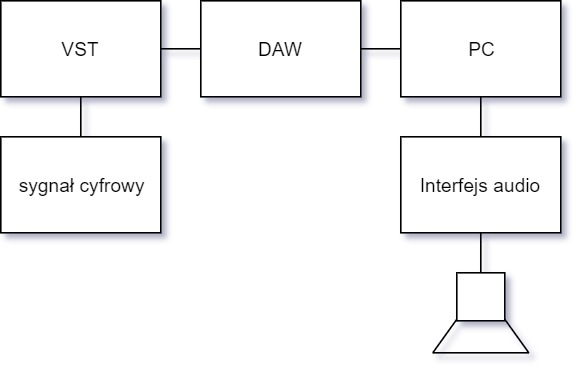
**4. Implementacja docelowego pogłosu**

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 JUCE. Jest to wieloplatformowy otwarto–źródłowy framework pozwalający na realizację aplikacji m. in. w technologii VST w języku C++ [16]. 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**

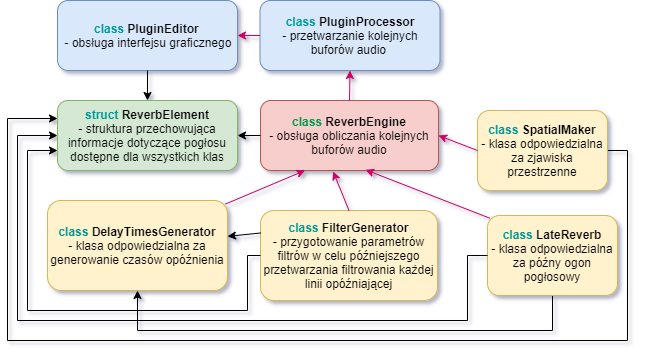
Projekt zakłada stworzenie aplikacji w formacie VST pozwalającej na modyfikację sygnału przypisanego do ścieżki w oprogramowaniu typu DAW (*Digital Audio Workstation*)*.* Modyfikacja ta to efekt sztucznego pogłosu będącego wynikiem implementacji w niniejszej pracy. Stworzona wtyczka VST posłuży jako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

  
Rys. 4.1. Schemat toru przetwarzania sygnału

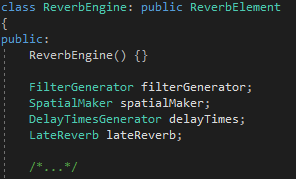
Implementacja docelowego pogłosu zakłada uwzględnienie wyników badań opisanych w rozdziale 3 i na ich podstawie zastosowania wybranych elementów w celu uzyskania hybrydowego rozwiązania.

**4.2. Opis architektury 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.

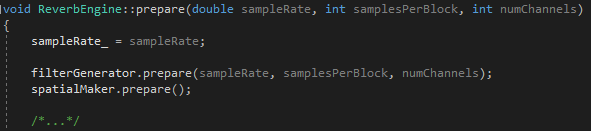
Rys. 4.1 Schemat przedstawiający hierarchię oraz opis klas w programie

Na Rys. 4.1. 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 dziedziczących klas. Pozostałe strzałki oznaczają, iż polem klasy, do której skierowana jest strzałka jest obiekt klasy, z której strzałka wychodzi. Rys 4.-. przedstawia zastosowanie mechanizmu tego na przykładzie klasy *ReverbEngine.*

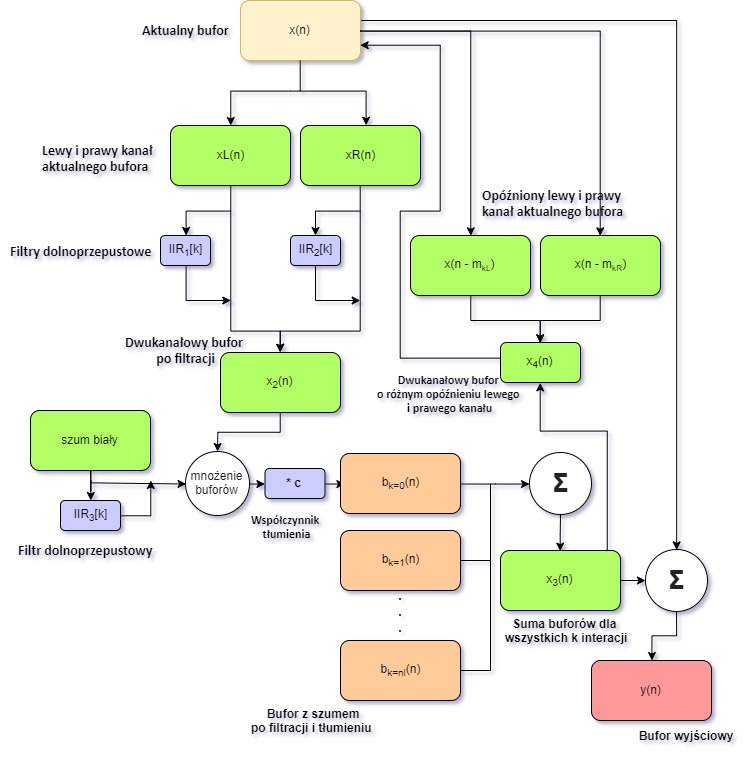
  
Rys. 4.-. Definicje zmiennych o typach stworzonych klas, w głównej klasie odpowiedzialnej za obliczenia dotyczące sygnału.

**4.3. Połączenie wybranych rozwiązań w docelowy pogłos**

Przetwarzanie sygnału w trakcie działania programu wiąże się z dużą złożonością obliczeniową. Dlatego też, duża część obliczeń prowadzących do uzyskania wyjściowego sygnału wykonywania jest przed rozpoczęciem przetwarzania kolejnych buforów sygnału. 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.-). Następnie informacje te przesyłane są do klasy bibliotecznej odpowiedzialnej za właściwe przetwarzanie sygnału.

  
Rys. 4.-. Przygotowanie

W niniejszym podrozdziale znajduje się szczegółowy opis działań prowadzących do uzyskania sygnału wyjściowego. Schematyczne przedstawienie opisanych działań tych znajduje się na Rys. 4.2.

Rys. 4.2 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. Losowanie wartości czasów opóźnienia odbywające się w trakcie przetwarzania sygnału byłoby znacznie bardziej złożone obliczeniowo wylosowanie ich jeden raz. Gdyby wartości te losowane były w czasie przetwarzania sygnału, mamy 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 wbudowana *JUCE::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. 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 klasę *DelayTimesGenerator*.

*//jak będzie mi się chciało, to tutaj tabela z wartościami tego czasu (to 12 ms) dla różnych wielkości bufora i f. próbkowania*

B). Wariant pogłosu złożonego z linii opóźniających o czasach opóźnienia będącymi 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 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, znaleziona maksymalna liczba linii opóźniających pozwalające na płynne przetwarzanie to ~150. Liczba ta zależy również od obecności innych obliczeń towarzyszących uzyskaniu finalnego efektu, takich jak: obliczenia związane z filtracją, uzyskaniem efektów przestrzennych oraz późnego pogłosu (podrozdziały 4.3.2 – 4.3.5). Przy założeniu wykonania wszystkich możliwych obliczeń przed rozpoczęciem przetwarzania, 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 ograniczeniu do tej liczby, aby uzyskać odpowiednio długi 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.

W obu wariantach jedna z wartości czasu opóźnienia zostaje ustalona na 0 (powód takiego działania wiąże się ze sposobem tworzenia linii kopii sygnału i opisany jest w podrozdziale 4.3.2).

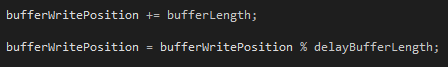
~~W przypadku zmiany przez użytkownika wielkości pogłosu (podrozdział 4.4.) nie jest konieczne ponowne losowanie wartości. Dzięki temu zmiana ta nie skutkuje również niepożądanymi artefaktami. W przypadku~~

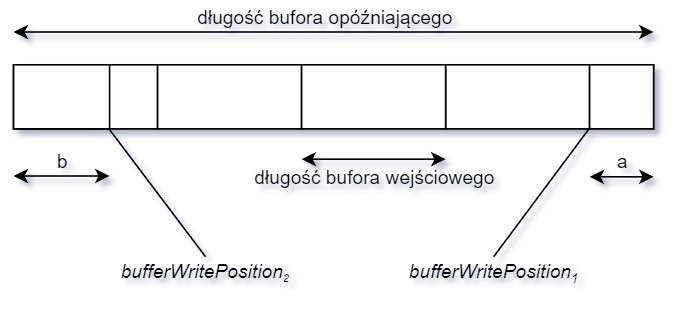
**4.3.2. Sposób uzyskania linii opóźniających ze sprzężeniem.**

A). Utworzenie sygnału sprzężonego z jego opóźnioną kopią.

W celu stworzenia bufora opóźniającego skorzystano z klasy *AudioBuffer,* realizującej wielokanałowy bufor audio złożony z liczb zmiennoprzecinkowych reprezentujących próbki sygnału [31]. 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 częściach algorytmu. Zastosowanie tej klasy pozwala również na wykorzystanie próbek sygnału w tej postaci przez inne klasy wbudowane frameworku.

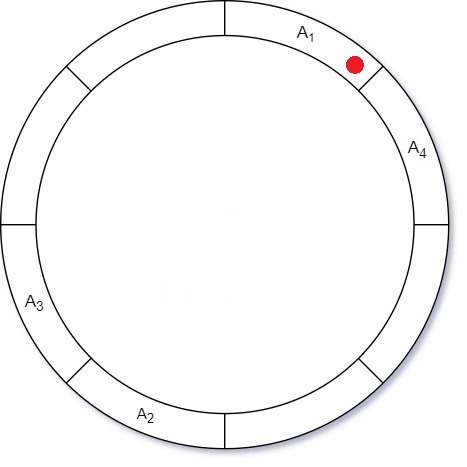
Stworzony bufor opóźniający zapewnia dodatkowy nośnik informacji niezbędny do wykonywania obliczeń wykorzystujących kopie pierwotnego sygnału – kopie te są głównym składnikiem pogłosu. Aby uzyskać sygnał będący sumą sygnału pierwotnego z jego opóźnioną kopią, w pierwszej kolejności stworzono kopię sygnału pierwotnego o długości odpowiadającej długości bufora opóźniającego. Długość bufora opóźniającego została ustalona na ilość próbek odpowiadającą 4 sekundom (jest zależna od częstotliwości próbkowania ustalonej przez użytkownika) – maksymalne opóźnienie linii opóźniającej jest nie większe niż 4 sekundy. Tak więc jest to ilość próbek wystarczająca do tego, aby w każdym momencie w buforze opóźniającym znajdował się cały sygnał pogłosowy. Następnie za pomocą funkcji bibliotecznej A*udioBuffer::copyFrom()* wykonane zostaje wypełnienie bufora opóźniającego następującymi po sobie buforami przygotowanymi pierwotnie do wysłania na wyjście (zakładając brak przetwarzania - bufory sygnału oryginalnego). 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.

  
Rys. 4.2. Sposób zmiany wartości zmiennej *bufferWritePosition*

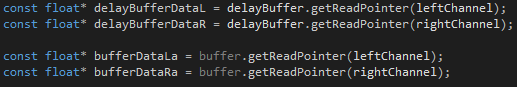
  
Rys. 4.3. Schemat przedstawiający sposób zmiany wartości zmiennej *bufferWritePosition.*

Założone podejście wymaga konieczności utworzenia zmiennej przechowującej informację o pozycji kopiowania i odczytywania danych w buforze opóźniającym. Zmienna ta (*bufferWritePosition*, Rys. 4.2) zainicjalizowana jest na wartością 0, a przy przetwarzaniu każdego kolejnego bufora jej wartość zwiększana o wartość długości bufora wejściowego. Na Rys. 4.3 przedstawiona jest sytuacja, w której wartość zmiennej *bufferWritePosition* przekracza wartość długości bufora opóźniającego. Przed zmianą wartości zmienna przyjmuje wartość *bufferWritePosition1,* po zmianie jej wartość to *bufferWritePosition2.* Na rysunku wartości a i b to liczby, których suma jest równa długości bufora wejściowego. Tak więc zmienna *bufferWritePosition2* jest liczbą powstałą przez odjęcie liczby *a* od długości bufora wejściowego (b to liczba próbek, które nie zmieściły się na końcu bufora opóźniającego.

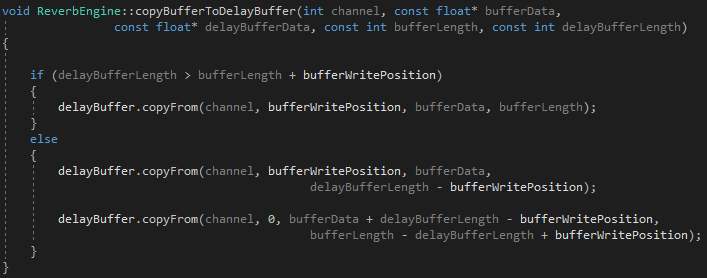
Przy wypełnianiu bufora opóźniającego, zastosowano strukturę danych stosowaną szeroko szczególnie przy potrzebie przetwarzania strumienia danych - bufor kołowy [18]. W celu wizualizacji działania tej struktury danych, sytuacja implementacji z niniejszej pracy została sprowadzona do prostego przypadku (Rys 4.3). 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 A1 - początek bufora wejściowego, A2 – 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 próbek 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 (A3 - początek drugiego bufora wejściowego, A4 - 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. 4.1. Schematyczne przedstawienie działania bufora kołowego.

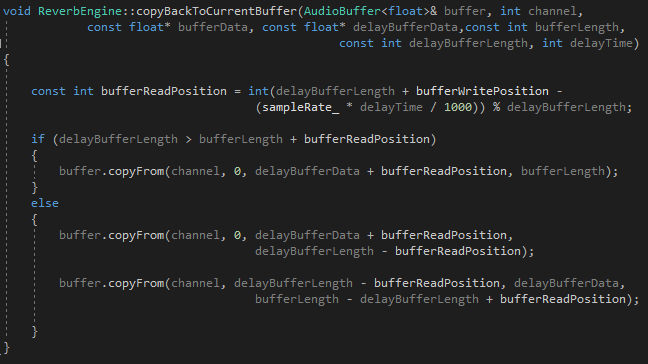
Jak opisano powyżej, przy wypełnianiu bufora opóźniającego, w pewnym momencie może dojść do sytuacji, w której w buforze opóźniającym nie będzie wystarczająco miejsca, aby zmieścił się kolejny bufor wejściowy. W tej sytuacji bufor opóźniający zostaje wypełniony do końca (tyloma próbkami, ile się zmieści), a reszta próbek bufora wejściowego wklejona zostaje na 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 pozycji wskaźnika plus długości bufora wejściowego). Jeśli tak, bufor wejściowy wklejany jest w miejsce aktualnej pozycji wskaźnika *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. 4.2).

  
Rys. 4.2. 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. Posiadanie informacji tych jest konieczne do poprawnego kopiowania i dodawania do siebie buforów, a stosowanie tych operacji jest niezbędne do uzyskania finalnego efektu.

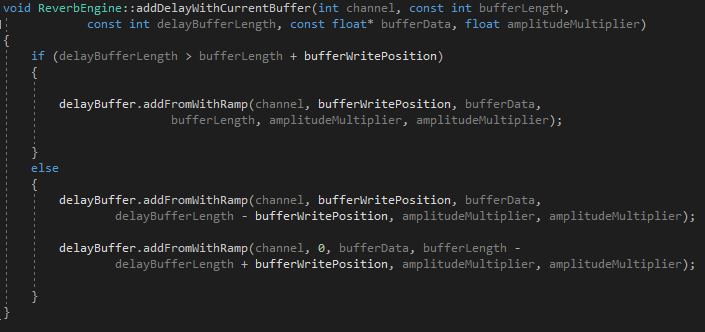
Rys. 4.2. Sposób wypełniania bufora opóźniającego buforami wejściowymi.

Następnie z wypełnionego bufora opóźniającego, kopiowany jest bufor z pewnego miejsca wcześniej (z przeszłości, miejsce to zależy od aktualnego czasu opóźnienia – podrozdział 4.3.2) i podmieniany z aktualnym buforem (Rys. 4.3). 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 iteracji (podpunkt B) – aktualnej wartości w wektorze czasów opóźnienia (jest to liczba próbek odpowiadająca temu czasowi opóźnienia, Rys. 4.3). 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.

Rys. 4.3. 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 zawiera on już swoją opóźnioną wersję. Tym sposobem podczas przetwarzania kolejnego bufora, przy kopiowaniu aktualnego bufora do bufora opóźniającego aktualny bufor zawiera już sumę sygnału i jego kopii. Mając w buforze wejściowym aktualny fragment sygnału i jego opóźnioną kopię, przetwarzanie kolejnego bufora zapewnia istnienie 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. 4.4).

Rys. 4.4. 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:

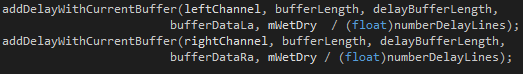
* podmiana bufora wejściowego z fragmentem sygnału z przeszłości (akapit trzeci podpunktu A). Miejsce, z którego odbywa się kopiowanie zależy od aktualnego indeksu wektora czasów opóźnienia i jest różna dla każdej iteracji,
* dodanie bufora aktualnego do bufora opóźniającego (podpunkt A).

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). Po wykonaniu wypunktowanych w niniejszym podpunkcie operacji, w buforze opóźniającym każdy kolejny fragment o długości bufora wejściowego jest sumą wszystkich fragmentów sygnału z przeszłości, które były uprzednio kopiowane do bufora wejściowego. Towarzyszące przetwarzaniu każdego kolejnego bufora wysyłanie sumy tej na wejście zapewnia sprzężenie, dodatkowo zwiększając zawartość symulacji odbić.

//Stworzone linie opóźniające istnieją przez czas trwania (krótki bardzo).

**4.3.3. Tłumienie odbić**

Symulacja tłumienia kolejnych powtórzeń sygnału wykonywana jest przed dodaniem aktualnego bufora do bufora opóźniającego (podrozdział 4.3.2). Zastosowana przy tym dodawaniu funkcja biblioteczna *addFromWithRamp()* [41] 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. 4.4 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 liczbie linii opóźniających (Rys. 4.4). 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]).

Rys. 4.4. Sposób tłumienia amplitudy

**4.3.4. Filtracja linii opóźniających**

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

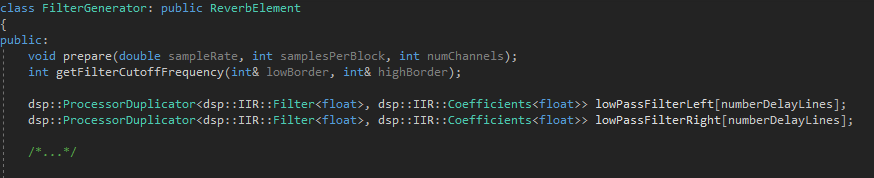
Filtrami zastosowanymi do ograniczenia pasma częstotliwości poszczególnych linii opóźniających były filtry z nieskończoną odpowiedzią impulsową (IIR). Do implementacji filtrów wykorzystane zostały klasy biblioteczne realizujące większość obliczeń związanych z projektowaniem filtrów i samą filtracją.

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

* *dsp::ProcessorDuplicator<MonoProcessorType, StateType>* – klasa realizująca …[36],
* *dsp::IIR::Filter<SampleType>* – klasa realizująca przetwarzanie sygnału filtrem IIR [37],
* *dsp::IIR::Coefficients<NumericType>* – klasa realizująca przechowywanie współczynników filtru IIR w celu późniejszego wykorzystania przez klasę *dsp::IIR::Filter<SampleType>* [38]
* *dsp::ProcessSpec –* struktura niezbędna do prawidłowego przetwarzania sygnału, współpracująca z innymi klasami [39].

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. 4.4.).

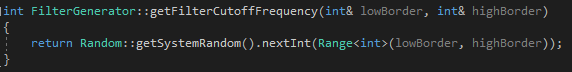
Indeks *numberDelayLines* to stała typu *static const int* i jest polem stworzonej struktury *ReverbElement.* Stała ta przechowuje liczbę linii opóźniających niezmienną w trakcie działania programu (liczba linii aktywnych w danym momencie może się różnić, w zależności od ustawienia pokrętła określającego wielkość pogłosu – podrozdział 4.4). 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.

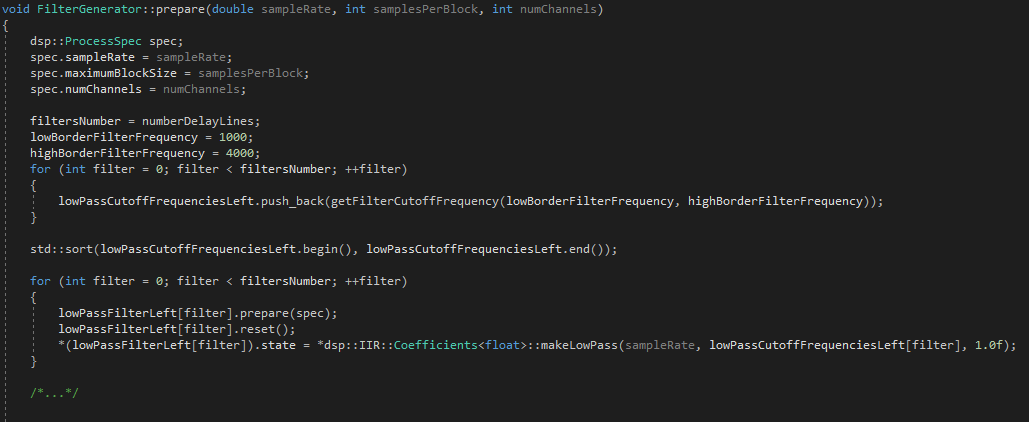
  
Rys. 4.3. Definicja klasy *FilterGenerator* i ich najważniejszych metod, oraz sposób tworzenia tablicy typu *dsp::ProcessorDuplicator.*

  
Rys. 4.4. Sortowanie wektora czasów opóźnienia linii opóźniających i jego odwrócenie.

Rys. 4.5. Sortowanie wektorów częstotliwości odcięcia filtrów dla lewego i prawego kanału.

Chcąc filtrować dolnoprzepustowo tym mocniej (z niższą częstotliwością odcięcia) im fragment kopii sygnału z dalszego momentu wstecz w czasie, wykonano sortowanie wartości czasów opóźnienia (Rys. 4.4.) (w wersji pogłosu z losowymi czasami opóźnienia sortowanie to jest konieczne), przy jednoczesnym sortowaniu częstotliwości odcięcia filtrów (Rys. 4.5.). Aby najdłuższy czas opóźnienia odpowiadał najniższej częstotliwości odcięcia, jeden z wektorów musi zostać odwrócony (Rys. 4.4.).

Rys. 4.6. Funkcja realizująca generowanie częstotliwości odcięcie filtru dolnoprzepustowego.

Rys. 4.7. Sposób przygotowania obiektu struktury *dsp::ProcessSpec,* wygenerowanie wektora częstotliwości odcięcia filtrów oraz przygotowanie filtrów do późniejszego przetwarzania (kanał lewy).

Rys 4.8. Przetwarzanie prawego kanału aktualnego bufora audio przygotowanymi filtrami.

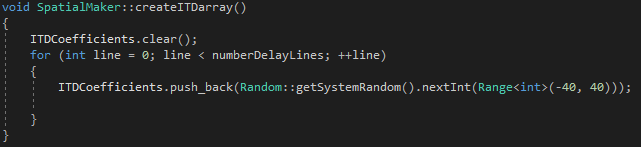
Przy dodawaniu do siebie kopii sygnału z różnych miejsc w czasie (podpunkt B, podrozdział 4.3.2) towarzyszącemu generowaniu wyjściowego bufora, wykonywana jest jego filtracja – po wykonaniu kopii fragmentu sygnału o długości bufora wejściowego z pewnego miejsca w czasie, wykonywana jest jego filtracja. 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 (Rys. 4.4, 4.5).

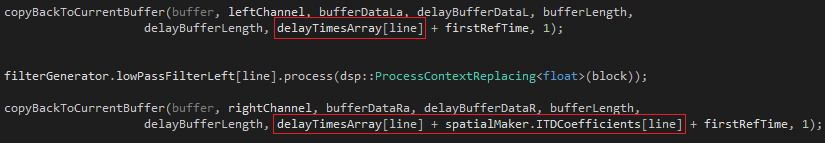
Ostateczny krok, a więc sama filtracja bufora wymaga (w tym podejściu) zastosowania klasy bibliotecznej *dsp::AudioBlock* [40] i przypisania do jej obiektu pożądanego do filtracji bufora. To, jaką zawartość posiada bufor w momencie przypisywania klasie *dsp::AudioBlock* (Rys. 4.8)*,* zależy od miejsca 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. 4.7).

**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. 4.9). Generowanie wektora odbywa się takiej samej zasadzie, co wektora częstotliwości odcięcia filtrów (podrozdział 4.3.4, Rys. 4.6), z tą różnicą, iż zakres losowania wartości jest przedziałem [-40,40] ms (podpunkt C, podrozdział 2.2).

Rys. 4.9. Sposób tworzenia wektora różnicy czasów opóźnienia.

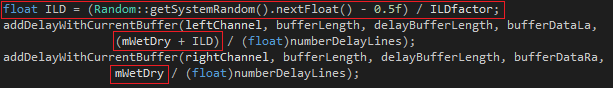
Rys. 4.10. Wykorzystanie wektora różnicy czasów opóźnienia w celu symulacji zjawiska międzyusznej różnicy czasu.

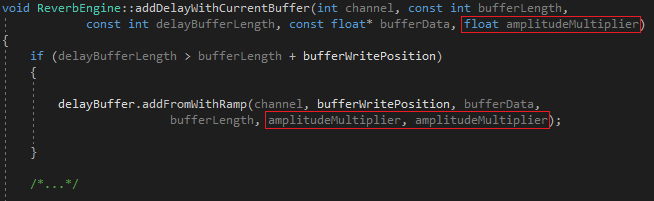
Przekazywana do funkcji *copyBackToCurrentBuffer* (ciało funkcji na Rys. 4.2) aktualna wartość wektora *delayTimesArray* (Rys. 4.10) w danej iteracji jest liczbą milisekund odpowiadającą miejscu, z którego kopiowany jest bufor (podrozdział 4.3.2). Dodając do tej wartości aktualną wartość wektora różnicy 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 (danej iteracji) kopii sygnału lewego, w przypadku liczby dodatniej – prawego.

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 4.3.2).

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, miejsce przesyłania argumentu odpowiedzialnego za tłumienie do funkcji oznaczono na Rys. 4.4). 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 to [-0,5;0,5]. Wylosowana wartość dodawana jest do aktualnej wartości amplitudy jednego z kanałów i dzielona przez liczbę linii opóźniających (podrozdział 4.3.3). Wartość ta dzielona jest przez współczynnik, który może być modyfikowany przez użytkownika za pomocą pokrętła (podrozdział 4.4). Dzięki temu użytkownik może określać, jak bardzo amplituda pomiędzy kanałami w liniach opóźniających ma się różnić (Rys. 4.3).

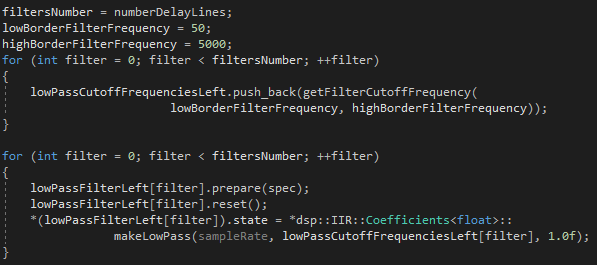
Rys. 4.3. Sposób ograniczania amplitudy w celu symulacji zjawiska międzyusznej różnicy poziomu.

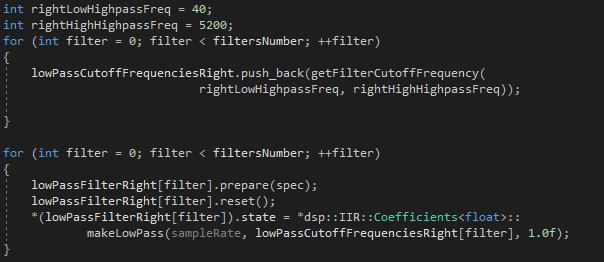
Rys. 4.4.

C). Różna filtracja obu kanałów

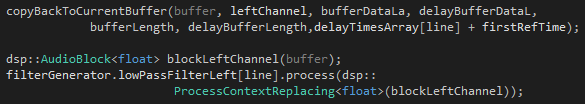
Symulacja przestrzenności z wykorzystaniem filtrów zakłada filtrację lewego i prawego kanału w inny sposób, dzięki czemu słuchacz odczuwać będzie większe wrażenie przestrzenności.

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 4.3.3 są one losowane dwukrotnie - dzięki temu dla lewego i prawego kanału są one różne. 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. 4.5, 4.6).

Rys. 4.5. Sposób przygotowania filtrów dolnoprzepustowych dla lewego kanału.

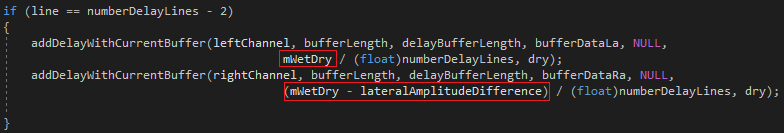
Rys. 4.6. Sposób przygotowania filtrów dolnoprzepustowych dla prawego kanału.

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 w jednym z kanałów, konieczne jest dokonanie tego przypisania w momencie, gdy dany bufor zawiera tylko jeden z kanałów. Tak więc w tym przypadku funkcja *process* klasy *dsp::IIR::Filter* musi zostać wywołana także pomiędzy skopiowaniem lewego kanału fragmentu bufora opóźniającego do wejściowego (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. 4.5). Tym sposobem filtrowany jest tylko bufor lewego kanału. 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.

Rys. 4.5. 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 jednak tylko dla jednego odbicia – dla kopii sygnału o czasie odbicia, a więc najkrótszym czasie opóźnienia.

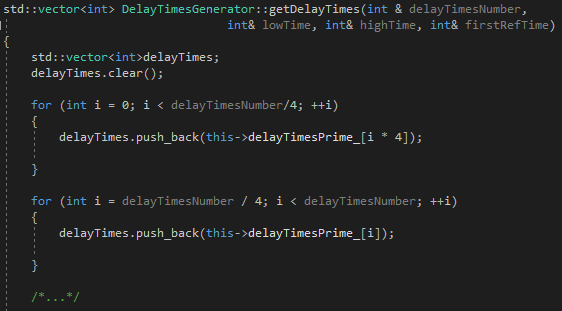
Rys. 4.11. Realizacja różnicy poziomu między kanałami pierwszych odbić bocznych.

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 4.3.2) wartość pierwszego odbicia znajduje się w przedostatnim elemencie tablicy (wartość najmniejsza w wektorze nie licząc dźwięku bezpośredniego). Na Rys. 4.11 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* dotyczy kanału lewego, drugie – prawego. Fragmenty oznaczone czerwonymi obramowaniami przekazują do funkcji informację, jak bardzo tłumione będzie pierwsze odbicie (podrozdział 4.3.2). 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).

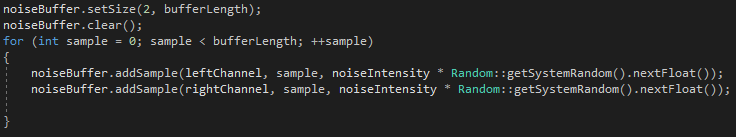
  
Rys. 4.12. Sposób zagęszczenia późnego ogona pogłosowego.

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 będących pierwszych występujących po sobie po kolei od wartości maksymalnej pierwszej ćwiartki wektora (Rys. 4.12). Tym sposobem w drugiej części wektora znajduje się o wiele więcej liczb, w przedziale liczbowym o podobnej długości.

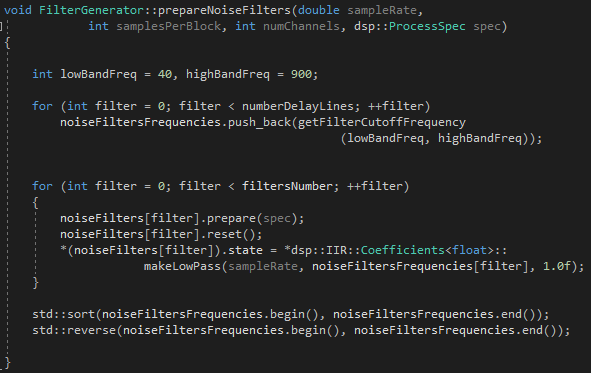
|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Liczba wartości  w pierwszej ćwiartce | Liczba wartości  w pozostałej części |
| Liczba linii opóźniających |  |  |  |

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]. Po wypełnieniu bufora szumem, zostaje wykonana jego filtracja dolnoprzepustowa, a następnie bufor szumowy mnożony jest z buforem sygnału w sposób opisany poniżej.

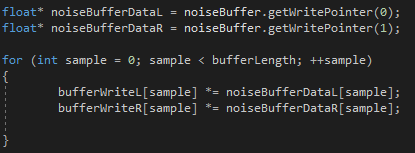
Rys. 4.13. Sposób wypełnienia lewego i prawego kanału bufora szumem białym.

W pierwszej kolejności konieczne jest wypełnienie lewego i prawego kanału bufora pseudolosowymi liczbami zmiennoprzecinkowymi. Sposób wypełnienia bufora przedstawiony jest na Rys. 4.13.

  
Rys. 4.14. 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. 4.14). 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. 4.15).

Rys. 4.15. Filtracja bufora wypełnionego szumem.

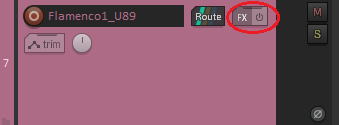
  
Rys. 4.16. Mnożenie bufora sygnału z szumem filtrowanym szumem białym.

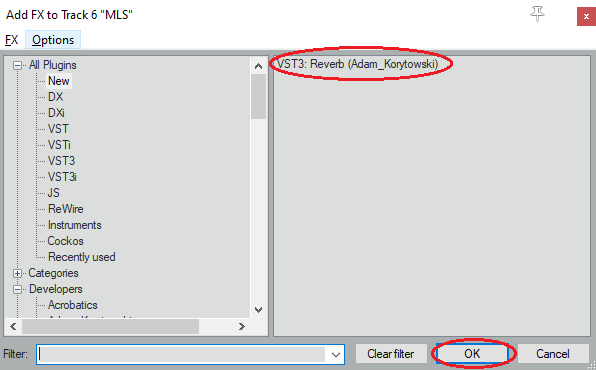
Bufor szumowy i aktualny bufor sygnału mają taką samą długość - każda kolejna próbka wyjściowego sygnału to próbka bufora aktualnego sygnału pomnożona z odpowiadającą jej próbką bufora szumowego (Rys. 4.16).

**4.4. Część funkcjonalna aplikacji**

**4.4.1. Przygotowanie wtyczki do działania**

Ze względu na format stworzonej aplikacji (wtyczka VST3), do korzystania z aplikacji niezbędne jest oprogramowanie typu DAW (*ang. Digital Audio Workstation*). Aby wtyczka była widzialna 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. 4.20-4.21).

  
Rys. 4.20. Przycisk umożliwiający przypisanie wtyczki do ścieżki w programie Reaper 6.

Rys. 4.21. Sposób załadowania wtyczki VST3 do ścieżki w przypadku programu Reaper 6.

Po poprawnym załadowaniu wtyczki do ścieżki, sygnał przypisany do tej ścieżki będzie sygnałem przetworzonym przez operacje, które wykonuje wtyczka podczas jej działania. Dokładny sposób przetworzenia sygnału zależy od ustawień elementów interfejsu danych użytkownikowi do modyfikacji.

**4.4.2. Interfejs użytkownika aplikacji**

**5. Podsumowanie i wnioski**

Celem niniejszej pracy było stworzenie cyfrowej symulacji zjawiska pogłosu w oparciu o analizę istniejących rozwiązań implementacyjnych oraz świadomość istnienia percepcyjnych zjawisk akustycznych. Aplikacja docelowo została zrealizowana w formacie VST. Przedstawione zostały najbardziej popularne podejścia do realizacji elementów sztucznego pogłosu, dokonano także oceny poszczególnych elementów pogłosu korzystając z zaproponowanych parametrów. Na tej podstawie dokonany został wybór konkretnych parametrów/sposobów implementacyjnych w celu realizacji finalnej aplikacji. Przedstawiono także zaprojektowaną procedurę zrealizowania sztucznego pogłosu z zastosowaniem wybranych rozwiązań i ich modyfikacjami.

*//podsumowanie będzie poszerzone o komentarz wyników, opis procedury itp.*

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.

Należy pamiętać również o tym, iż duża ilość zmiennych przy badaniach (m. In. minimalna i maksymalna wartość czasu opóźnienia linii opóźniającej, zakresy częstotliwości odcięcia filtrów, różnica w amplitudzie odbić bocznych) powoduje, iż w wszystkie możliwe przypadki są praktycznie niemożliwe to zbadania (ze względu na ich liczbę), w zależności od kombinacji wyniki mogą się różnić. Nawet w przypadku wykonania badań przy parametrach zalecanych przez literaturę, w przypadku manipulacji którejś ze zmiennych przez użytkownika podczas działania programu może wpłynąć na zdefiniowane parametry.

W przypadku znalezienia parametrów zbieżnych co do oceny ze wskazaniami subiektywnymi,

.. wyniki mogą różnić się w zależności od przyjętych zakresów parametrów

… , a co za tym idzie będzie istniała możliwość oceny jakości pogłosu w sposób obiektywny z dużą szansą, iż wskazania parametrów pokrywają się z wrażeniami słuchowymi, bez konieczności wykorzystywania dużej grupy ludzi do badań.

Historia realizacji tego typu projektów pokazuje, iż czynnik oceny subiektywnej jest nieodłącznym jej elementem – w przeszłości ze względu fakt niewielkich możliwości obliczeniowych w czasach publikacji kluczowych dla tematyki prac. Ocena subiektywna wynikała także z powtarzalności oceny przy testach odsłuchowych – przez powtarzalność opinii osób badanych na temat sygnałów uznawano metodę badawczą za wiarygodną.

Faktyczna sytuacja związana z propagacją fali w pomieszczeniu jest znacznie bardziej skomplikowana niż symulacja będąca wynikiem zaimplementowanego rozwiązania. Należy jednak pamiętać, iż przedstawione rozwiązanie miało na celu realizację pogłosu w postaci formie efektu, w odróżnieniu do uzyskania jak najbardziej realistycznego pogłosu. Istnieje także konieczność proponowania innych parametrów pozwalających na obiektywne porównywanie sygnałów pod kątem zjawisk towarzyszących propagacji dźwięku w pomieszczeniu. Metodą na ocenę zamodelowanych zjawisk akustycznych mogą być także stosowane dotąd badania na grupie słuchaczy, którzy będą oceniać sygnał pod względem zaproponowanych parametrów i kryteriów ocen.

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.

**Bibliografia**

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

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

[3] Schroeder – Colorless Sounding Artificial Reverberation

[4] Schroeder – Natural Sounding Artificial Reverberation

[5] J. Stautner and M. Puckette – Designing Multi–Channel Reverberators

[6] Beranek (?)

[7] Music 175: Time and Space Tamara Smyth, Department of Music, University of California, San Diego (UCSD) April 16, 2019:  
http://musicweb.ucsd.edu/~trsmyth/space175/space175.pdf

[8] https://www.dsprelated.com/freebooks/pasp/Artificial\_Reverberation.html

[9] Moorer – About this reverberation business

[10] V.Välimäki, B. Holm–Rasmussen, B. Alary and H. M. Lehtonen – Late Reverberation Synthesis Using Filtered Velvet Noise

[11] P. Kleczkowski – Percepcja Dźwięku

[12] A. H. Marshall – A note on the importance of room cross–section in concert halls

[16] Kod źródłowy oraz opis frameworku JUCE:  
https://github.com/WeAreROLI/JUCE

*… git*

[17] Charakterystyka Teorii Dupleksowej:  
[http://diamonddissertation.blogspot.com/2010/05/rayleigh–duplex–theory.html](http://diamonddissertation.blogspot.com/2010/05/rayleigh-duplex-theory.html)

[18] Opis potrzeby stosowania bufora kołowego oraz problemu producenta i konsumenta:  
[https://embedjournal.com/implementing–circular–buffer–embedded–c/](https://embedjournal.com/implementing-circular-buffer-embedded-c/)

[19] [https://en.wikipedia.org/wiki/Spectral\_density#:~:text=Energy%20spectral%20density%20describes%20how%20the%20energy%20of,transients%E2%80%94that%20is%2C%20pulse-like%20signals%E2%80%94having%20a%20finite%20total%20energy.](https://en.wikipedia.org/wiki/Spectral_density" \l ":~:text=Energy spectral density describes how the energy of,transients—that is%2C pulse-like signals—having a finite total energy.)

[20] [https://pl.wikipedia.org/wiki/Widmowa\_g%C4%99sto%C5%9B%C4%87\_mocy](https://pl.wikipedia.org/wiki/Widmowa_gęstość_mocy)

[21] <https://en.wikipedia.org/wiki/Spectral_flatness>

[22] [https://en.wikipedia.org/wiki/Spectral\_density#Cross-spectral\_density](https://en.wikipedia.org/wiki/Spectral_density" \l "Cross-spectral_density)

[24] <https://en.wikipedia.org/wiki/Coherence_(signal_processing)>

[25] Dokumentacja funkcji *coherence* z biblioteki *scipy w języku Python:*  
<https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.coherence.html>

[26] Dokumentacja funkcji spectral\_flatness z biblioteki *scipy* w języku Python*:* <https://librosa.org/librosa/generated/librosa.feature.spectral_flatness.html>

[27] Dokumentacja funkcji *welch* z biblioteki *scipy* w języku Python*:* [https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.welch.html#scipy.signal.welch](https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.welch.html" \l "scipy.signal.welch)

[28] Davis, Gary (1987). *The sound reinforcement handbook* (2nd ed.). Milwaukee, WI: Hal Leonard. p. 259. [ISBN](https://en.wikipedia.org/wiki/ISBN_(identifier)) [9780881889000](https://en.wikipedia.org/wiki/Special:BookSources/9780881889000). Retrieved February 12, 2016.

[29] Charakterystyka splotu dyskretnego:  
<https://pages.jh.edu/~signals/discreteconv2/index.html>

[30] Charakterystyka filtru wszechprzepustowego:  
<https://ccrma.stanford.edu/~jos/pasp/Schroeder_Allpass_Sections.html>

[31] Dokumentacja klasy AudioBuffer frameworku JUCE: <https://docs.juce.com/master/classAudioBuffer.html>

[32] Dokumentacja klasy *Random* frameworku JUCE:  
<https://docs.juce.com/master/classRandom.html>

[33] Dokumentacja klasy *Range* frameworku JUCE:  
<https://docs.juce.com/master/classRange.html>

~~[34] Opis problem producenta i konsumenta:~~   
[~~http://pages.cs.wisc.edu/~remzi/OSTEP/threads–cv.pdf~~](http://pages.cs.wisc.edu/~remzi/OSTEP/threads-cv.pdf)

[35] Opis problemu realizacji pogłosu i wskazówki projektowe:  
<https://www.dsprelated.com/freebooks/pasp/Artificial_Reverberation.html>

[36] Dokumentacja klasy *dsp::ProcessorDuplicator* frameworku JUCE:  
[://docs.juce.com/master/structdsp\_1\_1ProcessorDuplicator.html](https://docs.juce.com/master/structdsp_1_1ProcessorDuplicator.html)

[37] Dokumentacja klasy *dsp::Filter::IIR* frameworku JUCE: [http://charette.no–ip.com:81/programming/doxygen/juce/classjuce\_1\_1dsp\_1\_1IIR\_1\_1Filter.html#details](http://charette.no-ip.com:81/programming/doxygen/juce/classjuce_1_1dsp_1_1IIR_1_1Filter.html#details)

[38] Dokumentacja klasy *dsp::IIR::Coefficients* frameworku JUCE:  
<https://docs.juce.com/master/structdsp_1_1IIR_1_1Coefficients.html>

[39] Dokumentacja struktury *dsp::ProcessSpec* frameworku JUCE:  
<https://docs.juce.com/master/structdsp_1_1ProcessSpec.html>

[40] Dokumentacja klasy *dsp::AudioBlock* frameworku JUCE:  
<https://docs.juce.com/master/classdsp_1_1AudioBlock.html>

[41] Dokumentacja funkcji *addFromWithRamp()* klasy *AudioBuffer* frameworku JUCE:  
<https://docs.juce.com/blocks/classAudioBuffer.html#a1853da41f2a3d87fa6291a2bf2adca05>