AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej

Praca dyplomowa inżynierska

Programowa implementacja efektu pogłosu

Software implementation of the reverb effect

Autor: *Radosław Filip Reguła*
Kierunek studiów: Inżynieria Biomedyczna
Opiekun pracy: *dr Marek Pluta*

Kraków, 2019

OŚWIADCZENIA STUDENTA

Kraków, dnia .....r.

.....
Imiona i nazwisko studenta

.....
Kierunek, poziom, forma studiów i profil

.....
Nazwa Wydziału

.....
Imiona i nazwisko opiekuna pracy dyplomowej

ja niżej podpisany(-a) oświadczam, że:

jako twórca / współtwórca* pracy dyplomowej inżynierskiej / licencjackiej / magisterskiej* pt.
.....
.....

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(-a) 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(-am) osobiście i samodzielnie i nie korzystałem(-am) 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. [ ]** 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 studenta w 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.

Spis treści

1. Cel i zakres pracy

Celem niniejszej pracy było stworzenie prototypów, a następnie wykorzystanie wybranego algorytmu generacji sztucznego pogłosu do implementacji w języku C++ programu dodającego do pliku dźwiękowego efekt o edytowalnych za pomocą graficznego interfejsu użytkownika parametrach brzmieniowych. Odpowiedni algorytm miał być wybrany w toku realizacji pracy na podstawie porównania najpopularniejszych istniejących metod. Rozdział 2. stanowi wprowadzenie do tematu pracy, opisując metodykę działań i charakterystykę podjętego zagadnienia. Rozdział 3. zawiera streszczenie fizycznych fundamentów zjawiska pogłosu, których zrozumienie pełni ważną rolę w projektowaniu metod symulacyjnych. Kolejne dwa rozdziały pracy poświęcone są rozwiązaniom generacji sztucznego pogłosu – studium wybranych zagadnień z tej dziedziny zamieszczono w rozdziale 4., zaś w rozdziale 5. opisano proces testowej implementacji trzech wybranych metod cyfrowych, wraz z uzasadnieniem wyboru metody użytej w finalnej aplikacji. W rozdziale 6. zamieszczono przedstawienie końcowego wyniku pracy, czyli programu z interfejsem graficznym. Na końcu pracy zamieszczono spis wykorzystanych źródeł naukowych.

2. Wstęp

Zagadnienie implementacji cyfrowych efektów dźwiękowych jest dziedziną, która teoretyczne zdobycze takich nauk jak technika cyfrowa, przetwarzanie sygnałów czy programowanie przekłada na grunt działalności zarówno inżynierskiej, jak i artystycznej. Dynamiczny rozwój tej dziedziny zapoczątkowany został w latach 60. XX wieku, wraz ze wzrastającym wykorzystaniem metod obliczeniowych w realizacji zadań, które dotychczas bazowały na rozwiązaniach analogowych. Dzięki badaczom z takich placówek, jak laboratoria Bella czy IRCAM w Paryżu, cyfrowe przetwarzanie dźwięku dość szybko osiągnęło wystarczająco wysoki poziom technologiczny, aby być wykorzystywane w wielu różnych i odległych od siebie koncepcyjnie dziedzinach nauki. W dzisiejszych czasach, rezultaty tamtych eksperymentów pozwoliły na znaczące rozwinięcie dziedzin takich jak produkcja muzyczna (efekty dźwiękowe w formie oprogramowania są powszechnym narzędziem pracy inżynierów studyjnych), przetwarzanie mowy (generatory mowy) czy produkcja filmowa (symulacje zjawisk akustycznych w ścieżce dźwiękowej), ale także akustyka budowlana i projektowanie architektoniczne (cyfrowe modelowanie warunków akustycznych pomieszczenia w oparciu o szczegóły geometryczne i materiałowe). Lata pracy nad udoskonaleniem wydajności obliczeniowej i otrzymywanych rezultatów percepcyjnych wpłynęły na bardzo wysoki poziom zaawansowania technologicznego dziedziny, która dziś oferuje rozwiązania w formie urządzeń zewnętrznych lub oprogramowania komputerowego, zapewniających symulacje rzeczywistych zjawisk akustycznych lub generację nieosiągalnych w rzeczywistości efektów, działających w czasie rzeczywistym lub nie.

Sam pogłos stanowił pierwotnie bardzo interesujący przedmiot badań, ze względu na efekt realistyczności, który nadaje on nagraniom. Ścieżki dźwiękowe zupełnie pozbawione pogłosu brzmią nienaturalnie, a uzyskanie tego efektu w momencie nagrania było często niezwykle kłopotliwe, ponieważ wiązało się z koniecznością odbycia sesji nagraniowej w odpowiednio dobranej akustycznie przestrzeni. Temat cyfrowej realizacji symulacji zjawiska rewerberacji został więc podjęty szybko i dość szybko przyniósł bardzo zadowalające rezultaty brzmieniowe. W międzyczasie opracowano szereg metod, na opracowaniu których skupia się niniejsza praca.

Przyjęta metodyka działań związana była z pierwotną ideą wykorzystania opracowanego rozwiązania obliczeniowego, które stanowić będzie najbardziej optymalne podejście do problemu z punktu widzenia osoby bazującej na wiedzy z zakresu programowania komputerów i przetwarzania sygnałów cyfrowych, jednak niedoświadczonej w tematyce tworzenia efektów dźwiękowych. Z tego powodu, ważną część pracy stanowił etap przeglądu literaturowego, który pozwolił na ustalenie obecnego stanu wiedzy w rozpatrywanej dziedzinie. Działanie to realizowane było od ogólnych przeglądów stanu techniki, aż do szczegółowych rozpraw na temat indywidualnych podejść do tematu autorstwa badaczy z ośrodków zajmujących się dźwiękiem. Na ich podstawie wyłoniono trzy bazowe algorytmy, których szerokie wykorzystanie i zaawansowane metody modyfikacji świadczą o wyjątkowym znaczeniu dla rozwoju dziedziny. Te algorytmy zaimplementowano w postaci skryptów w języku Python – ten język programowania wybrano ze względu na prostą składnię, dającą możliwość szybkiego pisania efektywnego kodu oraz istnienie szeregu przeznaczonych dla niego bibliotek znacząco ułatwiających obliczenia numeryczne, pracę z sygnałami cyfrowymi i wizualizację danych. Implementację metod poprowadzono w zgodności z wytycznymi przedstawionymi w przyjętych za najistotniejsze pracach naukowych, do których odniesienia znajdowały się w większości opracowywanych materiałów źródłowych. Po wykonaniu prototypów, analizowane algorytmy poddano porównaniu w celu wyłonienia metody najlepszej do użycia w ostatecznym programie. Metoda ta stała się osią programu finalnego, który oprócz mechanizmu symulacji efektu posiadał interfejs graficzny oraz opcje ułatwiające obsługę i dobór odpowiedniego rozwiązania brzmieniowego. Ostateczny program napisany został w języku C++ z wykorzystaniem szkieletu aplikacyjnego przeznaczonego do aplikacji dźwiękowych, JUCE. Wybór tego języka w ostatecznej implementacji związany był głównie z jego dużo większą wydajnością, co w przypadku pracy na sygnałach o dużej częstotliwości próbkowania ma niebagatelne znaczenie. C++ jest językiem programowania najpowszechniej wykorzystywanym w tworzeniu aplikacji multimedialnych i wtyczek audio. Wykorzystanie możliwości języka C++ oraz platformy JUCE pozwoliło na znaczące ułatwienie procesu przygotowania interfejsu graficznego oraz zaprojektowania struktury działania programu.

Niniejsza praca zachowuje przytoczony powyżej logiczny układ organizacji działań. Opisy poszczególnych etapów realizowanego zadania wzbogacone są o odnoszące się do treści ilustracje, a także wykresy i schematy opracowanych rozwiązań. Przytoczono krótkie fragmenty kodu, służące do przybliżenia opisywanego akurat zagadnienia.

3. Teoretyczne podstawy zjawiska pogłosu

Pogłos jest zjawiskiem fizycznym, które musi być traktowane jako nieodłączny element każdej zamkniętej przestrzeni akustycznej. Jest ono bowiem nierozerwalnie związane ze sposobem rozchodzenia się fal dźwiękowych i ich oddziaływania zarówno ze stałymi przeszkodami, jak i samym ośrodkiem. Zjawisko pogłosu jest możliwe do zaobserwowania w czasie następującym bezpośrednio po ustaniu emisji dźwięku w danej przestrzeni. W sytuacji tej dochodzi do załamania ustalonego wcześniej stanu równowagi, w którym energia akustyczna emitowana przez źródło o stałej mocy uzupełniała starty energii wynikające z pochłaniania. Wytworzone przez źródło dźwięku fale akustyczne rozchodzą się we wszystkich kierunkach, docierając do odbiorcy zarówno bezpośrednio, trasą pozbawioną fizycznych przeszkód, jak i jako dyskretne echa, opóźnione i częściowo wytłumione w wyniku odbić fali od barier. Dźwięk rejestrowany przez odbiorcę jest zatem wynikiem sumowania się tych składowych, co objawia się poprzez stopniowy i skokowy wzrost ciśnienia akustycznego w danym punkcie przestrzeni [1]. Po zakończeniu działania emitera dźwięku fala bezpośrednia zanika natychmiastowo, jednak fale odbite docierają do odbiorcy jeszcze przez pewien skończony okres czasu, stopniowo wytracając swoją energię w wyniku absorpcji przy każdym odbiciu oraz tłumiącego wpływu ośrodka rozchodzenia się fali. Sensorycznie, taki stan rzeczy doprowadza do pozornego „zawieszenia" dźwięku. Czas trwania tego zawieszenia określa się jako czas pogłosu. Jest to parametr mierzalny i wykorzystywany szeroko do opisu charakterystyki akustycznej pomieszczenia. Czas pogłosu definiuje się jako czas, w którym natężenie dźwięku spada po wyłączeniu źródła do określonego ułamka swojej wartości początkowej [2]. Do pomiarów przyjęta została stała 60 dB. Jest to ekwiwalent zmiany natężenia akustycznego 10^6 razy lub zmiany poziomu ciśnienia akustycznego 10^3 razy [1]. Czas rewerberacji jest funkcją wielu czynników, wśród których wymienić można wielkość pomieszczenia, jego kształt, obecność elementów tłumiących w jego wnętrzu (wliczając w to np. siedziska na hali koncertowej czy samą publikę) oraz współczynnik tłumienia barier, takich jak ściany, podłoga czy sufit [3]. W pomieszczeniu o dużym rozmiarze, w którym fala pokonuje dłuższą średnią drogę pomiędzy odbiciami i w którym ściany wykonane są z materiału słabo chłonącego (np. kamienne, pokryte kafelkami), a także w którym występuje mało przeszkód dla fal dźwiękowych (pusta przestrzeń), czas pogłosu będzie najdłuższy. Pomieszczenia małe, wypełnione i o chłonnych ścianach będą charakteryzowały się krótkim pogłosem. Ten stan rzeczy znalazł odzwierciedlenie w procesorach sztucznego pogłosu, w których ustawienia długości zaniku dźwięku często określane są poprzez nazwy pomieszczeń („*small room*", „*cathedral*", „*chamber*", pol. „mały pokój", „katedra", „komnata").

Zapewnienie odpowiedniego do zastosowania pomieszczenia czasu pogłosu jest ważnym wyzwaniem architektonicznym. Pionierem skonkretyzowanych badań w tym kierunku był W. C. Sabine. Jako pierwszy zaproponował on wzór, który pozwala na określenie czasu po-

głosu pomieszczenia w zależności od jego wymiarów geometrycznych i współczynników określających chłonność akustyczną [4].

$$t_{60} = \frac{0{,}161V}{\alpha_{śr}S} \tag{1}$$

Gdzie:
t_{60} – czas pogłosu w pomieszczeniu [s],
V – objętość pomieszczenia [m^3],
S – powierzchnia ograniczająca (ściany, sufit, podłoga) [m^2],
$\alpha_{śr}$ – średni współczynnik pochłaniania dźwięku.

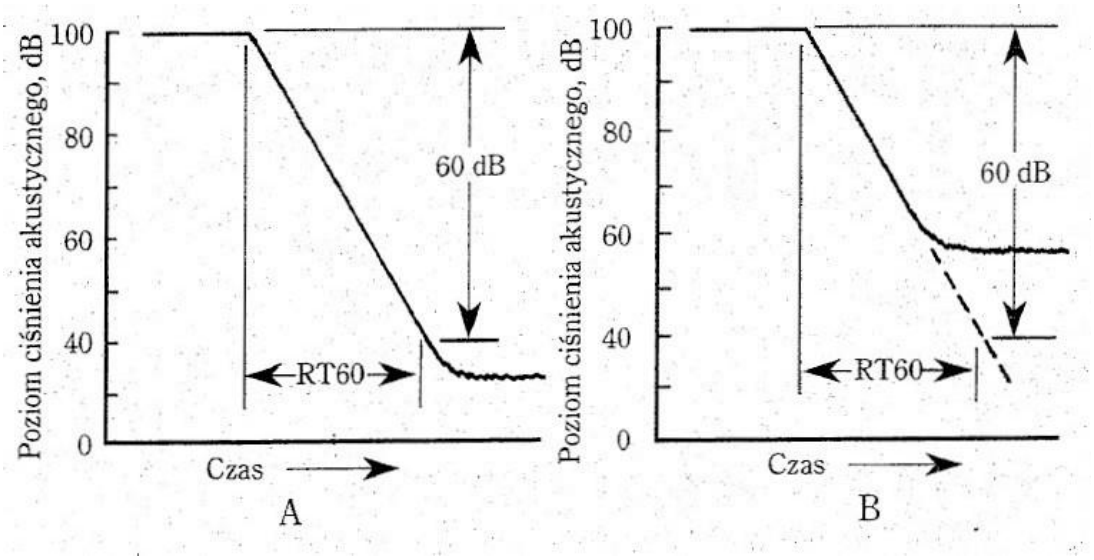
W następnych latach odkryto, że wzór Sabine'a znajduje zastosowanie dla wnętrz słabo pochłaniających. Wersję uogólnioną dla pomieszczeń o zróżnicowanym pochłanianiu opisuje wzór Norrisa – Eyringa [2]:

$$t_{60} = \frac{0{,}161V}{-Sln(1-\alpha_{śr})} \tag{2}$$

Czas pogłosu jest także zależny od tłumienia fal dźwiękowych przez powietrze, które zmienia się wraz ze zmianą jego wilgotności. Zwiększenie wilgotności powietrza doprowadza do wydłużenia czasu rewerberacji. Efekt ten, ze względu na swoją subtelność, nabiera znaczenia dopiero dla dużych hal o długim czasie pogłosu [5]. Pomimo to, jest brany pod uwagę przez projektantów narzędzi do cyfrowych symulacji pogłosowych [6].

Sposoby pomiaru czasu pogłosu można podzielić na metody pośrednie i bezpośrednie [7]. Metody bezpośrednie opierają się na odczycie odpowiedniej wartości z krzywej zaniku dźwięku, uzyskanej podczas przeprowadzonego w ustandaryzowany sposób eksperymentu. Metody pośrednie wykorzystują zależności pomiędzy innymi, łatwiej mierzalnymi wielkościami a czasem pogłosu. Pomiar metodą bezpośrednią polega na wzbudzeniu pogłosu badanej przestrzeni poprzez zastosowanie źródła dźwięku o dużej mocy oraz zarejestrowaniu odpowiedzi pomieszczenia przy pomocy bezkierunkowych mikrofonów połączonych z miernikami poziomu dźwięku. Do pomiarów stosuje się źródła impulsowe, takie jak strzały z pistoletu, wyładowania elektryczne czy przekłuwane balony, a także źródła stałe, takie jak generatory fali sinusoidalnej lub generatory szumu losowego [1]. Ważnym czynnikiem jest uzyskanie odpowiednio dużej energii w całym zakresie widma (w przypadku źródeł impulsowych) i możliwość klarownego, momentalnego przerwania generacji dźwięku (w przypadku źródeł stałych). Częstym utrudnieniem w zastosowaniu metody bezpośredniej jest niemożność uzyskania odpowiednio dużej rozpiętości dynamicznej, wynikającej np. ze zbyt dużego szumu tła. W wypadku takiego pomiaru, wykres nie będzie w stanie zapewnić pożądanego spadku wynoszącego 60 dB. W takich sytuacjach stosuje się ekstrapolację przebiegu tłumienia [1].

Metodę uzyskiwania wartości czasu pogłosu (RT60) według tej metodologii przedstawia Rys. 1.



Rys. 1 (A) Sposób pomiaru czasu pogłosu przy uzyskaniu pożądanego spadku 60 dB. (B) Sposób pomiaru czasu pogłosu w przypadku wystąpienia konieczności ekstrapolacji [1].

Uzyskane podczas pomiaru metodą bezpośrednią nagrania stanowią tzw. odpowiedzi impulsowe pomieszczeń. Takie próbki dźwiękowe, z wyjątkiem swojej roli pomiarowej, stanowią gotowy składnik budulcowy procesorów pogłosowych opartych na metodzie splotowej [8].

Pogłos jest zjawiskiem wpływającym w znaczący sposób na odbiór wrażeń akustycznych w zamkniętej przestrzeni. Jego nadmiar (długi czas) doprowadzić może do spadku zrozumiałości mowy. Wynika to z faktu, iż potęgujące się dyskretne echa początkowych sylab nakładają się na dalszą część słowa, doprowadzając do wyraźnie mniejszej rozróżnialności głosek cichszych. Zbyt duży pogłos jest także niepożądany w salach koncertowych i studiach muzycznych, gdzie stanowczo utrudnia poprawny odbiór muzyki i maskuje dynamikę nagrań bądź partii instrumentalnych. Z kolei zbyt krótki pogłos lub zupełny jego brak w pomieszczeniu doprowadzają do sytuacji, w której muzyka i mowa brzmią „sucho" i „bez życia", a w granicznym przypadku zupełnie nienaturalnie i męcząco [5]. Taka sytuacja ma miejsce w komorach bezechowych. W takich pomieszczeniach wszystkie fale z wyjątkiem fali bezpośredniej zostają wygaszone przez układy tłumiące znajdujące się w ścianach oraz podłodze i suficie, przez co czas pogłosu jest bliski 0 sekund. Doprowadza to do pozornego zwiększenia głośności dźwięków (które nie są tam nałożone na odbite fale tła) i uczucia wzrostu ciśnienia wewnątrz ucha [9]. Dla zastosowań architektonicznych, w zależności od pojemności i przeznaczenia sali można wyznaczyć tzw. optymalny czas pogłosu, który powinien stwarzać najbardziej sprzyjające warunki akustyczne. Jest on proporcjonalny do szóstego pierwiastka z objętości pomieszczenia [5].

$$t_{60\ (opt)} = k\sqrt[6]{V} \quad (3)$$

Gdzie:

$t_{60\ (opt)}$ – optymalny czas pogłosu w pomieszczeniu [s],

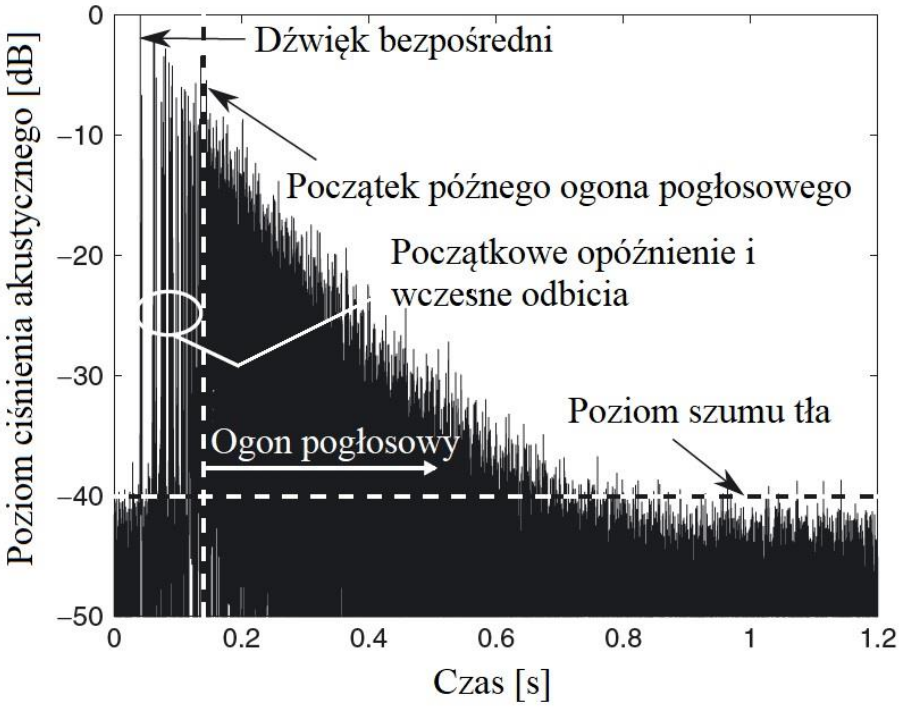
k – współczynnik zależny od przeznaczenia sali,

V – objętość pomieszczenia [m^3].

Optymalny czas pogłosu jest parametrem subiektywnym i często trudnym do realizacji w praktyce. Wynika to przede wszystkim z faktu dużej zależności optimum od przeznaczenia przestrzeni (słowo mówione, szybka, rytmiczna muzyka czy muzyka kameralna będą wymagały różnego czasu pogłosu; w studiu nagraniowym potrzebne jest duże tłumienie w celu odpowiedniej separacji ścieżek na nagraniu) oraz trudności projektanckich wynikających z zależności czas pogłosu od pojemności i stopnia zapełnienia sali [1].

Odpowiedź impulsowa pomieszczenia charakteryzuje się niejednorodnością. W jej strukturze można wyróżnić kilka następujących po sobie oddzielnych etapów, których precyzyjna symulacja jest ważnym zagadnieniem projektowania procesorów pogłosowych. Rozbrzmiewanie pogłosu jest poprzedzone tzw. początkowym opóźnieniem, czyli okresem czasu, w którym odbiorca zarejestrował już dźwięk bezpośredni (który przebywa najkrótszą drogę), ale nie zarejestrował jeszcze pierwszego powracającego odbicia fali. Krótkie opóźnienia doprowadzają do pozornego nałożenia się obu fal i zarejestrowania ich jako jednego dźwięku [1]. Długość opóźnienia początkowego daje odbiorcy informację o wielkości sali. W pomieszczeniach dużych jest ono wyraźnie słyszalne. Opóźnienie początkowe kończy się z chwilą dotarcia do odbiorcy pierwszego ze wczesnych odbić. Wczesne odbicia stanowią początek ogona pogłosowego i tworzą je pojedyncze, rozróżnialne echa dźwięku bezpośredniego, które następują przez ok. 80 ms odpowiedzi impulsowej. Duże zróżnicowanie kierunków nadchodzenia tych powtórzeń wpływa na wrażenie „głębi" pogłosu. Zgodnie z tzw. efektem Haasa, czas, po którym do odbiorcy dochodzą pierwsze odbicia ma duży wpływ na jego zdolność lokalizacji źródła dźwięku w przestrzeni. Przy dostatecznie małych opóźnieniach dźwięk bezpośredni i wczesne odbicia nie są rozróżniane i doprowadzają do odczucia zwiększonej głośności źródła. Przy większych opóźnieniach, lokalizacja źródła odbić jest rejestrowana jako tożsama z lokalizacją dźwięku bezpośredniego. Efekt ten zanika przy zwiększaniu początkowego opóźnienia, przy czym dokładna granica waha się w zależności od badań i wynosi ok. 11 – 15 ms [10]. Wczesne odbicia mają udział w kształtowaniu barwy dźwięku bezpośredniego i dodatkowo przyczyniają się do postrzegania rozmiaru sali przez odbiorcę [8]. Wraz z upływem czasu, ilość odbić dźwięku docierająca do odbiorcy wzrasta. Na wykresie rozróżnienie pojedynczych ech staje się niemożliwe, a brzmieniowo pogłos nabiera charakteru stopniowo zanikającego szumu, który jest przedłużeniem dźwięku bezpośredniego. Ten etap nosi nazwę późnego ogona pogłosowego. Parametrem tego stadium jest gęstość odbić, czyli ilość pojedynczych ech przypadających na 1 sekundę ogona. Wielkość ta jest rozpatrywana szczególnie skrupulatnie w cyfrowych efektach opartych o linie opóźniające, gdyż symulacja dużej gęstości jest wymagająca obliczeniowo, jednak jej niewystarczająca wartość

powoduje nienaturalne, drżące brzmienie [11]. Gęste odbicia miarowo wytracają swoją energię w wyniku pochłaniania przez bariery. Ogon pogłosowy zanika w czasie równym t_{60} pomniejszonym o czas trwania etapu wczesnych odbić i opóźnienia początkowego. Gdy poziom dźwięku pochodzącego od odbitych fal spadnie poniżej poziomu szumu tła, pogłos przestaje być słyszalny. Schematyczne przedstawienie etapów zjawiska rewerberacji przedstawia Rys. 2.



Rys. 2 Schematyczne przedstawienie impulsowej odpowiedzi pogłosowej z uwzględnieniem jej poszczególnych etapów [8].

Bardzo ważnym zagadnieniem związanym z odpowiedziami impulsowymi pomieszczeń i ogólną charakterystyką zjawiska pogłosu jest fakt, że czas pogłosu t_{60} jest parametrem, u którego występuje zależność od częstotliwości [1]. Sytuacja ta wynika przede wszystkim z różnej zdolności do pochłaniania i odbijania dźwięku o różnych częstotliwościach przez tłumiące materiały wykorzystywane w budownictwie. Na każdej powierzchni formuje się warstwa graniczna, która odpowiedzialna jest za absorbowanie drgań akustycznych. Warstwa ta jest tym grubsza, im bardziej nieregularna jest powierzchnia. Najgrubsza warstwa, a tym samym największe tłumienie występuje dla powierzchni porowatych [3]. Takie przeszkody charakteryzują się większym współczynnikiem odbicia dla niskich częstotliwości, a mniejszym dla częstotliwości średnich i wysokich. Czas pogłosu jest wobec tego dłuższy dla niskich tonów, a krótszy dla wyższych. Taki stan rzeczy jest odbierany jako najbardziej naturalny, a muzyka słuchana w takich warunkach pogłosowych ma „ciepłe” i „pełne” brzmienie. Z tego powodu w profesjonalnie zaprojektowanych salach koncertowych, niskie częstotliwości wybrzmiewają najdłużej w ogonie pogłosowym. Z drugiej strony, materiały twarde,

o gładkiej powierzchni i o jednolitej mikrostrukturze (takie jak marmur) mają współczynnik odbicia zbliżony wartością dla wszystkich częstotliwości [12]. Odpowiedź impulsowa w pomieszczeniach o ścianach wykonanych z takich materiałów charakteryzuje się dużo większą zawartością wysokich częstotliwości, a brzmienie pogłosu jest „zimniejsze" i „suchsze". Ta zależność wprowadziła konieczność projektowania symulacji pogłosowych, w których niskie częstotliwości wybrzmiewają dłużej niż te wysokie. Jest to powód, dla którego większość rozwiązań cyfrowych zawiera w swojej strukturze filtry dolnoprzepustowe niskiego rzędu [13].

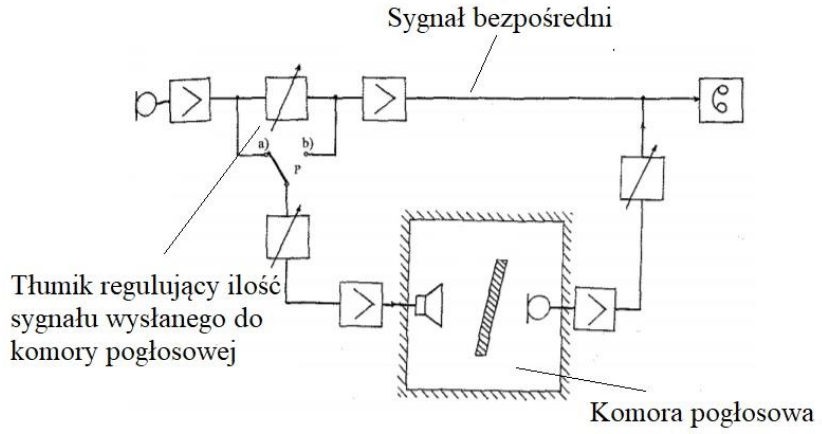
4. Metody symulacji pogłosu

Wraz z rozwojem techniki audio i realizacji dźwięku narodziła się potrzeba uzyskania większej kontroli nad zjawiskiem rewerberacji w nagraniach. Manipulacja pogłosem jest niezwykle ważnym elementem każdej produkcji dźwiękowej – pozwala na zwiększenie naturalności i intymności nagrań mowy czy partii wokalnych oraz dodaje ciepła i naturalnie brzmiącego wybrzmiewania w nagraniach instrumentalnych. Możliwość zmiany charakteru pogłosu jest także użytecznym narzędziem w dziedzinie udźwiękowienia filmu - w nagraniach np. dialogów, producent jest w stanie wierniej oddać charakterystykę akustyczną przestrzeni, w której znajdują się aktorzy, a także kreować pozorny dystans dzielący źródła dźwięku od odbiorcy [14]. Bez możliwości symulacji efektu już po procesie rejestracji dźwięku realizatorzy zdani byli na konieczność uzyskania odpowiedniego pogłosu w momencie nagrywania. Sytuacja ta była problematyczna – czas pogłosu pożądany na danej ścieżce dźwiękowej musiał wynikać z własności akustycznych pomieszczenia, w którym dokonywano rejestracji i nie mógł być edytowany w późniejszym stadium produkcji. Powodowało to oczywiście powstawanie nieprzekraczalnych barier logistycznych. Poszukiwanie rozwiązań doprowadziło do opracowania szeregu metod imitowania efektu pogłosu i dodawania go do istniejących nagrań dźwiękowych. Początkowo, stan techniki wymusił opracowanie narzędzi wykorzystujących mechaniczne oddziaływanie fal dźwiękowych z materią i fuzję wytworzonej rewerberacji z istniejącą ścieżką poprzez ponowne nagranie za pomocą odpowiedniego przetwornika odbiorczego. W późniejszych latach, poczynając od lat 60. XX wieku, wzrost znaczenia techniki cyfrowej i opracowanie podstawowej metodyki cyfrowego przetwarzania sygnałów pozwolił na wprowadzenie szerokiego wachlarza rozwiązań algorytmicznych.

4.1. Sposoby otrzymywania efektu rewerberacji za pomocą rozwiązań elektromechanicznych

Najwcześniejszym rozwiązaniem pozwalającym na edycję efektu pogłosu w istniejącym nagraniu było wykorzystanie komory pogłosowej. Jest to zamknięte pomieszczenie o silnie odbijających ścianach i najczęściej nieprostokątnym kształcie. Objętość komory waha się w zakresie 30 – 300 m^2 i jest najbardziej istotnym parametrem wpływającym na długość i walory akustyczne uzyskiwanego pogłosu [14]. Komora pogłosowa stanowi bezpośredni element toru przetwarzania sygnału akustycznego. Dźwięk odtwarzany jest w niej za pomocą głośnika, a uzyskane odbicia rejestrowane są przez mikrofon ustawiony po drugiej stronie pomieszczenia. Dzięki równoległemu połączeniu torów sygnału bezpośredniego i sygnału poddawanemu rewerberacji możliwe było ustalenie proporcji sygnału czystego i pogłosu w ostatecznym nagraniu. Takie ustawienie jest schematycznie przedstawione na Rys. 3. Zastosowanie komory pogłosowej pozwoliło na częściowe rozwiązanie problemu zależności brzmienia pogłosu od miejsca rejestracji oryginalnego sygnału. Był to jednak wyna-

lazek cechujący się wieloma niedogodnościami w użytkowaniu: czas pogłosu mógł być regulowany tylko w niewielkim zakresie za pomocą fizycznych tłumików i zależał w głównej mierze od rozmiarów i materiałów budowlanych komory; pomieszczenie musiało być częścią budynku i w ten sposób mogło być poddawane modyfikacjom w bardzo ograniczonym zakresie; wnętrze musiało spełniać wymagania dotyczące izolacji akustycznej; głośnik i mikrofon wprowadzały zakłócenia w sygnale [14]. Komory można spotkać do dziś w wielu studiach nagraniowych.

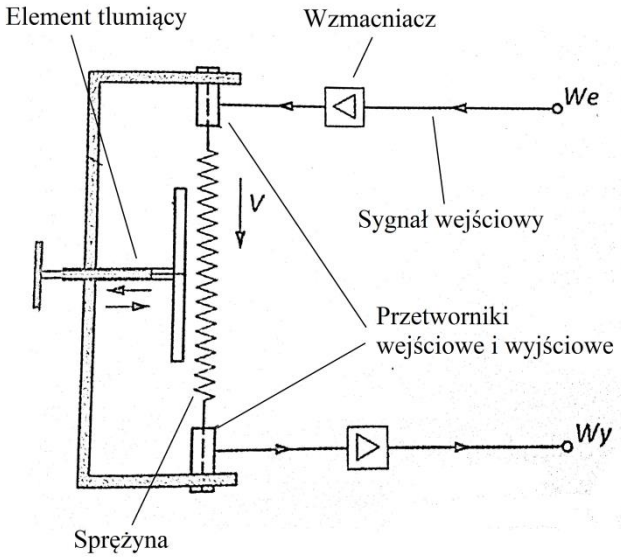


Rys. 3 Schemat toru sygnałowego wykorzystującego komorę pogłosową [15].

Kolejnym rozwiązaniem był pogłos płytowy lub pogłos Kuhla [14]. Urządzenie to do wytwarzania pogłosu wykorzystuje metalową płytę, najczęściej prostokątną i wykonaną ze stali. Równolegle do niej zawieszona jest płyta tłumiąca. Całość umieszczona jest w ramie, a odległość między płytami podlega regulacji, pozwalającej na większą niż w przypadku komory pogłosowej kontrolę czasu efektu. Do wymuszania drgań płyty stalowej wykorzystuje się przetwornik elektromechaniczny pobudzany akustycznym sygnałem wejściowym. W oddaleniu od układu wejściowego na powierzchni płyty umieszczone są dwa przetworniki elektroakustyczne, które odbierają sygnał dźwiękowy wzbogacony o pogłos. Efekt uzyskiwany w ten sposób dość znacznie różni się brzmieniowo od naturalnej odpowiedzi pomieszczenia - ma jasne i metaliczne brzmienie, przypominające szum, występują wyraźnie odrębne od siebie echa [8]. Mimo to, rozwiązanie takie zdobyło popularność ze względu na długo wyczekiwaną przez realizatorów mobilność i możliwość łatwej regulacji parametru czasu pogłosu, niezależnej od adaptacji akustycznej pomieszczenia, w którym dokonuje się nagrania bądź komory pogłosowej. Pogłos płytowy nadal pozostawał jednak urządzeniem o ustalonej, niezmiennej charakterystyce brzmieniowej, zależnej od budowy urządzenia i użytych materiałów [14].

Pogłos sprężynowy działa na bardzo podobnej zasadzie jak pogłos płytowy, jednak miejsce metalowej płyty zajmuje w nim pobudzana do drgań sprężyna [8]. Tak jak w przypadku rozwiązania z płytą, sygnał dźwiękowy jest przekazywany na sprężynę za pomocą przetwornika nadawczego, a umiejscowiony na drugim końcu urządzenia przetwornik odbiorczy pozwala na jego rejestrację wraz z wytworzoną symulacją pogłosu. Mechaniczny regulator cza-

su pogłosu operuje na zasadzie tłumika, zbliżanego lub oddalanego od drgających elementów. Schemat budowy takiego pogłosu przedstawiono na Rys. 4. Czas i brzmienie wytworzonego efektu zależy od szeregu czynników, takich jak: długość sprężyny, materiał, z jakiego została wykonana, jej średnica i liczba zwojów [14]. Konstrukcja pogłosów sprężynowych pozwalała na zapewnienie użytkownikowi możliwości głębszej ingerencji w brzmienie pogłosu – stosowano różne wymiary sprężyn w jednej obudowie, a także równoległe wykorzystanie kilku sprężyn dla jednego sygnału w celu pogłębienia efektu. Charakterystyczne brzmienie tego typu pogłosu wynika z faktu, iż fizyczny opis ruchu struktur helikalnych jest dużo bardziej skomplikowany niż płyt czy prostych przewodów. Ruch elementów sprężystych pod wpływem mechanicznego wzbudzenia sygnałem wejściowym prowadzi do wytworzenia złożonej kombinacji ech. Wynikają one zarówno z koherentnej propagacji fali wzdłuż sprężyny, jak i silnie rozproszonego wybrzmiewania związanego m. in. z poprzecznym ruchem struktury [8]. Pogłos sprężynowy zaskarbił sobie duże uznanie wśród realizatorów dźwięku dzięki niewielkim wymiarom, niskiej cenie i dość szerokiej możliwości kreacji brzmienia. Do dziś jest często spotykany w profesjonalnych studiach muzycznych, jest też często elementem bardziej złożonych konstrukcji, takich jak wzmacniacze gitarowe czy syntezatory analogowe.



Rys. 4 Schemat toru sygnałowego zawierającego pogłos sprężynowy [14].

4.2. Cyfrowe metody symulacji zjawiska pogłosu

Wraz z rozwojem technologii opartych na cyfrowej reprezentacji sygnałów, konstruktorzy efektów dźwiękowych zwrócili się w kierunku tworzenia algorytmów numerycznych, które wykorzystują jako dane wejściowe sygnał akustyczny w postaci cyfrowej (próbkowany z odpowiednią częstotliwością sygnał analogowy o skwantowanych wartościach, zależnych od rozdzielczości użytego przetwornika analogowo – cyfrowego). Wynikiem działania tych algorytmów są sygnały cyfrowe reprezentujące dźwięk bezpośredni wraz z nałożonym efek-

tem sztucznego pogłosu. Rozwiązania te cechuje zazwyczaj dużo większa niż w przypadku konstrukcji elektromechanicznych możliwość edycji parametrów brzmieniowych efektu, bardzo szybkie działanie (zakładając operacje na maszynach o dużej mocy obliczeniowej, takich jak dzisiejsze komputery) oraz uniezależnienie się od zewnętrznych urządzeń, które wymagały odpowiedniego połączenia z pozostałą częścią toru sygnałowego i zajmowały najczęściej dużo miejsca. Warunkiem koniecznym wykorzystania tego typu metod jest oczywiście posiadanie sygnału akustycznego w formie cyfrowej, co narzuca użytkownikowi konieczność wykorzystania konwerterów analogowo – cyfrowych oraz cyfrowo – analogowych, jeśli sygnał finalny ma mieć formę analogową.

Algorytmy służące do symulacji pogłosu można podzielić ze względu na bazową strategię rozwiązania. Wyróżnia się metody oparte o podejście fizyczne i metody oparte o podejście percepcyjne [13].

Metody oparte o podejście fizyczne cechują się rozpatrywaniem problemu sztucznej rewerberacji jako komputerowej emulacji zachowania dźwięku w danej przestrzeni. Wykorzystują one przestrzenną symulację zjawisk akustycznych w celu uzyskania brzmienia pogłosu opartego o istniejące (lub nieistniejące, ale o znanych wymiarach i parametrach) pomieszczenia. Najpopularniejszym rozwiązaniem tego typu jest pogłos splotowy, którego użycie wymaga dysponowania próbką odpowiedzi impulsowej. W przypadku, gdy przestrzeń poddawana symulacji ma znane i określone wymiary przestrzenne, współczynniki określające pochłanianie i odbijanie fal dźwiękowych wszystkich powierzchni oraz rozplanowane położenia i kierunkowości źródeł oraz odbiorników dźwięku, możliwe jest uzyskanie tej odpowiedzi na drodze komputerowej ekstrapolacji odbić akustycznych. Takie rozwiązania sprawdzają się bardzo dobrze w zagadnieniach związanych z akustyką budowlaną i projektowaniem sal koncertowych, gdzie mogą stanowić wskazówkę dotyczącą spodziewanego efektu akustycznego przestrzeni. Są one jednak wymagające obliczeniowo oraz mało elastyczne. Ich zaletą jest możliwość uzyskania brzmienia konkretnego pomieszczenia bez konieczności przeprowadzania w nim sesji nagraniowej.

Metody oparte o podejście percepcyjne wykorzystują względną niewrażliwość ucha człowieka na szczegóły brzmieniowe występujące w ogonie pogłosowym [8]. Ich zadaniem jest wytworzenie pogłosu, który będzie percepcyjnie nierozróżnialny od tego wynikającego z odbić fal akustycznych. Rozwój tych strategii pozwolił na opracowanie szeregu algorytmów, które w obliczeniowo wydajny i przez to szybki i responsywny sposób pozwalają na wzbogacenie sygnału o symulację rewerberacji. Nie służą one jednak wiernemu oddaniu brzmienia konkretnej przestrzeni. Podejście percepcyjne pozwoliło na dynamiczny rozwój oprogramowania generującego efekty odbiegające od reprodukcji fizycznego pogłosu, ale skupionego na uzyskaniu ciekawych lub zaskakujących wyników. Algorytmy te wyróżniają się możliwością zachowania przez użytkownika dużej kontroli nad wieloma parametrami dźwięku i dużą ilością opcji modyfikacji, co pozwala na łączenie kilku rozwiązań w jednym procesorze efektowym. Takie podejście stosowane jest np. w celu rozłącznej symulacji wczesnych odbić i ogona pogłosowego [6]. Metody percepcyjne oparte są najczęściej na podsta-

wowych operacjach przetwarzania sygnałów: filtracji wszechprzepustowej i dolnoprzepustowej, filtracji grzebieniowej i opóźnieniu z użyciem linii opóźniających.

Niniejsza praca kładzie nacisk na szczegóły implementacyjne i realizację techniczną metod percepcyjnych. Rozważania nad sposobami uzyskania fizycznie precyzyjnych symulacji zjawiska pogłosu są przeprowadzone w podstawowym zakresie dla zachowania kompletności wywodu.

4.2.1. Pogłos splotowy

Najbardziej podstawowym sposobem uzyskania efektu pogłosu na sygnale cyfrowym jest wykorzystanie matematycznej operacji splotu. Algorytm dokonuje splotu sygnału wejściowego (dźwięku, do którego chcemy dodać efekt) z uzyskaną w wyniku nagrania bądź symulacji odpowiedzią impulsową pomieszczenia w postaci cyfrowej. Taka operacja może zostać wykonana poprzez potraktowanie każdej próbki odpowiedzi jako współczynnika filtra o skończonej odpowiedzi impulsowej, a następnie dokonanie filtracji ciągu wejściowego takim filtrem [8]. Podstawową przeszkodą, która uniemożliwiła szybkie rozpowszechnienie tej metody jest bardzo duża ilość operacji mnożenia i sumowania, której wymaga taka bezpośrednia filtracja i jej szybki wzrost wraz z wydłużaniem się wektorów: wejściowego i zawierającego współczynniki filtra. Zgodnie z definicją operacji splotu [16]:

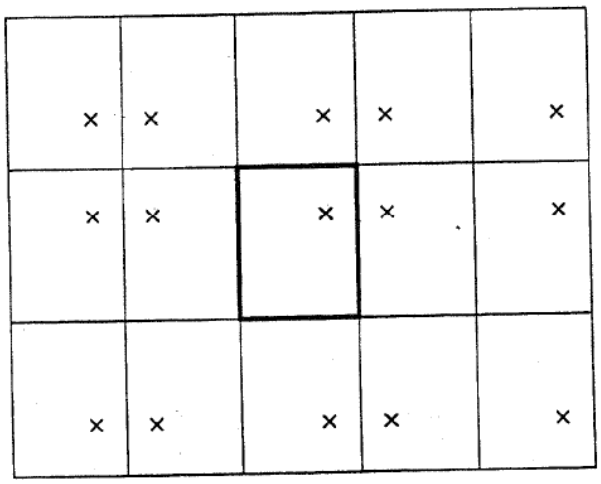
$$z(n) = x(n) * y(n) = \sum_{m=0}^{n} x(m)y(n-m) \qquad (4)$$

Gdzie:
z(n) – wektor wynikowy,
x(n), y(n) – wektory poddawane operacji splotu,

w przypadku odpowiedzi impulsowej trwającej 3 sekundy i popularnej w technice audio częstotliwości próbkowania 44,1 kHz, każda próbka wejściowa wymusza wykonanie przez procesor 44100 * 3 = 132300 operacji mnożenia i sumowania [17]. Taka ilość obliczeń wyklucza wykorzystanie procesorów splotowych w takiej formie jako efektów działających w czasie rzeczywistym. W celu zmniejszenia złożoności obliczeniowej i przyspieszenia działania algorytmu zaproponowano metody, wykorzystujące tożsamość operacji splotu w dziedzinie czasu z operacją mnożenia w dziedzinie częstotliwości [8]. Sygnał wejściowy i wektor współczynników filtra dzielone są na bloki równej długości, na których następnie wykonywana jest operacja szybkiej dyskretnej transformacji Fouriera. Bloki poddawane są mnożeniu punktowemu, a następnie wynik działania poddaje się odwrotnej dyskretnej transformacji Fouriera. Wynik działania algorytmu powstaje w wyniku odpowiedniego połączenia wektorów wynikowych w jeden, ciągły sygnał. Takie rozwiązanie wprowadza latencję pomiędzy wejściem a wyjściem układu o czasie trwania równym czasowi potrzebnemu na przetworzenie dwóch bloków, co wyklucza możliwość efektywnego działania w czasie rzeczywistym. Problem ten może zostać

załagodzony poprzez wykorzystanie bloków o rosnącej długości, każdy będący dwukrotnością długości w próbkach bloku poprzedzającego [8].

Jeżeli użytkownik nie dysponuje odpowiedzią impulsową pomieszczenia, którego pogłos chciałby zasymulować istnieje możliwość wygenerowania jej z użyciem metod przestrzennej symulacji zjawisk akustycznych [8]. Jednym z rozwiązań jest próba numerycznego rozwiązania równania falowego z wykorzystaniem metody elementów skończonych lub metody elementów brzegowych. Ten tok postępowania jest zbyt wymagający obliczeniowo dla całego zakresu słyszalności, bywa jednak wykorzystywany dla niskich częstotliwości. Częściej wykorzystywanym w symulacjach akustycznych podejściem jest model geometryczny, w którym wyróżnić można na metodę źródeł pozornych i metodę promieniową [18]. Metoda źródeł pozornych pozwala na rozpatrzenie źródła dźwięku umieszczonego w przestrzeni ograniczonej rozpraszającymi barierami jako wielu źródeł w przestrzeni nieograniczonej. Każde odbicie fali jest zastępowane przez tzw. źródło pozorne, czyli element wytwarzający fale akustyczną o właściwościach tożsamych z dźwiękiem po odbiciu, umieszczonego w pozycji wynikającej z geometrycznych zależności rozpatrywanej sceny. W celu uzyskania odpowiedzi impulsowej, fale pochodzące ze wszystkich źródeł pozornych są sumowane w miejscu, gdzie powinien znajdować się odbiorca. W praktyce, metoda ta pozwala na właściwą emulację etapu wczesnych odbić. Dla pomieszczeń prostokątnych, rozwiązanie problemu położenia źródeł pozornych jest łatwe do uzyskania i zostało przestawione na Rys. 5. To samo zadanie w przestrzeniach o nieregularnym kształcie stanowi dużo większe wyzwanie.



Rys. 5 Regularne rozmieszczenie źródeł pozornych dla prostokątnego pomieszczenia [13].

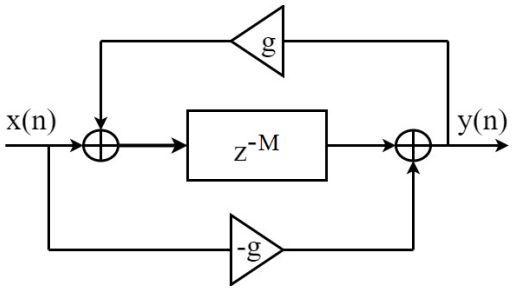
Metoda promieniowa pozwala na bardziej zaawansowane modelowanie odbić fali akustycznej. Ciągłą falę dźwiękową rozpatruje się w niej jako zbiór „promieni", czyli dyskretnych elementów niosących stały ułamek energii źródła [18]. Kierunek ruchu każdego promienia śledzony jest aż do spadku jego energii poniżej określonego początkowo progu, a utrata energii następuje w wyniku odbicia fali od bariery. Na wygenerowaną odpowiedź impulsową składają się uśrednione wkłady poszczególnych promieni, które dotarły do modelowanego za

pomocą sfery odbiornika w wybranym punkcie przestrzeni. Metoda ta jest lepiej przystosowana do symulowania odbić rozproszonych, które odpowiedzialne są za powstawanie gęstego ogona pogłosowego [13].

Pogłos splotowy, początkowo niewykorzystywany z powodu niewielkiej wydajności obliczeniowej sprzętu pozwalającego na wykonywanie operacji przetwarzania sygnałów, stał się dużo popularniejszy w dobie szybkich procesorów. Obecnie spotkać się można z dużą ilością komercyjnych procesorów pogłosowych opartych właśnie na tej metodzie przetwarzania, a ich zastosowanie wybiega daleko poza nadawanie sygnałom akustycznym wrażenia wybrzmiewania w konkretnej, istniejącej przestrzeni.

4.2.2. Metody wykorzystujące filtry wszechprzepustowe i filtry grzebieniowe

W latach 60. XX wieku rozpoczęto intensywne badania nad sposobami uzyskania efektu pogłosu z wykorzystaniem obwodów elektronicznych. W roku 1962, M. R. Schroeder jako główne problemy uzyskanych do tamtej pory rozwiązań wskazał ich niejednorodną charakterystykę amplitudowo – częstotliwościową, co doprowadzało do „koloryzacji" brzmienia, czyli jego odejścia od neutralnego stanu i podkreślenia wybranych składników tonalnych oraz niezdolność uzyskania odpowiedniej gęstości echa w ogonie pogłosowym, która sprawiała, że cechował się on drżącym, niejednorodnym wybrzmiewaniem [19]. Stwierdził, że pożądanym rozwiązaniem będzie opracowanie układu, który zapewni odpowiednią ilość powtórzeń na sekundę (optymalną wartość oceniono na 1000) i jednocześnie będzie cechował się zupełnie płaską charakterystyką amplitudowo – częstotliwościową. Metodę tą oparł na wykorzystaniu tzw. filtra wszechprzepustowego. Schemat blokowy filtra wszechprzepustowego przedstawiono na Rys. 6.



Rys. 6 Schemat blokowy filtra wszechprzepustowego.

Filtr wszechprzepustowy charakteryzuje się wzmocnieniem jednostkowym dla całego zakresu częstotliwości [19]:

$$y(n) = -gx(n) + x(n-M) + gy(n-M) \tag{5}$$

$$y(n) - gy(n-M) = -gx(n) + x(n-M) \tag{6}$$

$$Y(z) - gz^{-M}Y(z) = -gX(z) + z^{-M}X(z) \tag{7}$$

$$Y(z)(1 - gz^{-M}) = X(z)(z^{-M} - g) \tag{8}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{z^{-M} - g}{1 - gz^{-M}} \tag{9}$$

$$z = e^{j\omega T}, \omega \in \left[-\frac{\pi}{T}, \frac{\pi}{T}\right] \tag{10}$$

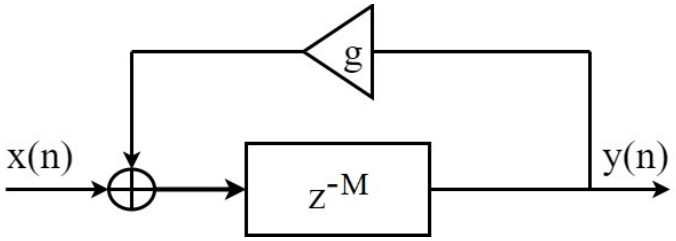
$$H\left(e^{j\omega T}\right) = H(z) \tag{11}$$

$$\begin{aligned} \left|H\left(e^{j\omega T}\right)\right| &= \left|\frac{e^{-j\omega TM} - g}{1 - ge^{-j\omega TM}}\right| = \left|e^{-j\omega TM}\right| \left|\frac{1 - ge^{j\omega TM}}{1 - ge^{-j\omega TM}}\right| \\ &= \frac{\left|1 - ge^{j\omega TM}\right|}{\left|1 - ge^{-j\omega TM}\right|} = 1 \end{aligned} \tag{12}$$

Gdzie:
M – opóźnienie sygnału w próbkach,
ω – pulsacja [rad/s], *ω = 2πf* gdzie *f* – częstotliwość [Hz],
T – okres próbkowania [s].

Inaczej sytuacja wygląda w przypadku charakterystyki amplitudowo – fazowej takiego filtra. Zwykle cechuje się ona różną wartością opóźnienia dla poszczególnych częstotliwości. Schroeder zauważył, że szeregowe połączenie takich struktur prowadzi do szybkiego wzrostu gęstości echa i konstrukcja złożona już z pięciu filtrów wszechprzepustowych o opóźnieniu dla pierwszego z nich $\tau_1 = M/T = 0.1$ s prowadzi do uzyskania gęstości 810 odbić na sekundę, wystarczająco zbliżonej do zakładanej jako optymalna [19].

Inną strukturą wprowadzoną przez Schroedera do zastosowania w symulacji efektu pogłosu był tzw. filtr grzebieniowy. Filtr ten, będący podobnie jak filtr wszechprzepustowy strukturą o nieskończonej odpowiedzi impulsowej, cechuje się charakterystycznym kształtem przebiegu charakterystyki amplitudowo – częstotliwościowej przypominającym grzebień. Schemat blokowy takiego układu przedstawiono na Rys. 7.



Rys. 7 Schemat blokowy filtra grzebieniowego ze sprzężeniem zwrotnym.

Transmitancja takiego układu przedstawia się następująco [6]:

$$y(n) = x(n - M) + gy(n - M) \tag{13}$$

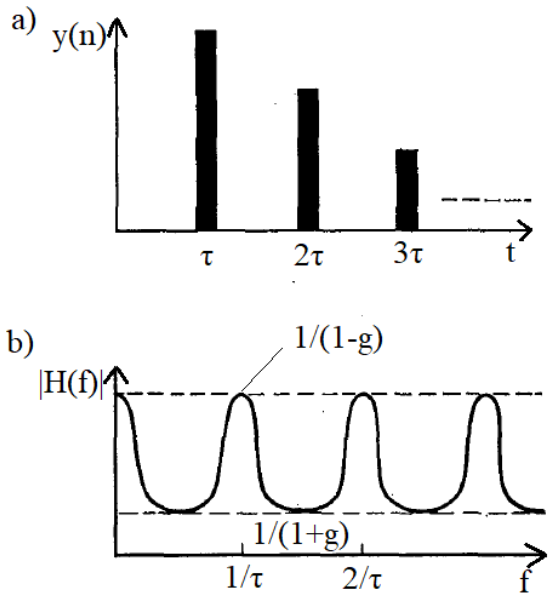
$$y(n) - gy(n - M) = x(n - M) \tag{14}$$

$$Y(z) - gY(z)z^{-M} = X(z)z^{-M} \tag{15}$$

$$Y(z)(1 - gz^{-M}) = X(z)z^{-M} \tag{16}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{z^{-M}}{1 - gz^{-M}} \tag{17}$$

Dzięki zastosowaniu takiej struktury, sygnał stanowiący wejście układu jest na wyjściu powielany, a każde kolejne echo jest oddalone od poprzedniego o M próbek i osłabione w stosunku g^N, gdzie N to liczba porządkowa przyporządkowywana kolejnym echom (N = 0, 1, 2, 3…). Nawet bez analizy matematycznej łatwo więc zauważyć, że warunkiem zachowania stabilności filtra jest $|g| < 1$. Sumowaniu podlegają wtedy osłabiane kopie sygnału wejściowego, co pozwala na jego kompletny zanik. Ten sam warunek musi być spełniony w przypadku filtra wszechprzepustowego. Na Rys. 8 przedstawiony został schemat odpowiedzi impulsowej i charakterystyki amplitudowo – częstotliwościowej filtra grzebieniowego ze sprzężeniem zwrotnym.



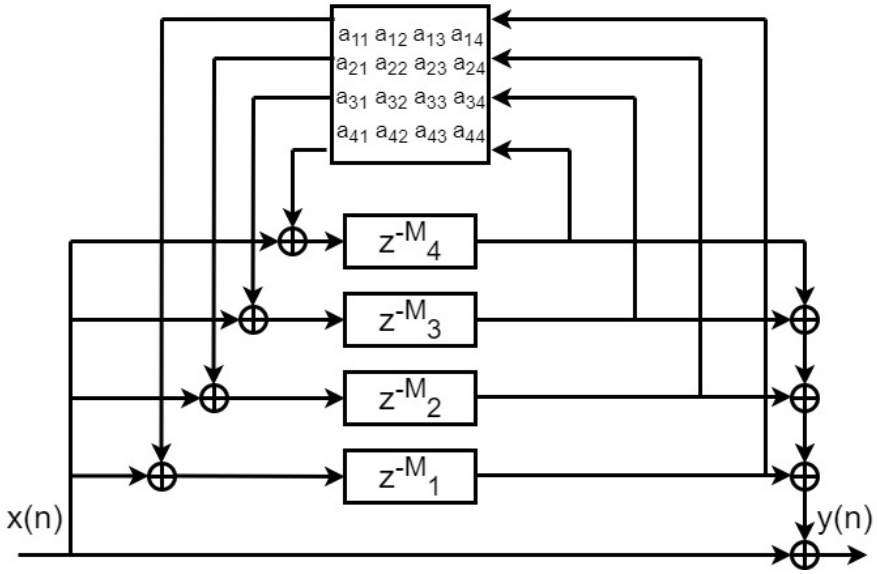
Rys. 8 a) Schematyczne przedstawienie odpowiedzi impulsowej filtra grzebieniowego ze sprzężeniem zwrotnym. b) Schematyczne przedstawienie charakterystyki amplitudowo - częstotliwościowej filtra grzebieniowego ze sprzężeniem zwrotnym [19].

Argumentem Schroedera przemawiającym za użyciem filtrów grzebieniowych, początkowo rozpatrywanych w dziedzinie symulacji pogłosu jako niedoskonałej wersji filtra wszechprzepustowego, było występowanie fluktuacji w odpowiedzi impulsowej prawdziwych pomieszczeń. Fluktuacje te mają średnio 10 dB, a w skrajnych przypadkach osiągają 40 dB i według badań przeprowadzonych w Laboratoriach Bella nie są rejestrowane przez słuch człowieka w przypadku, gdy ich gęstość jest wystarczająco wysoka [19]. Zaproponowano równoległe połączenie filtrów grzebieniowych i ustalono, że dla uzyskania odpowiedniej gęstości fluktuacji w dziedzinie częstotliwości i gęstości echa w dziedzinie czasu wymagane jest użycie trzech lub czterech takich struktur, połączonych szeregowo z dwoma filtrami wszechprzepustowymi.

Z biegiem czasu, projekt Schroedera doczekał się znaczącej ilości ulepszeń i redefinicji, które pozwalają na tworzenie procesorów efektowych o dobrej jakości symulacji i dość szerokim zakresie kontroli nad parametrami brzmieniowymi. Wartymi nadmienienia osiągnięciami w tej dziedzinie było użycie tzw. zagnieżdżonego filtra wszechprzepustowego w symulacji sztucznego pogłosu, zaproponowane przez W. G. Gardnera [20]. Wykorzystał on fakt, iż struktura filtra wszechprzepustowego, w którym linia opóźniająca z^{-M} zostanie zastąpiona kolejnym filtrem wszechprzepustowym zachowuje warunek jednostkowego wzmocnienia dla wszystkich częstotliwości, jednocześnie pozwalając na wzrost ilości zasymulowanych odbić w czasie, co nie ma miejsca w przypadku standardowego układu. Zanotował on dzięki temu znaczne polepszenie brzmienia rewerberacji. Flagowym przykładem wykorzystania odkryć Schroedera do zaprojektowania dobrej klasy efektu pogłosu jest projekt J. A. Moorera [6]. Moorer wykorzystał filtry grzebieniowe i wszechprzepustowe połączone według oryginalnego konceptu, proponując zestaw parametrów służących otrzymaniu optymalnej symulacji przy niewielkim zużyciu zasobów maszyny obliczeniowej. Użył on filtrów dolnoprzepustowych pierwszego rzędu zagnieżdżonych w pętlach sprzężenia zwrotnego filtrów grzebieniowych, co pozwoliło na zamodelowanie zależności czas pogłosu od częstotliwości, a także zestawu linii opóźniających do niezależnej od ogona pogłosowego generacji wczesnych odbić. Dokładny opis tego rozwiązania przedstawiony zostanie w rozdziale 5., wraz z prezentacją prototypowej implementacji algorytmu i wyników jego działania.

4.2.3. Sieci pogłosowe FDN

Sieci pogłosowe FDN (od angielskiego *feedback delay networks*) zostały po raz pierwszy zaproponowane przez J. Stautnera i M. Puckette'a w 1982 roku [21]. Stanowiły one rozszerzenie filtra grzebieniowego ze sprzężeniem zwrotnym. Linia opóźniająca została zastąpiona w nich szeregiem linii o zróżnicowanych długościach, a parametr g kwadratową macierzą sprzężenia zwrotnego o rozmiarze równym ilości wprowadzonych linii. Struktura taka w najprostszym wydaniu została przedstawiona na Rys. 9.



Rys. 9 Schemat blokowy podstawowej struktury sieci pogłosowej FDN.

Sieci pogłosowe w zamyśle służyły jak najefektywniejszemu zwiększeniu gęstości echa. Największy skok rozwojowy metoda ta wykonała wraz z pracą J.-M. Jota, który usystematyzował wypracowane dotychczas rozwiązania i zaproponował metodologię projektową, która jest stosowana do dziś w celu uzyskania wydajnych i dobrze brzmiących procesorów sztucznego pogłosu [11]. Największy wpływ na ostateczne działanie efektu opartego o sieci FDN ma wybór macierzy sprzężenia zwrotnego oraz wzajemny stosunek wybranych długości linii opóźniających. W trakcie rozwoju technik symulacji pogłosu zauważono, że odpowiedź impulsowa pomieszczenia powinna przypominać wykładniczo zanikający w czasie biały szum. Pozbawiona elementów tłumiących sieć pogłosowa powinna więc w odpowiedzi na impuls jednostkowy generować na wyjściu sygnał jak najbardziej zbliżony do stałego w czasie białego szumu. Taka struktura nazwana jest przypadkiem bezstratnym [22]. Oryginalnie zaproponowana macierz do zastosowania w sieci pogłosowej miała następujący kształt [21]:

$$A = \frac{g}{\sqrt{2}}\begin{bmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} \tag{18}$$

Gdzie:
g – współczynnik wzmocnienia, $|g| \leqslant 1$.

Stautner i Puckette ustalili, że stabilność takiego systemu jest zapewniona wtedy, gdy macierz sprzężenia zwrotnego jest unitarna (a więc ortogonalna w przypadku, gdy wszystkie elementy są rzeczywiste) [13]. Inne rozwiązania skupiały się głównie na zastosowaniu dwóch rodzajów macierzy: macierzy Hadamarda i macierzy Housholdera [22]. Macierz Hadamarda jest macierzą składającą się z elementów o wartości wyłącznie -1 lub 1, ułożonych tak, że w każdych sąsiednich kolumnach i każdych sąsiednich wierszach połowa składników różni się znakiem, a połowa składników zachowuje ten sam znak [23]. Własnością takiej macierzy

o wymiarach n x n jest posiadanie przez nią największej możliwej wartości bezwzględnej wyznacznika wśród macierzy zespolonych o takich wymiarach o wartościach $|a_{ij}| \leqslant 1$. Wpływa to na jej optymalną zdolność rozpraszania. Przykładowa macierz Hadamarda:

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \tag{19}$$

Rozwiązaniem zaproponowanym przez Jota była macierz Housholdera. Charakteryzuje się ona własnościami obliczeniowymi, które pozwalają na zaprojektowanie wydajnych implementacji operacji mnożenia z wektorami wejściowymi i tym samym przyspieszenie całego algorytmu. Macierz taką otrzymuje się za pomocą następującej metody, nazwanej refleksją lub odbiciem Householdera [13]:

$$A_N = J_N - \frac{2}{N} u_N u_N^T \tag{20}$$

Gdzie:

J_N - macierz permutacji o rozmiarze N x N,

u_N – wektor kolumnowy jedynek o rozmiarze N x 1.

Macierz taka zawiera dwie wartości, obie niezerowe, co pozwala na bardzo szybki i efektywny wzrost gęstości echa w ogonie pogłosowym.

Drugim zagadnieniem wpływającym w bardzo istotny sposób na końcowy rezultat brzmieniowy procesora opartego o sieci pogłosowe FDN jest liczba linii opóźniających i wartości tych opóźnień dla każdej z nich. Najczęściej powtarzany w literaturze warunek stanowi, że muszą być to liczby względnie pierwsze [22]. Parametryzacja taka pozwala na uzyskanie odpowiednio płynnego zaniku ogona pogłosowego i uniknięcie wystąpienia w nim pulsujących, drgających lub w jakikolwiek sposób okresowo powtarzających się tonów. Długości linii muszą być dobrane tak, aby w wybrzmiewaniu zachowana była odpowiednio wysoka „gęstość modów", którą Schroeder powiązał z maksimami charakterystyki amplitudowo – częstotliwościowej filtrów grzebieniowych [24]. W pojedynczym filtrze, maksima te rozmieszczone są co $1/\tau$ Hz, gdzie τ oznacza opóźnienie w sekundach. Dla czasu rewerberacji wynoszącego 1 sekundę, odpowiednia według Schroedera gęstość wynosi 0,15 1/Hz i wartość ta rośnie liniowo wraz z wydłużaniem się czasu pogłosu. Podana przez niego formuła przedstawia się więc następująco:

$$M \geq 0{,}15 t_{60} f_s \tag{21}$$

Gdzie:

M – suma długości wszystkich linii opóźniających w próbkach,

f_s – częstotliwość próbkowania [Hz].

Zachowanie tych dwóch warunków powinno prowadzić do uzyskania odpowiednio gęstej i jednorodnej odpowiedzi algorytmu pogłosowego nawet na bardzo krótkie wymuszenia. Istnieją rozwiązania, które w celu zapewnienia najbardziej percepcyjnie dokładnego wybrzmiewania stosują zmienne w czasie długości linii opóźniających, wykorzystując do ich regulacji np. sinusoidalne oscylatory [22]. Takie układy wymagają dużo dokładniejszego strojenia, jednak pozwalają na uzyskanie efektu bardzo wysokiej jakości. Gęstość echa jest również nierozerwalnie związana z liczbą wykorzystanych linii opóźniających. W literaturze najczęściej napotykane wartości to 4, 8, 12 i 16. W trakcie realizacji pracy ustalono, że percepcyjnie zadowalające rezultaty uzyskać można przy wykorzystaniu 12 linii. Mniejsza liczba prowadzi do bardzo wyraźnych drgań ogona pogłosowego w przypadku sygnałów wejściowych o impulsowym charakterze.

Aby otrzymać rzeczywisty symulator, w strukturze przypadku bezstratnego należy umieścić filtry, które spowodują stopniowy zanik ogona pogłosowego. Zgodnie z fizyczną charakterystyką zjawiska rewerberacji, filtry te muszą mieć charakter dolnoprzepustowy i nie mogą wprowadzać wzmocnienia większego niż jednostkowe [22]. Są one połączone szeregowo z liniami opóźniającymi. Mogą one charakteryzować się łagodnym spadkiem charakterystyki amplitudowo – częstotliwościowej, co pozwala na wykorzystanie filtrów o nieskończonej odpowiedzi impulsowej pierwszego rzędu. Jest to najbardziej wydajne implementacyjnie rozwiązanie. Odpowiednia kalkulacja współczynników takich filtrów w zależności od wybranych przez użytkownika parametrów długości czasu pogłosu i tłumienia wysokich częstotliwości prowadzi do poprawnego zamodelowania tych własności w efekcie. Tak zaprojektowany algorytm pozwala na bardzo szeroką kontrolę nad oczekiwaną symulacją ogona pogłosowego, dzięki czemu procesory efektowe oparte o sieci pogłosowe FDN są używane najczęściej w wysokiej jakości komputerowych programach dźwiękowych.

Dokładny opis metodyki parametryzacji algorytmu został szczegółowo przytoczony w rozdziale 5., dotyczącym prototypowej implementacji efektów pogłosowych.

4.2.4. Cyfrowe symulacje metod analogowych i inne rozwiązania

Rozwój cyfrowych metod symulacji pogłosu umożliwił inżynierom dźwięku odstąpienie od stosowania w tej dziedzinie nieporęcznych i mało elastycznych urządzeń elektromechanicznych. Mimo to, efekty te ze względu na charakterystyczne brzmienie i ograniczoną dostępność ciągle stanowią obiekt fascynacji i wbrew pozorom wcale nie zostały wyparte z użycia. Wobec takiego stanu rzeczy, eksperymentatorzy zajmujący się przetwarzaniem sygnałów dźwiękowych stanęli przed wyzwaniem stworzenia ich komputerowych symulacji. Zadanie to wymaga opracowania trójwymiarowego modelu fizycznego drgającej struktury, w którym w celu wiernego oddania charakterystyki brzmieniowej uwzględnić należy sposób rozchodzenia się w niej fal, jej kształt, materiały oraz wymiary. Takie emulacje są niezwykle kosztowne obliczeniowo i obecne badania skupiają się na polepszeniu ich wydajności [8].

Przykładowo, cyfrowa symulacja pogłosu płytowego wymaga znajomości następujących parametrów: gęstości płyty ρ [kg/m^3], jej modułu Younga E [kg/s^{2}m], współczynnika Poisso-

na *v*, grubości H [m] i powierzchni A [m^2], musi być znany także sposób osadzenia płyty (w celu ustalenia, które jej części mogą swobodnie wibrować). Wtedy, dla odpowiednio cienkiej płyty, jej równanie ruchu można opisać następująco [8]:

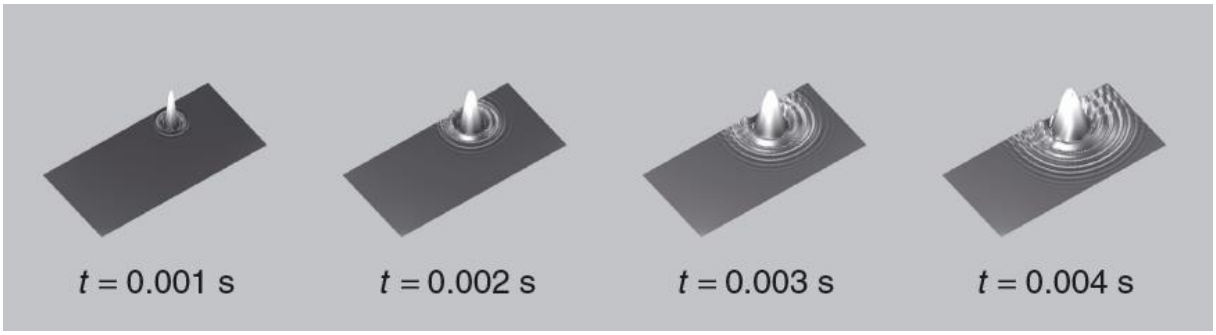
$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \nabla^2 \nabla^2 u \tag{22}$$

$$\kappa = \sqrt{EH^2/(12(1-\nu^2)\rho} \tag{23}$$

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{24}$$

Gdzie:
u – wychylenie punktu w danym punkcie płyty i czasie [m],
x, y – koordynaty określonego punktu płyty,
t – czas [s].

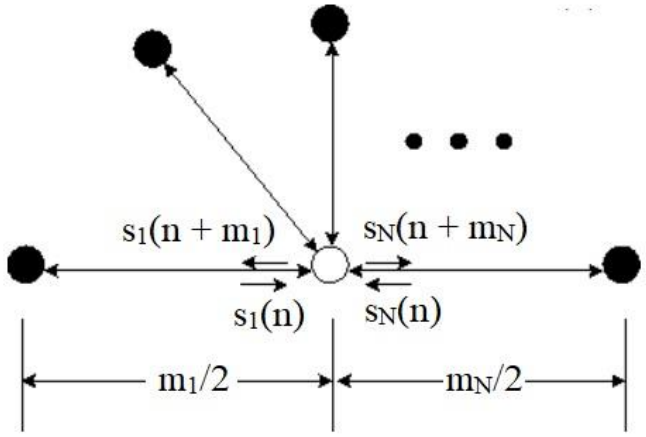
Numeryczne rozwiązanie tego równania prowadzi do uzyskania modelu wibracji płyty, co pozwala na generację jej odpowiedzi impulsowej. W ten sposób możliwe jest wykorzystanie tej metody do symulacji pogłosu dla dowolnego sygnału wejściowego. Oczywiście, brzmienie efektu nie będzie wiernym odwzorowaniem wykorzystania prawdziwej, fizycznej płyty, co wynika z m. in. konieczności dyskretyzacji układu. Na Rys. 10 zaprezentowano graficzną interpretacją takiego rozwiązania.



Rys. 10 Symulacja odkształcenia płyty stalowej w odpowiedzi na impuls w określonych chwilach czasu z κ = 0,5 [8].

Badania prowadzone są także nad cyfrowym odwzorowaniem pogłosów sprężynowych, jednak jest to zagadnienie dużo bardziej skomplikowane fizycznie ze względu na większą złożoność rozchodzenia się i dyspersji fal dźwiękowych w strukturach helikalnych. Dobre przybliżenia uzyskiwano z wykorzystaniem metody różnic skończonych, która jest jednak obliczeniowo wymagająca. Zauważono, że w celu uzyskania wystarczająco analogicznego percepcyjnie odwzorowania cyfrowego możliwe jest użycie linii opóźniających i filtrów wszechprzepustowych [8].

Inną podejściem do projektowania cyfrowych procesorów pogłosowych jest wykorzystanie metody falowodowej [25]. Wykorzystuje ona również podejście modelowania fizycznego i może być rozpatrywana jako uogólnienie sieci pogłosowych FDN. Schemat tej metody przedstawiono na Rys. 11.



Rys. 11 Schematyczne przedstawienie działania efektu pogłosu opartego o metodę falowodową [25].

Zasadniczym punktem struktury jest „węzeł rozpraszający" (ang. scattering junction), oznaczony a schemacie białą kropką. Jest on implementowany, identycznie jak w sieciach FDN, za pomocą macierzy zapewniającej bezstratne rozpraszanie sygnału wejściowego. Jest z nim połączonych *N* gałęzi, tzw. falowodów, a każda z nich zakończona jest strukturą zapewniającą idealne, nieodwracające odbicie fali (czarna kropka na schemacie). „Przejście" sygnału tą gałęzią prowadzi więc do opóźnienia go o *m* próbek, podczas gdy długość falowodu jest dwukrotnie krótsza niż korespondującej linii opóźniającej w sieci FDN. Odległość ta musi bowiem zostać pokonana dwa razy. Działanie takiej struktury falowodów można opisać następującym równaniem wektorowym [13]:

$$p^- = Ap^+ \tag{25}$$

$$p_i^+ = s_i(n) \tag{26}$$

$$p_i^- = s_i(n + m_i) \tag{27}$$

Gdzie:
A – macierz w węźle rozpraszającym.

Warto nadmienić, że metoda falowodowa jest wykorzystywana w wielu zastosowaniach związanych z modelowaniem przestrzeni akustycznych 3D, a także struktur drgających 2D, takich jak membrany [25].

5. Prototypowe implementacje wybranych metod cyfrowej symulacji pogłosu

Przegląd istniejących rozwiązań z zakresu symulacji efektu pogłosu pozwolił na uzyskanie szerokiego spektrum możliwości dotyczących ostatecznej implementacji aplikacji. W celu bliższego zapoznania się z dostępnymi strukturami, a w szczególności ze sposobami ich optymalizacji i parametryzacji, trzy algorytmy o największym znaczeniu w przemysłowych zastosowaniach zostały zrealizowane i przetestowane przy pomocy szeregu sygnałów testowych. Prototypowe realizacje porównano pod kątem brzmienia efektu, stopnia skomplikowania algorytmu, możliwości edycji parametrów oraz szansy udoskonalenia działania programu w przyszłości. Na podstawie przeprowadzonych badań wybrano metodę, która posłużyła jako podstawa działania finalnej wersji projektu.

Wersje prototypowe zostały stworzone dla następujących metod: pogłosu splotowego, pogłosu opartego o filtry wszechprzepustowe i grzebieniowe wykorzystującego strukturę Moorera oraz pogłosu opartego o sieci pogłosowe FDN, w formie zaproponowanej przez Jota. Prototypy zrealizowano w formie skryptów w języku Python wersji 3.7. Za środowisko programistyczne posłużył Spyder 3.3.6. Przetwarzanie sygnałów prowadzone było z użyciem bibliotek NumPy i SciPy, a wizualizacja wyników z użyciem biblioteki Matplotlib. Każdy ze skryptów posiadał funkcjonalność wczytania danych numerycznych z pliku WAV oraz zapisania wektorów wynikowych do pliku WAV. Do tego celu posłużyły moduły `wave` i `struct` z biblioteki standardowej języka Python. Skrypty testowe obsługiwały nieskompresowane pliki dźwiękowe stereo o rozdzielczości 16 bitów i częstotliwości próbkowania 44,1 kHz, które wczytywane były do dwuwymiarowej tablicy NumPy. Dokonano tego w następujący sposób:

```
1. wav_file = wave.open(sample_in, 'r')
2.     num_samples_sample = wav_file.getnframes()
3.     num_channels_sample = wav_file.getnchannels()
4.     sample = wav_file.readframes(num_samples_sample)
5.     total_samples_sample = num_samples_sample * num_channels_sample
6.     wav_file.close()
7.
8.     sample = struct.unpack('{n}h'.format(n = total_samples_sample), sample)
9.     sample = np.array([sample[0::2], sample[1::2]], dtype = np.float64)
```

Skrypt ten operuje na plikach audio, w których próbki z osobnych kanałów są ułożone naprzemiennie. Oznacza to, że finalna liczba próbek przechowywana w pliku wejściowym `sample_in` jest iloczynem liczby „ramek” i liczby kanałów, dostępnych za pomocą funkcji `getnframes()` i `getnchannels()` [26]. Po odczytaniu próbek pliku, kolejnym etapem jest „rozpakowanie” ich za pomocą funkcji `unpack()`, czyli interpretacja jako 16-

bitowych liczb całkowitych. Domyślnie są one bowiem zwracane przez funkcję `readframes()` jako obiekt typu `bytes` [27]. W ostatniej przytoczonej linijce zmiennej `sample` zostaje przypisana dwuwymiarowa tablica NumPy, której elementami są próbki wejściowe, podzielone już na dwa osobne kanały, w formie 64-bitowej liczby zmiennoprzecinkowej. Ta konwersja konieczna jest w celu przeprowadzenia operacji przetwarzania, które opierają się na wykorzystaniu liczb niecałkowitych. Zwiększenie rozmiaru stanowi zabezpieczenie przed ewentualnym przekroczeniem zakresu, mogącym wystąpić w toku niektórych operacji arytmetycznych (np. splotu bezpośredniego).

W analogiczny sposób dokonano zapisu sygnału wynikowego do pliku:

```
1. frame_rate = 44100.0
2. nframes = total_samples_sample
3. comptype = "NONE"
4. compname = "not compressed"
5. nchannels = 2
6. sampwidth = 2
7.
8. wav_file_write = wave.open(sample_out, 'w')
9. wav_file_write.setparams((nchannels, sampwidth, int(frame_rate), nframes,
10.                          comptype, compname))
11.
12. for s in range(nframes):
13.     wav_file_write.writeframes(struct.pack('h', signal_to_render[s]))
14.
15. wav_file_write.close()
```

Na początku, konieczne jest ustawienie parametrów zapisu za pomocą funkcji `setparams()`. Są to m.in.: liczba ramek zapisu, częstotliwość próbkowania, rozdzielczość bitowa i liczba kanałów, a także opcjonalny sposób kompresji. Zapis następuje z użyciem funkcji `writeframes()`, której argumentem są „spakowane" z powrotem do obiektu typu `bytes` próbki obu kanałów sygnału wejściowego, wcześniej skonfigurowane do występowania naprzemiennie. Warto zauważyć, że w celu zapisania do pliku wynikowego, próbki poddano zwrotnej konwersji do 16-bitowego typu całkowitego (argument `'h'` funkcji `pack()`) [27].

Każdy kod był testowany za pomocą tych samych plików dźwiękowych, o przytoczonych wyżej parametrach pozwalających na ich wczytanie do tablic NumPy. W doborze sygnałów testowych kierowano się uzyskaniem jak najszerszego spektrum brzmieniowego w celu miarodajnej oceny algorytmu w rzeczywistych zastosowaniach realizacyjnych. Sygnały testowe były oczywiście pozbawione pogłosu, lub zawierały go w niewielkim stopniu. Wykorzystano następujące dźwięki:

- klaśnięcie,
- krótką frazę gitarową wykonaną na gitarze elektrycznej,
- uderzenie w werbel,
- krótką eksplozję białego szumu,

- długie, wybrzmiewające akordy na pianinie,
- męski głos,
- dwa wysokie dźwięki na syntezatorze,
- dwa basowe, przesterowane dźwięki na syntezatorze.

Każdy skrypt generował wykres przebiegu fali sygnału wejściowego i wyjściowego, co pozwalało na szybką ocenę poprawności wybranych aspektów działania kodu. Istniała także możliwość użycia w miejsce sygnału wczytanego z pliku sygnału wygenerowanego wewnątrz skryptu. Najczęściej służyło to zbadaniu odpowiedzi układu na impuls o długości trwania jednej próbki (delta Kroneckera). Możliwość wykonania takiego testu znacznie usprawniła proces np. implementacji filtrów czy linii opóźniających oraz pomagała w badaniu algorytmów finalnych, m. in. pod kątem czasu rewerberacji czy współczynników tłumienia ogona pogłosowego lub dźwięku bezpośredniego.

5.1. Testowa implementacja pogłosu splotowego

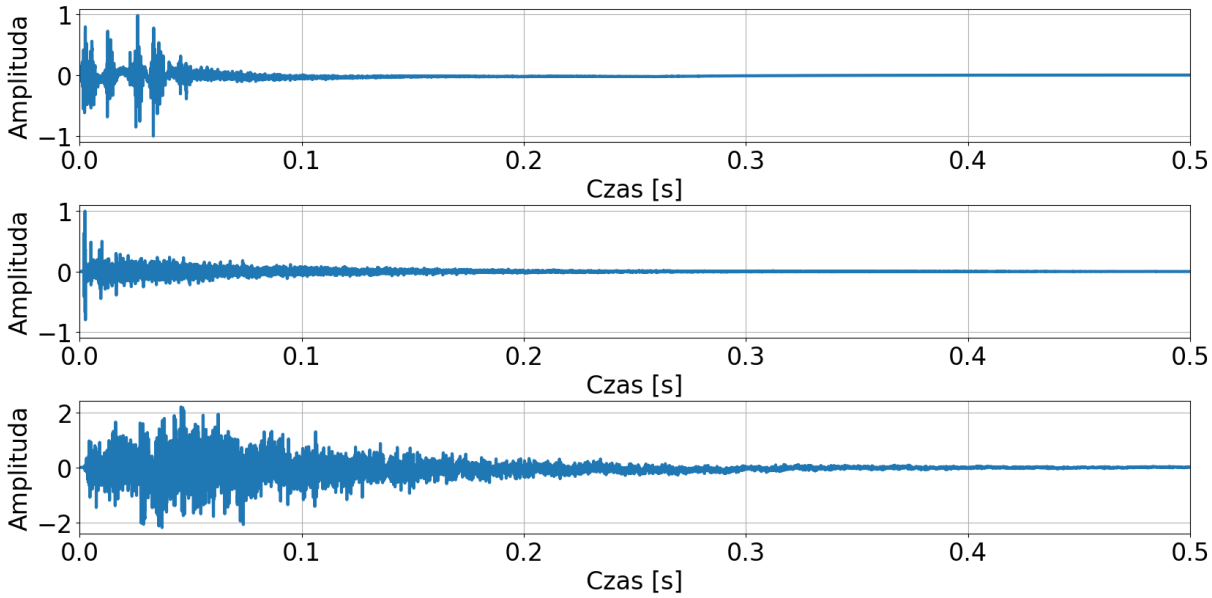
Prototypowa implementacja pogłosu splotowego wymagała posiadania oprócz sygnałów wejściowych także testowych odpowiedzi impulsowych, na których można dokonać operacji splotu. Przykładowe odpowiedzi, uzyskane metodą modelowania fizycznego, zostały zaczerpnięte z zasobów witryny Vixengo [28]. Użyta w testach algorytmu paczka zawierała 19 plików dźwiękowych WAV o 16-bitowej rozdzielczości i częstotliwości próbkowania 44,1 kHz. W skrypcie realizującym opisywaną metodę najpierw wczytano sygnał testowy i wybraną odpowiedź impulsową, a następnie wykonano operację splotu, osobno na wektorach odpowiadających sobie kanałów. Do przeprowadzenia splotu wykorzystano wbudowaną funkcję biblioteki SciPy, `convolve()`, znajdującą się w module przetwarzania sygnałów `signal`. Funkcja ta przyjmuje cztery argumenty: dwa pierwsze to wektory wejściowe, trzeci to specyfikator długości wektora wynikowego (zastosowano argument domyślny, zwracający pełny wynik operacji o długości równej sumie długości wektorów wejściowych minus jedna próbka), a czwarty to specyfikator metody. Do wyboru są trzy możliwości – `direct` wymusza bezpośredni splot w dziedzinie czasu, `fft` powoduje wykorzystanie transformaty Fouriera i mnożenie widm częstotliwościowych, a `auto` pozwala na automatyczne podjęcie decyzji na podstawie estymacji czasu potrzebnego na wykonanie operacji [29]. W celu weryfikacji hipotezy, wedle której metoda wykorzystująca szybką transformację Fouriera jest znacząco szybsza dokonano testu, angażując cztery sygnały testowe różnej długości i jedną odpowiedź impulsową. Każdy splot został wykonany dwukrotnie, raz z użyciem metody `direct` i raz z użyciem metody `fft`. Zmierzone czasy odnoszą się do czasu przetwarzania obu kanałów, każdy kanał składał się z podanej ilości próbek. Wyniki przedstawiono w Tab. 1.

Tab. 1 Wyniki testu czasu przeprowadzania operacji splotu metodą bezpośrednią i metodą wykorzystującą FFT.

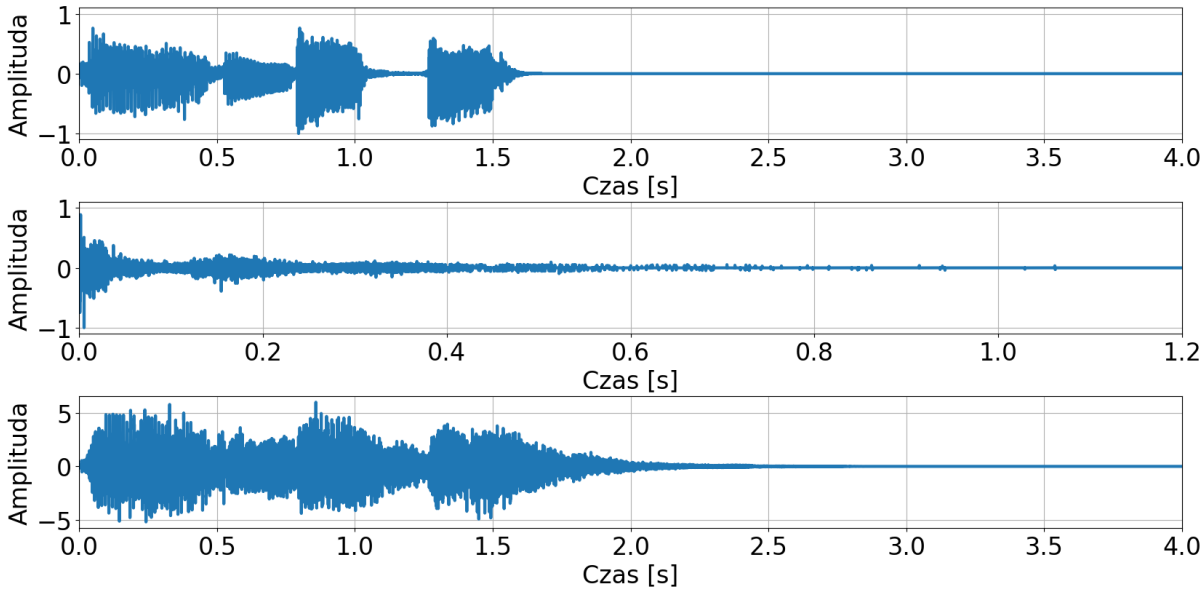
Rodzaj sygnału wejściowego	**Długość odpowiedzi impulsowej w próbkach**	**Długość sygnału wejściowego w próbkach**	**Czas przeprowadzania operacji splotu [s]**	
			`direct`	`fft`
Klaśnięcie	66367	88200	0,813	0,034
Fraza gitarowa		198450	2,304	0,057
Mowa		292420	7,452	0,067
Syntezator, wysokie tony		104792	1,155	0,028

Bez pogłębionej analizy statystycznej łatwo zauważyć, że metoda używająca FFT jest znacznie efektywniejsza od metody splotu bezpośredniego, uzyskując wynik finalny w czasie nawet dwa rzędy wielkości krótszym. Ostateczna wersja skryptu opierała się więc na takiej implementacji.

Na Rys. 12 i 13 przedstawiono rezultaty zastosowania pogłosu splotowego.



Rys. 12 Wynik działania pogłosu splotowego. Od góry: prawy kanał sygnału podstawowego, klaśnięcia; prawy kanał odpowiedzi impulsowej; prawy kanał sygnału wynikowego.



Rys. 13 Wynik działania pogłosu splotowego. Od góry: prawy kanał sygnału podstawowego, frazy gitarowej; prawy kanał odpowiedzi impulsowej; prawy kanał sygnału wynikowego.

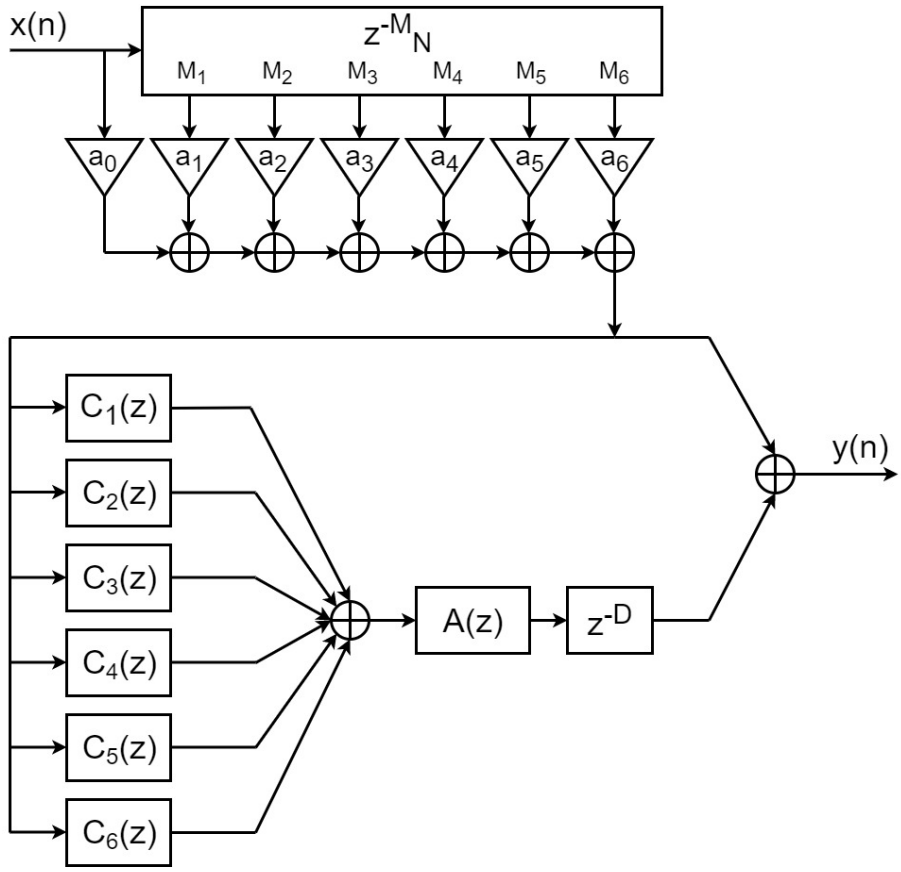
Jak łatwo zauważyć, operacja splotu doprowadza do dużego wzrostu amplitudy sygnału wynikowego w stosunku do sygnałów wejściowych. Taka sytuacja wymusza na użytkowniku konieczność odpowiedniego osłabienia wektorów początkowych przed operacją lub wektora zawierającego wynik splotu po operacji tak, aby jego ostateczna amplituda mieściła się w zakresie typu numerycznego obsługiwanego przez plik WAV. Problem ten pozostanie obecny także przy innych algorytmach pogłosu – sumowanie zwielokrotnionego przez linie opóźniające sygnału zawsze wiąże się z dużym wzrostem jego energii.

Pogłos splotowy to metoda nieskomplikowana pod względem obliczeniowym i łatwa w implementacji. Pozwala na osiągnięcie bardzo szerokiego zakresu brzmień, pod warunkiem dysponowania odpowiednią bazą odpowiedzi impulsowych pomieszczeń lub narzędzi umożliwiających sztuczną generację takich odpowiedzi. Nie występują w niej problemy dotyczące niewystarczającej gęstości odbić czy koloryzacji brzmienia, pozwala na uzyskanie powtarzalnego efektu dla każdego sygnału podstawowego. Jest precyzyjna, a jej wynik jest percepcyjnie nierozróżnialny od rzeczywistego nagrania w danej przestrzeni w przypadku operowania na odpowiedziach istniejących pomieszczeń. Do jej największych wad można zaliczyć niemożność osiągnięcia rezultatu przy dysponowaniu wyłącznie sygnałem wejściowym, duże wymagania obliczeniowe (opisane w rozdziale 4.) i zupełny brak możliwości edycji parametrów brzmieniowych na etapie działania samego algorytmu.

5.2. Testowa implementacja efektu pogłosu wykorzystującego strukturę Moorera

Drugą implementacją testową był efekt oparty na projekcie J. A. Moorera z 1979 roku [6]. Moorer w swojej pracy bazował na badaniach przeprowadzonych przez Schroedera i tak-

że opowiedział się za strukturą połączonych równolegle filtrów grzebieniowych wraz z szeregowym połączeniem filtrów wszechprzepustowych. Całość uzupełniał moduł odpowiedzialny za modelowanie wczesnych odbić, oparty o 6 linii opóźniających. Struktura Moorera przedstawiona została na Rys. 14.



Rys. 14 Schemat blokowy systemu generacji sztucznego pogłosu, zaproponowany przez J. A. Moorera. Literą C oznaczono filtry grzebieniowe, zaś A wszechprzepustowe.

Implementację systemu rozpoczęto od bloku symulacji wczesnych odbić. Jej wynikiem jest sygnał będący sumą wektora bezpośredniego nieprzesuniętego oraz 6 jego instancji opóźnionych o stałą liczbę próbek, przy czym każdy z tych wektorów przemnożony był przez osobny współczynnik wzmocnienia. Obecność tych współczynników wynika bezpośrednio ze zjawiska tłumienia fali przy każdym jej odbiciu od bariery. W swojej pracy Moorer nadmienił, że w trakcie badań wypróbowano wiele metod generacji autentycznie brzmiących wczesnych odpowiedzi pogłosowych, jednak żadna nie zapewniała optymalnej metody rozwiązania problemu. Wskazał niedogodności, takie jak możliwość zajścia efektu filtra grzebieniowego przy niedokładnie dobranych czasach opóźnień. Ostatecznie, zaproponowano ustalone metodą prób i błędów parametry dla układu z sześcioma i osiemnastoma liniami opóźniającymi. Ze względu na prototypowy charakter implementacji, zdecydowano się na zastosowanie w skrypcie tej pierwszej wersji. Zastosowane parametry zamieszczono w Tab. 2.

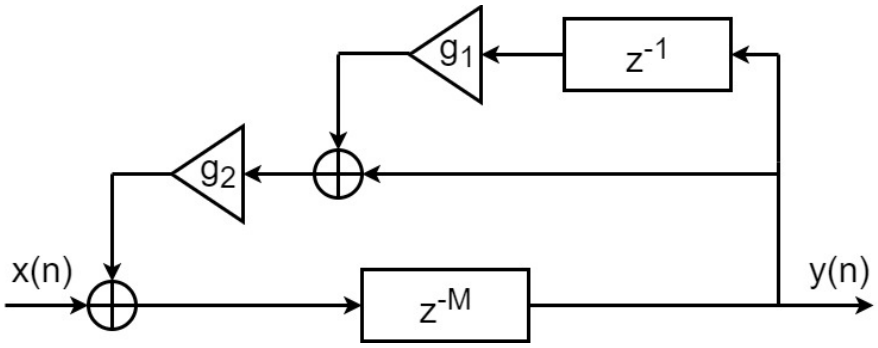
Tab. 2 Parametry bloku modelowania etapu wczesnych odbić w procesorze Moorera [6].

Numer linii opóźniającej n	**Opóźnienie M_n [s]**	**Współczynnik wzmocnienia a_n**
0 (dźwięk bezpośredni)	0	1,00
1	0,0199	1,02
2	0,0354	0,818
3	0,0389	0,635
4	0,0414	0,719
5	0,0699	0,267
6	0,0796	0,242

Cały sygnał wyjściowy opisanego modułu jest sumowany z sygnałem będącym wynikiem działania bloku symulacji ogona pogłosowego. Przed podjęciem badań, Moorer zauważył, że najważniejszymi problemami występującymi w układach pogłosowych opartych na filtrach były:

- Narastanie gęstości ogona pogłosowego w czasie, tym dłuższe im więcej filtrów użyto w symulatorze. Jest to efekt niepożądany, gdyż ogon pogłosowy powinien zaczynać się gęstym wybrzmiewaniem, które powinno zanikać w czasie.
- Bardzo duża zależność brzmienia ogona pogłosowego od doboru parametrów użytych filtrów. Niewielka zmiana opóźnienia w strukturze filtru może stanowić różnice między spójnym wybrzmiewaniem a bardzo nieprzyjemnie brzmiącymi drżeniami i niejednorodnościami.
- Wspominany często „metaliczny" ton, związany z nieprawidłową charakterystyką amplitudowo – częstotliwościową układu.

Zaprojektowana struktura miała zapewnić jak najdokładniejsze rozwiązanie tych problemów. Najważniejszą modyfikacją było wprowadzenie filtra dolnoprzepustowego w pętle sprzężenia zwrotnego filtrów grzebieniowych. Taki moduł znalazł się w miejscu każdego *C(z)* zaznaczonego na schemacie blokowym. Prezentuje go Rys. 15.



Rys. 15 Schemat blokowy $C_N(z)$.

Transmitancja takiego prostego filtra dolnoprzepustowego przedstawia się następująco [6]:

$$T(z) = \frac{1}{1 - g_1 z^{-1}} \tag{28}$$

Skoro filtr grzebieniowy zachowuje stabilność, gdy wzmocnienie pętli sprzężenia zwrotnego jest na moduł mniejsze niż 1, układ z dodatkowym filtrem będzie stabilny, gdy iloczyn wzmocnień pochodzących od g_2 i $T(z)$ będzie mniejszy od 1. Dolnoprzepustowość $T(z)$ wskazuje, że maksymalna wartość jego wzmocnienia wystąpi dla $\omega = 0$ i wobec tego wynosi:

$$T\left(e^{j\omega T}\right) = \frac{1}{1 - g_1 e^{-j\omega T}} \tag{29}$$

$$\omega = 0 \tag{30}$$

$$T(1) = \frac{1}{1 - g_1} \tag{31}$$

Co prowadzi do następującego kryterium stabilności każdego z $C_n(z)$:

$$g = \left|\frac{g_2}{1 - g_1}\right| < 1 \tag{32}$$

Gdzie:
g – całościowe wzmocnienie pętli sprzężenia zwrotnego filtra grzebieniowego.

Ostatecznie, transmitancja filtra grzebieniowego z filtrem dolnoprzepustowym w pętli sprzężenia zwrotnego, który zastosowano w skrypcie przedstawia się następująco:

$$\begin{aligned} C_N(z) = \frac{z^{-M_N}}{1 - g_2 T(z) z^{-M_N}} &= \frac{z^{-M_N}}{1 - \frac{g_2 z^{-M_N}}{1 - g_1 z^{-1}}} \\ &= \frac{z^{-M_N}(1 - g_1 z^{-1})}{1 - g_1 z^{-1} - g_2 z^{-M_N}} = \frac{z^{-M_N} - g_1 z^{-(M_N+1)}}{1 - g_1 z^{-1} - g_2 z^{-M_N}} \end{aligned} \tag{33}$$

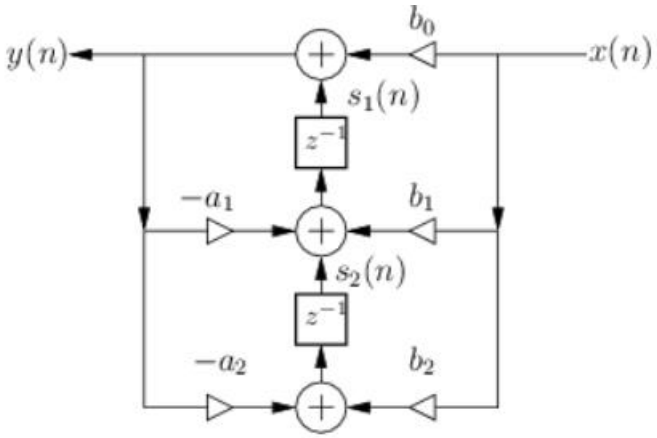
Zastosowanie filtra tak niskiego rzędu nie pozwala oczywiście na precyzyjne modelowanie osłabienia wysokich częstotliwości w rzeczywistych pomieszczeniach, jednak wymaga użycia wyłącznie jednego dodatkowego dodawania. Percepcyjnie jest to rozwiązanie wystarczające, a finalny efekt rzeczywiście przejawia dość znaczne tłumienie wraz ze wzrastającą częstotliwością. Ustalono także, że całościowo struktura powinna zawierać sześć takich fil-

trów połączonych szeregowo z jednym filtrem wszechprzepustowym. Zaproponowano wartości współczynników i opóźnień ustalone za pomocą testów odsłuchowych. Długości linii opóźniających mają być rozłożone liniowo w stosunku 1:1,5 przy stałej wartości parametru g kontrolującego czas pogłosu (0,83 przy zastosowaniu wskazówek autora ma dawać w rezultacie $t_{60} \approx 2$ s). Parametr g_2 ma być równy $g(1-g_1)$, co wynika z kryterium stabilności, a filtr wszechprzepustowy powinien charakteryzować się opóźnieniem ok. 6 ms i wzmocnieniem ok. 0,7. Autorzy podkreślili, że wartości te nie są wynikiem precyzyjnego rozumowania, a jedynie wskazówką dotyczącą otrzymania wiarygodnego efektu brzmieniowego. Proponowane wartości g_1 podane były dla częstotliwości próbkowania 25 kHz i 50 kHz, co wymagało wykonania prostej interpolacji w celu ich estymacji dla 44,1 kHz. Takie podejście było z resztą zaproponowane przez samego autora. Tab. 3 przedstawia parametry użyte w finalnym skrypcie. Założono f_s = 44,1 kHz.

Tab. 3 Parametry filtrów w bloku modelowania ogona pogłosowego struktury Moorera [6].

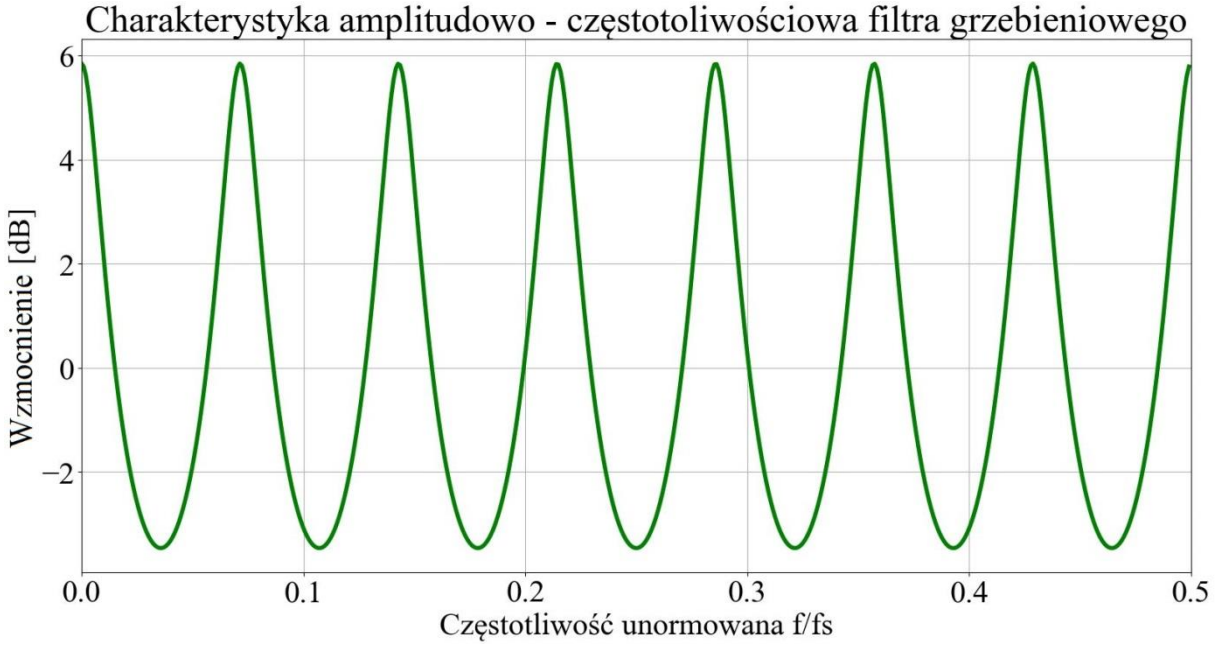
Filtr	**Opóźnienie [ms] / Opóźnienie w próbkach**	g_1
$C_1(z)$	50 / 2205	0,41
$C_2(z)$	56 / 2469	0,43
$C_3(z)$	61 / 2690	0,45
$C_4(z)$	68 / 2998	0,47
$C_5(z)$	72 / 3175	0,48
$C_6(z)$	78 / 3439	0,50
A(z)	6 / 286	0,70

W celu implementacji filtrów skorzystano z funkcji `lfilter` pochodzącej z modułu `signal` SciPy. Funkcja ta przyjmuje jako argumenty wektory współczynników filtra - licznika i mianownika orz sygnał wejściowy, którego przefiltrowana wersja jest następnie zwracana. Współczynniki filtrów obliczane są wprost z wyprowadzonych wcześniej funkcji przejścia przy każdym uruchomieniu skryptu, uwzględniając zastosowane przez użytkownika wzmocnienia i opóźnienia. Funkcja `lfilter` wykorzystuje w celu obliczenia wyniku strukturę transponowaną realizacji bezpośredniej układu IIR (ang. *direct II transposed structure*), w której dla każdej próbki obliczane są tzw. zmienne stanu [30]. Ich liczba jest równa rzędowi filtru i służą one przechowywaniu liczb potrzebnych do kalkulacji wartości sygnału opartych na przeszłych i przyszłych próbkach wektora początkowego. Schemat takiej struktury przedstawiony jest na Rys. 16.

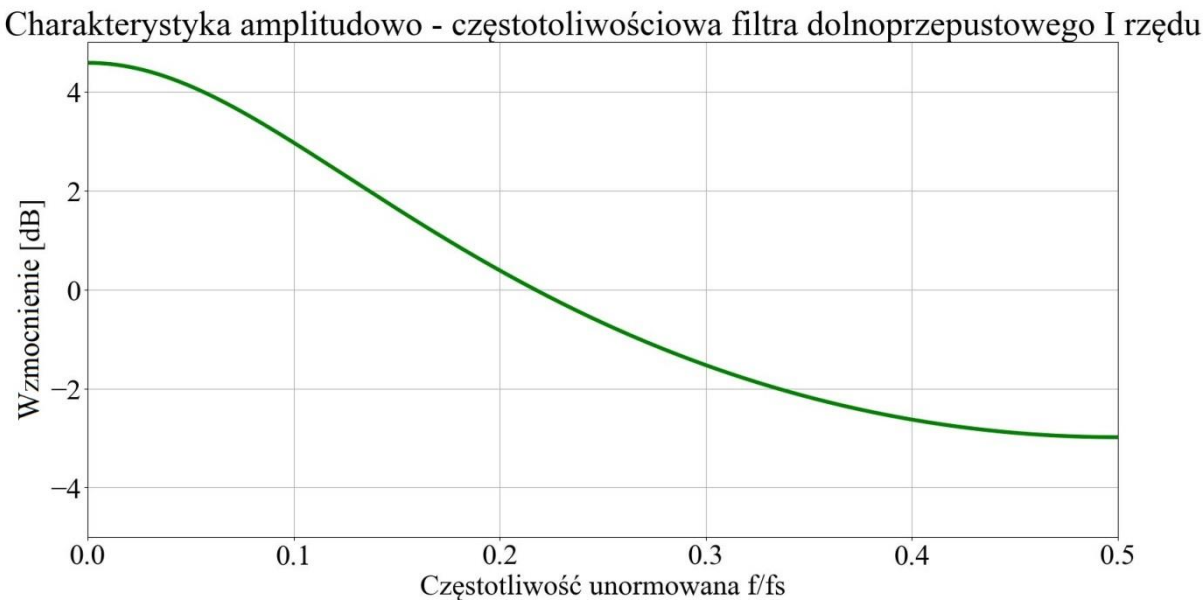


Rys. 16 Schemat blokowy przykładowej struktury transponowanej realizacji bezpośredniej układu o nieskończonej odpowiedzi impulsowej, zastosowanej w funkcji `scipy.signal.lfilter()`. $s_N(n)$ oznaczają wektory zmiennych stanu [31].

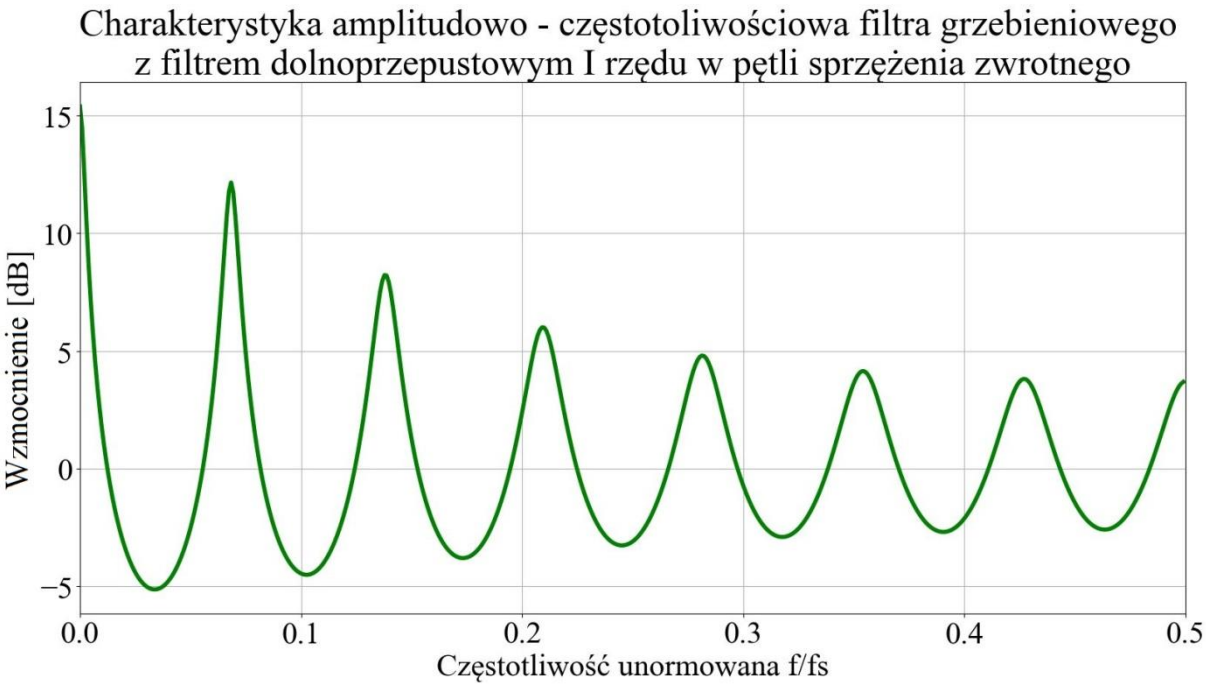
W celu potwierdzenia poprawności implementacji, dla każdego filtra wygenerowano w formie graficznej charakterystykę amplitudowo – częstotliwościową. Otrzymane wyniki były zgodne z oczekiwanymi. Charakterystyki przedstawiono na Rys. 17 – 20. Zbadano filtr $C_1(z)$ bez tłumienia wysokich częstotliwości, wyizolowany filtr umieszczony w pętli sprzężenia zwrotnego $C_1(z)$, kompletną strukturę $C_1(z)$ dla $g = 0.83$ i filtr $A(z)$. W celu lepszej wizualizacji wyników, w każdym z filtrów zastosowano opóźnienie wynoszące 15 próbek (dla dużych wartości opóźnień, „zęby" grzebienia występują bardzo gęsto. W filtrze $A(z)$ zmiana wielkości opóźnienia nie wpływa na kształt charakterystyki).



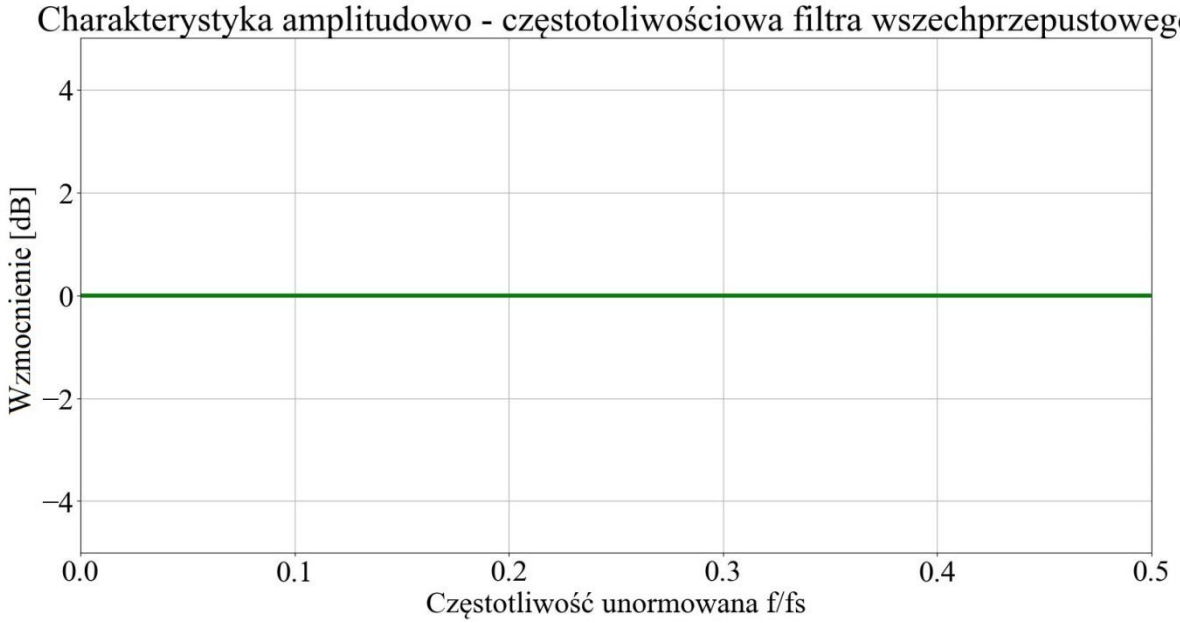
Rys. 17 Charakterystyka amplitudowo - częstotliwościowa filtra grzebieniowego dla g = 0,4897 i M = 15.



Rys. 18 Charakterystyka amplitudowo - częstotliwościowa filtra dolnoprzepustowego IIR I rzędu dla g = 0,41.



Rys. 19 Charakterystyka amplitudowo - częstotliwościowa dla pełnej struktury $C_1(z)$ z M = 15.



Rys. 20 Charakterystyka amplitudowo - częstotliwościowa filtra wszechprzepustowego dla $g = 0{,}7$ i $M = 15$.

W celu dopasowania czasowego etapu wczesnych odbić i ogona pogłosowego, cały sygnał wyjściowy z sieci filtrów poddany był dodatkowemu opóźnieniu. Wielkość tego opóźnienia oszacowano na podstawie analizy odpowiedzi impulsowej układu zakładając, że początek gęstego ogona pogłosowego występuje bezpośrednio po ostatnim ze wczesnych odbić i ustalono na $D = 1800$ próbek. Taka wartość odpowiada, łącznie z opóźnieniem samego ogona w stosunku do bezpośredniego sygnału, czasowi trwania etapu dyskretnych odbić ok. 90 ms. Oczywiście, parametr ten można edytować w celu nadania uzyskanemu efektowi wrażenia pochodzenia z „większego pokoju" lub z „mniejszego pokoju".

Zaprezentowane dotychczas rozwiązania odnosiły się do pojedynczego kanału, prototyp zaś w założeniu operować miał na stereofonicznych sygnałach wejściowych, produkując również stereofoniczny sygnał wyjściowy. W tym celu, opisana architektura powtórzona została dla obydwu kanałów. W takiej sytuacji, zasadne jest zastosowanie metody dekorelacji sygnałów należących do dwóch kanałów. Dekorelacja sygnałów jest to usunięcie ich wzajemnej korelacji, czyli statystycznej zależności występującej pomiędzy nimi [32]. W dziedzinie wielokanałowych sygnałów audio, dekorelacja służy zwiększeniu pozornej przestrzeni odsłuchowej i wzmocnieniu odczucia dochodzenia dźwięku do odbiorcy nie z jednego punktu, a z całego otoczenia (tak jak ma to miejsce w przypadku rzeczywistego pogłosu w pomieszczeniach) [8]. W opisywanej implementacji prototypowej zdecydowano się na użycie metody separacji stereofonicznej wzorowanej na programie Freeverb – aplikacji w języku C++, opartej o strukturę wykorzystującą filtry wszechprzepustowe i grzebieniowe, która jest dostępna w domenie publicznej [22]. Pierwszą z metod odmiennego przetwarzania lewego i prawego kanału jest zastosowanie zmiennej całkowitej, której wartość dodawana jest do każdej wartości długości linii opóźniającej w próbkach. W ten sposób, sygnał z każdego kanału jest transformowany przez inny zestaw filtrów, mimo że wzajemne relacje pomiędzy długościami linii

pozostają niezmienione. Domyślna wartość tej zmiennej wynosi 23, jednak jej modyfikacja w zakresie mniejszym niż długości linii nie wpływa znacząco na finalny efekt brzmieniowy. Drugą metodą jest obliczanie ostatecznego wyniku dla obu kanałów na podstawie następującej zależności:

$$\begin{bmatrix} outputL \\ outputR \end{bmatrix} = dry \begin{bmatrix} inputL \\ inputR \end{bmatrix} + \begin{bmatrix} wet1 & wet2 \\ wet2 & wet1 \end{bmatrix} \begin{bmatrix} outL \\ outR \end{bmatrix} \tag{34}$$

Gdzie:
`outputL, outputR` – wektory wynikowe dla kanału odpowiednio lewego i prawego,
`dry` – parametr kontrolujący udział sygnału bezpośredniego w wektorach wynikowych,
`inputL, inputR` – wektory wejściowe dla kanału odpowiednio lewego i prawego,
`outL, outR` – wektory wyjściowe struktury filtrów dla kanału odpowiednio lewego i prawego.

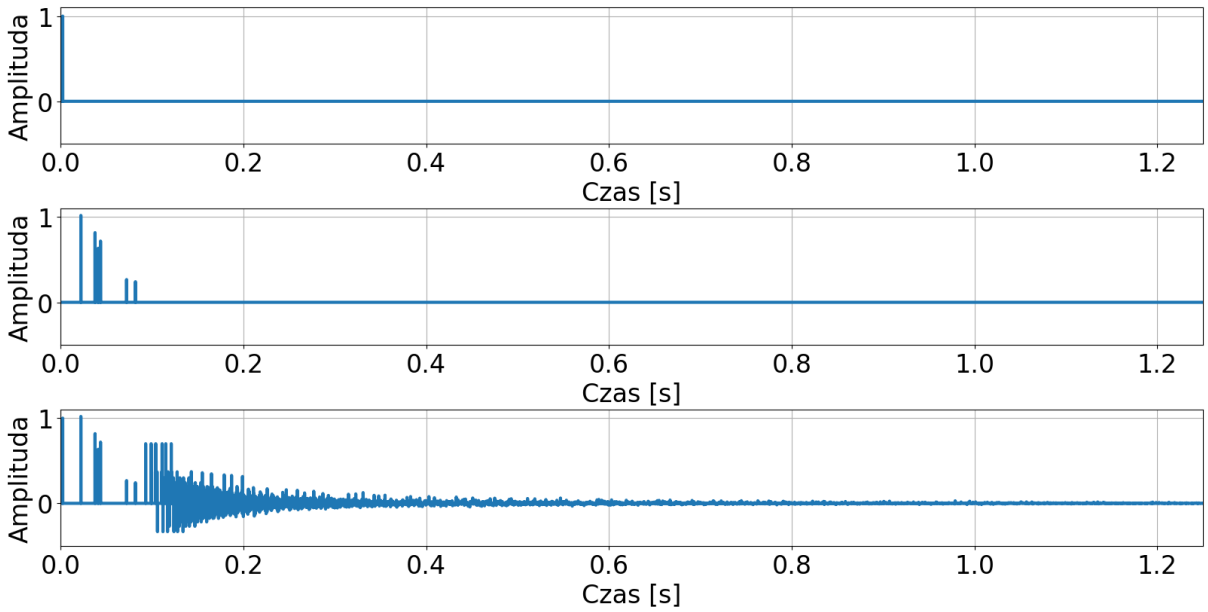
Zmienne `wet1` i `wet2` obliczane są na podstawie uzyskiwanych od użytkownika za pomocą interfejsu graficznego parametrów `wet` i `width` w następujący sposób [33]:

$$wet1 = wet\left(\frac{width}{2} + 0{,}5\right) \tag{35}$$

$$wet2 = wet\left(\frac{1 - width}{2}\right) \tag{36}$$

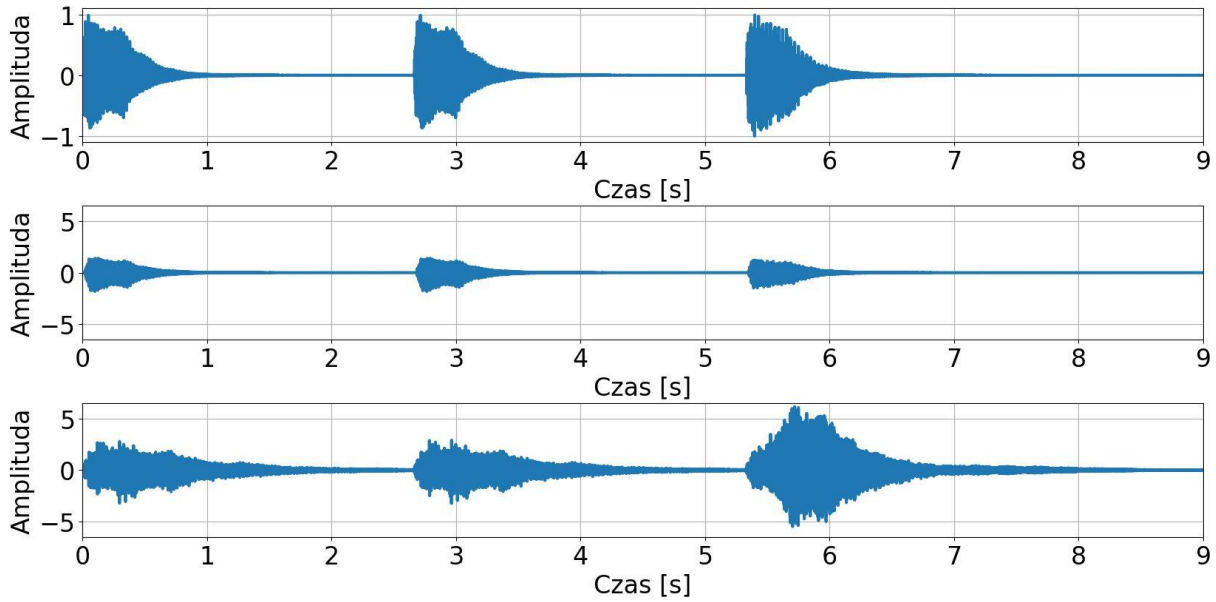
Łatwo zauważyć więc, że `wet1` i `wet2` wyznaczają proporcje, w jakich sumowane są ze sobą wektory wyjściowe struktury filtrów dla obu kanałów. Dopiero taki zabieg doprowadza do uzyskania ostatecznych sygnałów, stanowiących wynik działania algorytmu. Gdy oba te parametry mają tą samą wartość (`width` = 0, `wet1` = `wet2` = 0,5), do lewego i prawego kanału wysyłany jest identyczny sygnał. Gdy różnica wartości tych zmiennych jest największa (`width` = 1, `wet1` = 1, `wet2` = 0) nie występuje żadne sumowanie sygnałów, a separacja pomiędzy kanałami jest wtedy maksymalna [22]. Zastosowanie opisanych metod doprowadziło do uzyskania wiarygodnego odczucia „szerokości" źródła dźwięku i pozwoliło na przetwarzanie plików dwukanałowych bez silnie wpływającej na końcowy efekt percepcyjny konwersji do postaci monofonicznej.

Wynik działania algorytmu w odpowiedzi na wymuszenie w postaci delty Kroneckera przedstawiono na Rys. 21.



Rys. 21 Wynik działania pogłosu opartego o strukturę Moorera. Od góry: prawy kanał sygnału podstawowego, delty Kroneckera; prawy kanał symulacji wczesnych odbić; prawy kanał sygnału wynikowego, wczesnych odbić zsumowanych z ogonem pogłosowym i sygnałem bezpośrednim.

Rys. 22 obrazuje działanie algorytmu Moorera na sygnale dźwiękowym – akordach zagranych na pianinie.



Rys. 22 Wynik działania pogłosu opartego o strukturę Moorera. Od góry: prawy kanał sygnału podstawowego, akordów na pianinie; prawy kanał symulacji wczesnych odbić; prawy kanał sygnału wynikowego, wczesnych odbić zsumowanych z ogonem pogłosowym i sygnałem bezpośrednim.

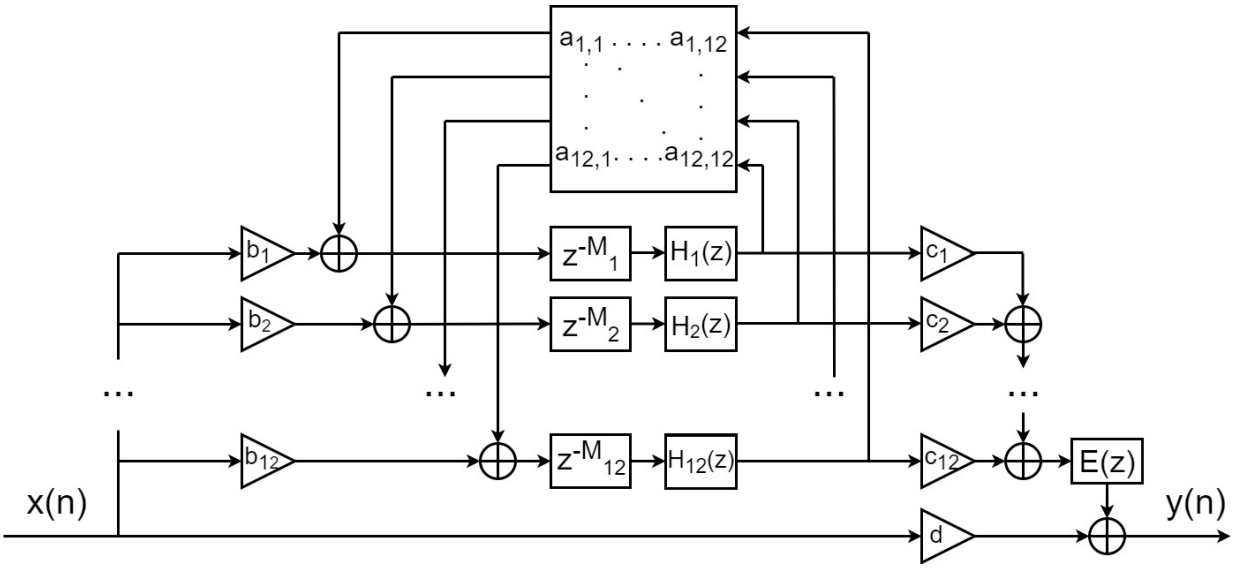
Największą zaletą procesora pogłosowego opartego o strukturę Moorera jest z pewnością możliwość otrzymania efektu finalnego bez wykorzystania żadnych dodatkowych ze-

wnętrznych sygnałów – do realizacji algorytmu konieczne jest jedynie dysponowanie sygnałem wejściowym. Stanowi to wyraźną zaletę tego rozwiązania w porównaniu do omawianego wcześniej pogłosu splotowego. Wykorzystanie filtrów i linii opóźniających przyczyniło się też do znacznego zmniejszenia wymagań obliczeniowych metody. Algorytm Moorera cechuje łatwa i szeroka możliwość modyfikacji. Istnieje bardzo wiele sposobów połączenia używanych filtrów, doboru ich ilości i konkretnej architektury, a każda zmiana doprowadza do otrzymania odmiennych rezultatów brzmieniowych. Z tego powodu, podobne metody symulowania pogłosu (często jednak o dużo większym stopniu skomplikowania) są rozwijane i używane w komercyjnych aplikacjach do dzisiaj. Opisywana struktura pozwala na dużo większą niż w przypadku pogłosu splotowego możliwość edycji parametrów brzmieniowych, oferując kontrolę nad czasem pogłosu, tłumieniem wysokich częstotliwości, a także czasami wzajemnych opóźnień między sygnałem bezpośrednim, etapem wczesnych odbić i późnego pogłosu. Pozwala także na uzyskanie efektu wystarczająco zbliżonego percepcyjnie do rzeczywistych pomieszczeń, głównie w przypadku sygnałów o charakterze innym niż impulsowy.

Wady procesora Moorera zostały po części wymienione wyżej – nie wszystkie sformułowane przez autora problemy udało się rozwiązać. Pogłos generowany w odpowiedzi na krótki, urywany dźwięk jest niejednorodny, o pulsującym charakterze, a niekiedy wyróżnić w nim można pojedyncze odbicia. Taki stan rzeczy jest wynikiem doboru długości linii opóźniających. Skrócenie ich i dobranie wartości zbliżonych do siebie pozwala na pozbycie się tego problemu, jednak jest to równoważne z uzyskaniem krótszego całkowitego czasu efektu. Ostateczna implementacja musi więc stanowić kompromis pomiędzy tymi dwoma elementami. „Metaliczny" charakter wybrzmiewania został w dużej mierze wyeliminowany poprzez dolnoprzepustową filtrację ogona pogłosowego, który sam wykazuje zaś odpowiednio szybki czas narastania. Podczas testów narzędzia stwierdzono, że zbyt duże opóźnienie sygnału wynikowego struktury filtrów od sygnału wejściowego doprowadza do wystąpienia nieprzyjemnego i nienaturalnego efektu pojedynczego, dyskretnego echa przed wybrzmiewaniem pogłosu. Korzystne jest więc zmniejszenie tego opóźnienia, nawet kosztem nałożenia się ogona pogłosowego na ostatnie wczesne odbicia. Struktura Moorera charakteryzuje się wymagającym „strojeniem", czyli doborem parametrów w celu uzyskania realistycznego efektu. Sam autor wspomina, że uzyskane rezultaty są w większości wynikiem długotrwałych testów odsłuchowych i metody prób i błędów, a nie precyzyjnego rozumowania. Oczywistym jest również, że omawiany efekt jako efekt percepcyjny nie oferuje symulacji rewerberacji istniejących przestrzeni, a jedynie słuchowo nierozróżnialny (w zamyśle) ekwiwalent. Nie może być więc używany do pomiarów ani projektowania architektonicznego, sprawdza się zaś w produkcji muzycznej, filmowej i przetwarzaniu mowy. Implementacyjnie jest to metoda bardziej złożona i pracochłonna niż pogłos splotowy, jednak wciąż nieskomplikowana ideowo. Jest również dużo mniej elastyczna.

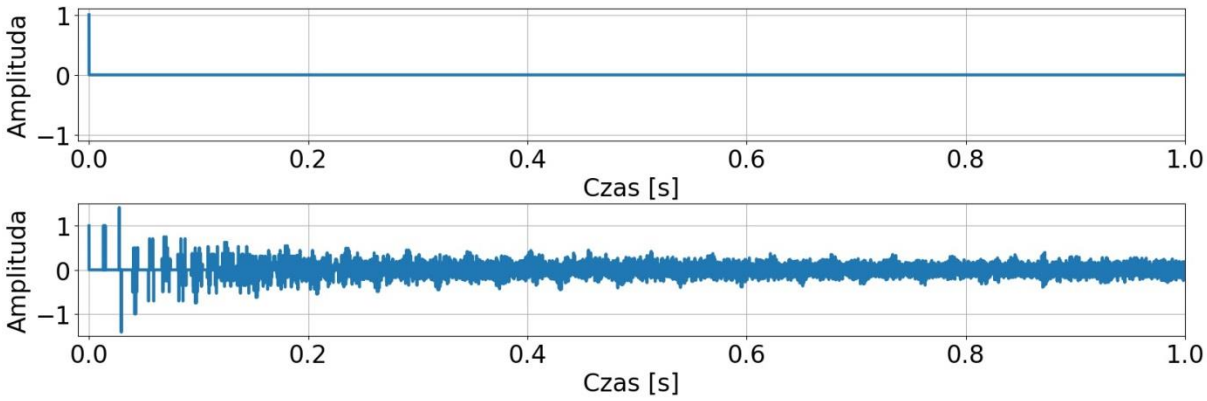
5.3. Testowa implementacja algorytmu opartego o sieci pogłosowe FDN

Trzecią testową implementację stanowił projekt oparty o sieci pogłosowe FDN, który opisany został w szczegółowy i usystematyzowany sposób przez J.-M. Jota w 1991 roku [11]. Struktura ta składa się z linii opóźniających, z których każda podlega wzmocnieniu, filtracji dolnoprzepustowej oraz finalnie sumowaniu w celu osiągnięcia sygnału wyjściowego. Kluczowym elementem jest obecność pętli sprzężenia zwrotnego, wspólnej dla wszystkich linii i zawierającej tzw. macierz rozpraszającą. Schemat blokowy tego prototypu został przedstawiony na Rys. 23.



Rys. 23 Schemat blokowy zaimplementowanego procesora pogłosowego opartego o sieć pogłosową FDN.

Budowę projektu rozpoczęto od implementacji prototypu bezstratnego. Jest to przypadek pozbawiony zaprezentowanych w powyższym schemacie filtrów oraz wzmocnień i w efekcie miał on doprowadzić do generacji niesłabnącego białego szumu w odpowiedzi na pobudzenie sygnałem wejściowym. Oryginalnie, zdecydowano się na wykorzystanie 4 linii opóźniających oraz macierzy przytoczonej w (18), zaprojektowanej przez Stautnera i Puckette'a w ich pionierskiej dla zagadnienia pracy z 1982 r. [21]. Taka prosta konstrukcja okazała się skuteczna w teorii, jednak testy odsłuchowe odsłoniły jej liczne niedoskonałości. Efekt działania algorytmu na tym etapie przedstawia Rys. 24.



Rys. 24 Odpowiedź impulsowa prototypu bezstratnego sieci pogłosowej FDN. Od góry: sygnał wejściowy, delta Kroneckera; sygnał wyjściowy.

Analizując powyższe wykresy, można dojść do wniosku, że ma się do czynienia z bardzo perspektywicznym prototypem: odpowiedź rzeczywiście jest nieskończona (układ znajduje się na granicy stabilności – jest granicznie stabilny), ponadto impulsowe wymuszenie doprowadza do uzyskania zwartego sygnału o charakterze szumu, a jego gęstość narasta w czasie. Początkowo, w czasie pierwszych ok. 100 ms mamy do czynienia z dyskretnymi odbiciami, które następnie płynnie i naturalnie przechodzą w jednolity szum. Jest to bardzo dokładna symulacja fizycznego zjawiska pogłosu, na który składają się etapy wczesnych odbić (o zbliżonym czasie trwania) oraz ogona pogłosowego. Niestety, testy odsłuchowe ujawniły, że brzmieniowo prototyp odbiega od rzeczywistości. Efekt był drżący i niejednorodny, występowały zaburzenia o charakterze okresowym (co można z resztą zaobserwować na przebiegu fali w postaci „pików", występujących z częstotliwością ok. 40 Hz). Ogon pogłosowy nie sprawiał wrażenia „gładkiego", dała się odczuć niewystarczająca gęstość symulowanych odbić. W celu skorygowania tych niedoskonałości wprowadzono dwie modyfikacje: pierwsza z nich dotyczyła liczby linii opóźniających. Za pomocą testów odsłuchowych stwierdzono, że zastosowanie 12 takich linii wystarczająco redukuje drżące brzmienie ogona pogłosowego, w przeciwieństwie do struktur z 6 i 8 gałęziami opóźniającymi. Zmiana ta nie miała znaczącego wpływu na czas obliczeń dokonywanych w ramach algorytmu, zdecydowano się więc na kontynuację pracy z wykorzystaniem takiego układu. Druga ze zmian związana była z doborem macierzy rozpraszającej. Macierz ta powinna zapewniać jak największą gęstość powtórzeń w ogonie pogłosowym – taka sytuacja ma miejsce, gdy każdy jej element jest niezerowy. Zdecydowano się na użycie macierzy Householdera, do obliczenia której wykorzystano macierz permutacji $I_{12\,x\,12}$ przytoczoną przez Gardnera [13]:

$$I_{12\,x\,12} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \tag{37}$$

Otrzymana w wyniku refleksji Householdera macierz $A_{12\ x\ 12}$ posiadała wyłącznie niezerowe wartości – liczbę $^{5}/_{6}$ w miejscu jedynek macierzy permutacji i liczbę $-^{1}/_{6}$ w miejscu zer. Test odsłuchowy tak przygotowanego bezstratnego prototypu wykazał brzmienie odpowiedzi impulsowej dużo bardziej zbliżone do oczekiwanego białego szumu, bez nieprzyjemnych artefaktów i zaburzeń. Zdecydowano, że dalsza implementacja będzie przeprowadzona w oparciu o taki szkielet.

Najważniejszym zagadnieniem na tym etapie pracy był dobór długości linii opóźniających. Przeprowadzono go w oparciu o dwa warunki:

1. Spełnienie wymagań dla długości gałęzi opóźniających wprowadzonych w rozdziale 4.2.3., dotyczących zastosowania liczb wzajemnie pierwszych oraz zgodności z nierównością (21).
2. Osiągnięcie ostatecznego czasu pogłosu do 5 s.

Warunek 2. i nierówność (21) wymuszają:

$$M \geq 0{,}15 t_{60} f_s = 0{,}15 * 5\ s * 44100\ Hz = 33075 \qquad (38)$$

Zastosowane długości linii w próbkach muszą więc być liczbami wzajemnie pierwszymi, których suma wynosi co najmniej 33075. W trakcie przeglądu literaturowego nie napotkano powszechnie stosowanej i gwarantującej percepcyjnie zadowalający rezultat metody ustalania tych wartości. Przeprowadzono więc wielopoziomowe testy odsłuchowe dużej liczby kombinacji w celu „nastrojenia" algorytmu. Zwrócono uwagę przede wszystkim na brak występowania jakichkolwiek powtarzalnych składowych w wybrzmiewaniu pogłosu. Ostatecznie, prototyp oparty został na liniach zgromadzonych w Tab. 4.

Tab. 4 Zastosowane w prototypie sieci pogłosowej FDN długości linii opóźniających w próbkach.

Lp.	1	2	3	4	5	6	7	8	9	10	11	12
m_i	601	1399	1747	2269	2707	3089	3323	3571	3911	4127	4639	4999

Gdzie:

$$M = \sum_{i=1}^{12} m_i = 36382 > 33075 \qquad (39)$$

Kolejnym zagadnieniem było uzyskanie odpowiedniego czasu pogłosu, polegając na konstrukcji zapewniającej nieskończone wybrzmiewanie ogona pogłosowego. Czas ten nie może być stały dla wszystkich częstotliwości zawartych w sygnale wejściowym – dla tonów wysokich powinien być krótszy, przy czym ostateczne proporcje powinny być edytowalne i udostępnione do regulacji użytkownikowi aplikacji. W celu osiągnięcia takiego stanu rzeczy, w każdej z 12 gałęzi sieci pogłosowej wprowadzono połączony szeregowo z linią opóźniającą

filtr dolnoprzepustowy [22]. Transformacja taka prowadzi do przesunięcia biegunów transmitancji układu sieci pogłosowej FDN z okręgu jednostkowego (znajdują się na nim, ponieważ układ jest granicznie stabilny) do jego wnętrza (co spowoduje, że odpowiedź impulsowa układu będzie skończona). Wykonano podstawienie:

$$z^{-1} \leftarrow G(z)z^{-1} \tag{40}$$

Gdzie:
G(z) – element przeprowadzający filtrację dolnoprzepustową jednej próbki.

Celem podstawienia jest uzyskanie osłabienia sygnału o 60 dB dla pulsacji ω po przetworzeniu n_{60} próbek, gdzie:

$$n_{60}(\omega) = \frac{t_{60}(\omega)}{T} \tag{41}$$

Co oznacza, że [22]:

$$\left|G\left(e^{j\omega T}\right)\right|^{n_{60}(\omega)} = 0{,}001 \tag{42}$$

$$H_i(z) = G^{m_i}(z) \tag{43}$$

$$\left|H_i\left(e^{j\omega T}\right)\right|^{\frac{t_{60}(\omega)}{m_i T}} = 0{,}001 \tag{44}$$

$$20\log_{10}\left|H_i\left(e^{j\omega T}\right)\right| = -60\frac{m_i T}{t_{60}(\omega)} \tag{45}$$

Ten wniosek, uzyskany przez Jota, pozwala na użycie dowolnej metody projektowania filtrów cyfrowych w celu uzyskania filtrów $H_i(z)$ o charakterystyce podanej w (45). W związku z faktem, że czas rewerberacji w odniesieniu do częstotliwości zmienia się w bardzo łagodny sposób, po raz kolejny zasadnym wydaje się użycie filtrów o nieskończonej odpowiedzi impulsowej niskiego stopnia. Zaproponowano następujące rozwiązanie:

$$H_i(z) = g_i\frac{1 - p_i}{1 - p_i z^{-1}} \tag{46}$$

Taka struktura filtra zapewnia, że dla każdej stabilnej wartości parametru p_i, parametr g_i będzie określał wzmocnienie filtra dla pulsacji $\omega = 0$. Jak wynika z (45), jego wartość będzie więc równa:

$$g_i = 10^{-\frac{3m_i T}{t_{60}(0)}} \tag{47}$$

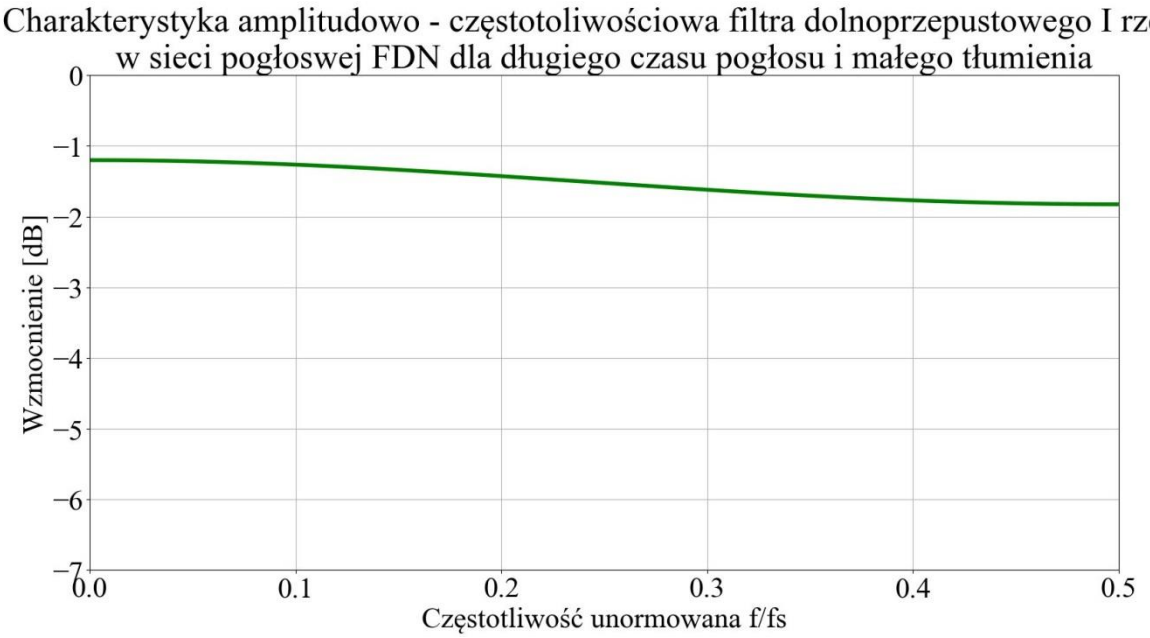
W ten sposób parametr g_i filtrów pozwala na bezpośrednią kontrolę czasu pogłosu, określonego domyślnie dla zerowej pulsacji, czyli najniższej częstotliwości występującej w sygnale. Drugi z parametrów, p_i, musi więc odpowiadać za proporcje tłumienia wysokoczęstotliwościowego. Wyraża się ona za pomocą proporcji czasu pogłosu dla $\omega = 0$ i czasu pogłosu dla częstotliwości równej połowie częstotliwości próbkowania, czyli najwyższej, która może być jednoznacznie odtworzona w cyfrowej postaci sygnału. Parametr ten oznaczono jako α. Wartość p_i wynosi więc [22]:

$$p_i = \frac{\ln(10)}{4} \log_{10} g_i \left(1 - \frac{1}{\alpha^2}\right) \tag{48}$$

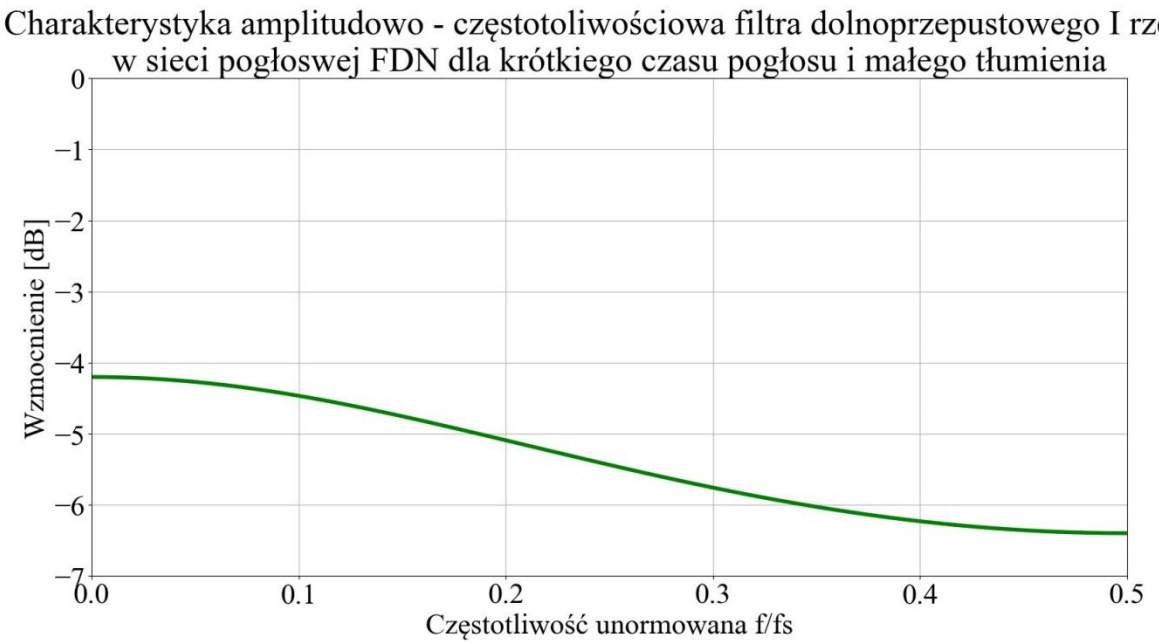
Gdzie:

$$\alpha = \frac{t_{60}\left(\frac{\pi}{T}\right)}{t_{60}(0)} \tag{49}$$

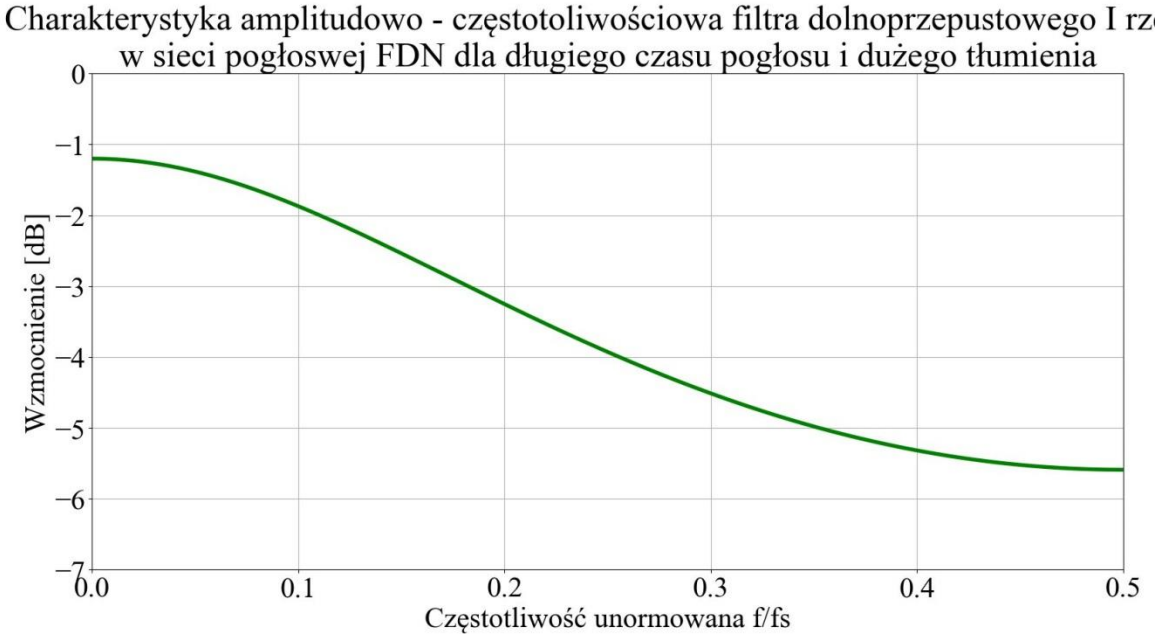
Podobnie jak w przypadku poprzedniego prototypu, filtry tłumiące zostały zaimplementowane z użyciem funkcji `scipy.signal.lfilter()`. Każda linia opóźniająca przetwarzana była z użyciem innej instancji filtra, ponieważ parametr g_i jest zależmy od m_i. Poniżej, na Rys. 25 – 27 zaprezentowano charakterystyki amplitudowo – częstotliwościowe filtra dla linii opóźniającej m_6 = 3089. Dla ułatwienia analizy, wszystkie charakterystyki przedstawiono w zakresie wzmocnienia od 0 dB do -7 dB.



Rys. 25 Charakterystyka filtra tłumiącego wysokie częstotliwości dla m_6 = 3089, $t_{60}(0)$ = 3,5 s, α = 0,7.



Rys. 26 Charakterystyka filtra tłumiącego wysokie częstotliwości dla m_6 = 3089, $t_{60}(0)$ = 1 s, α = 0,7.



Rys. 27 Charakterystyka filtra tłumiącego wysokie częstotliwości dla m_6 = 3089, $t_{60}(0)$ = 3,5 s, α = 0,35.

Jak łatwo zauważyć, charakterystyka działania filtru jest zgodnie z założeniami oparta o parametr czasu pogłosu i stosunku α. Modyfikacja $t_{60}(0)$ pozwala na kontrolę całościowego wzmocnienia we wszystkich pasmach (co doprowadza do szybszego zaniku dźwięku o każdej częstotliwości), natomiast modyfikacja α pozwala na ustalenie kształtu charakterystyki, w szczególności tempa jej opadania dla wysokich tonów.

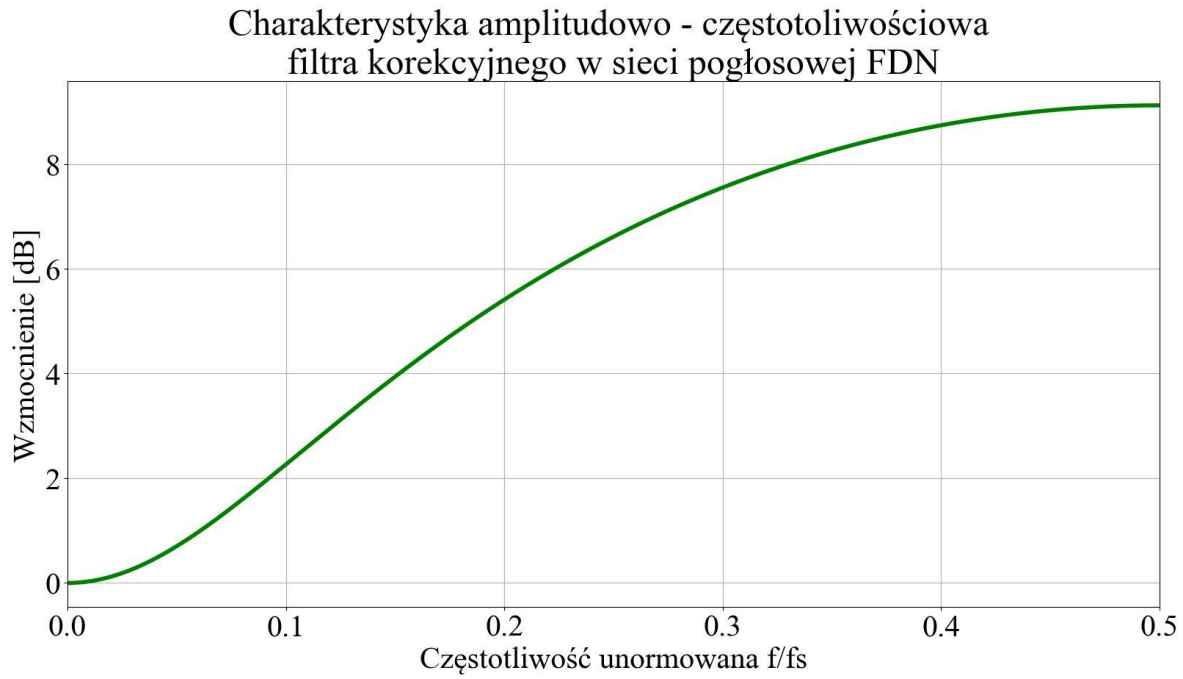
Jot zauważył także, że każdy biegun układu sieci pogłosowej FDN ma swój udział w kształtowaniu jej odpowiedzi impulsowej poprzez wkład energii w sumaryczną energię sygnału wyjściowego. W strukturze stosującej opisane wyżej filtry tłumiące, skrócenie czasu pogłosu prowadzi do proporcjonalnego spadku tego wkładu (przy założeniu niezmienności innych parametrów) [11]. Efekt ten powinien zostać skompensowany i jako wynik tego wprowadzony został filtr korekcyjny *E(z)* (ang. *tonal correction filter*). Jego realizacja oparta jest na filtrze o skończonej odpowiedzi impulsowej pierwszego rzędu [22]:

$$E(z) = \frac{1 - \beta z^{-1}}{1 - \beta} \tag{50}$$

Gdzie:

$$\beta = \frac{1 - \alpha}{1 + \alpha} \tag{51}$$

Filtr taki jest połączony szeregowo ze strukturą sieci FDN i prezentuje górnoprzepustowy charakter. Jego charakterystykę amplitudowo – częstotliwościową przedstawiono na Rys. 28.



Rys. 28 Charakterystyka zastosowanego filtra korekcyjnego dla α = 0.35.

Ostatnim etapem realizacji sieci pogłosowej FDN był dobór współczynników wzmocnień b_i, c_i i d. Współczynniki b_i odpowiadają za kontrolę wzmocnienia wejściowego, czyli modyfikują amplitudę sygnału w każdej z gałęzi opóźniających. Odpowiednie dobranie tych współczynników umożliwia uzyskanie sygnału wyjściowego o amplitudzie nieprzekraczającej zakresu numerycznego ostatecznego typu danych (uniknięcie przesterowania). Współczynniki c_i odpowiadają za wzmocnienie wyjściowe sieci pogłosowej. Można wykorzystać je

w wygodny sposób do wprowadzenia separacji kanałów, analogicznie jak w prototypie procesora Moorera. Jot zauważył, że system oparty o macierz rozpraszającą Householdera generuje periodyczne kliknięcia w odpowiedzi impulsowej. Można im zapobiec wprowadzając odwrócenie znaku co drugiej wartości c_i, czyli używając do ustalenia wzmocnienia wyjściowego następującej macierzy C:

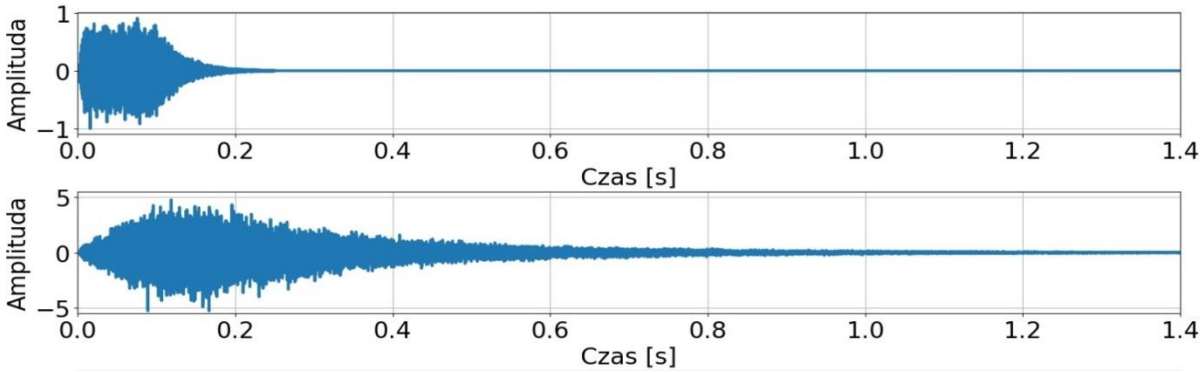
$$C = c\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ \vdots \\ -1 \end{bmatrix} \tag{52}$$

Dysponując takim warunkiem, sposobem na dekorelację kanałów w sygnale wynikowym jest użycie macierzy C o wymiarach 12 x 2, w której każda kolumna odpowiada jednemu z kanałów wyjściowych. Różnica między pierwszą a drugą kolumną umożliwia uzyskanie nieskorelowanych sygnałów w obu kanałach. W celu zastosowania tej metody, w prototypie użyto następującej macierzy C:

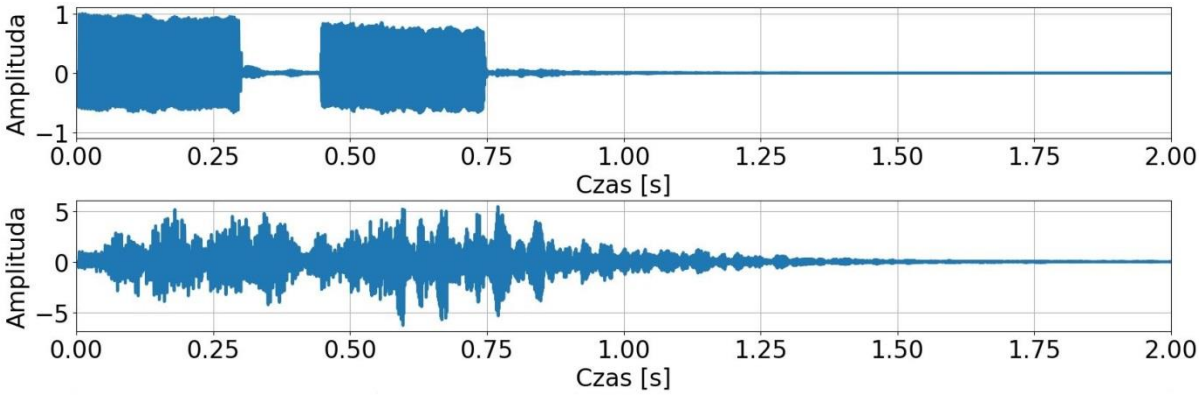
$$C = c\begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & -1 \\ \vdots & \vdots \\ 1 & -1 \\ -1 & -1 \end{bmatrix} \tag{53}$$

Współczynnik d odpowiada za regulację proporcji sygnału bezpośredniego w ostatecznym wyniku algorytmu. Jego wartość wraz z wybraną wartością c wpływają na ostateczną relację pomiędzy pogłosem a dźwiękiem bez efektu, co jest podstawowym parametrem edytowalnym w każdym procesorze pogłosowym.

Rys. 29 i 30 prezentują efekt działania prototypu sieci pogłosowej FDN.



Rys. 29 Wynik działania pogłosu opartego o sieć pogłosową FDN. Od góry: prawy kanał sygnału podstawowego, krótkiego fragmentu białego szumu; prawy kanał sygnału wynikowego, uzyskanego dla $t_{60}(0)$ = 2,5 s, α = 0,5 i jednostkowych współczynników wzmocnień.



Rys. 30 Wynik działania pogłosu opartego o sieć pogłosową FDN. Od góry: prawy kanał sygnału podstawowego, dwóch wysokich dźwięków na syntezatorze; prawy kanał sygnału wynikowego, uzyskanego dla $t_{60}(0)$ = 1,5 s, α = 0,8 i jednostkowych współczynników wzmocnień.

Sieci pogłosowe FDN to rozwiązanie najbardziej zaawansowane ideowo z testowanych metod symulacji rewerberacji, Ich powstanie i rozwój był wynikiem wielu lat eksperymentów i propozycji ulepszeń, opracowywanych przez inżynierów na całym świecie. Dzięki temu są najbardziej kompleksową i elastyczną metodą generacji sztucznego pogłosu. W testach odsłuchowych pogłos oparty o sieci FDN okazał się najbardziej realistyczny i brzmieniowo zbliżony do komercyjnie dostępnych efektów, np. wtyczek VST. Mimo, iż otrzymanie optymalnych ustawień wiązało się z przeprowadzeniem procesu „strojenia" algorytmu, nawet przy niedoskonałych współczynnikach wynik jego działania był bardzo zadowalający. Sieci pogłosowe FDN cechuje przejrzysta możliwość parametryzacji i szeroki zakres potencjalnych modyfikacji. Algorytm jest skalowalny, jego strukturę można uprościć bądź rozszerzyć w intuicyjny sposób, nietrudno także wprowadzić do niej dodatkowe elementy przetwarzające sygnał. Działanie tej metody jest oparte o drobiazgowo przeanalizowany aparat matematyczny, w jej opisie teoretycznym mniej miejsca pozostawione jest więc przypadkowi i działaniom opartym o próby i błędy. Istnieje możliwość redukcji złożoności obliczeniowej algorytmu przy wykorzystaniu i umiejętnej implementacji macierzy rozpraszającej, co sprzyja rozwojowi sieci pogłosowych w aplikacjach działających w czasie rzeczywistym.

Do wad sieci FDN zaliczyć można dość znaczną „objętość" algorytmu (metoda operuje na wielokrotnych liniach opóźniających i dużych macierzach, co wymusza na programiście umiejętność zachowania porządku w kodzie) oraz uzależnienie brzmienia efektu od drobiazgowego doboru parametrów, na czele z długościami linii opóźniających. Ten drugi mankament jest jednak powszechny w zagadnieniach symulacji pogłosowych i póki co nie opracowano sposobu na „zautomatyzowanie" tego procesu, można go traktować więc jako element sztuki w typowo inżynierskim zadaniu. Struktura sieci pogłosowej FDN może okazać się niestabilna, więc należy zwrócić uwagę na niedopuszczenie do takiego stanu rzeczy na etapie projektowania. Finalnie, proces poszukiwania optymalnego brzmienia efektu wymaga świadomego doboru kilku elementów (macierz rozpraszająca, liczba i długość linii opóźniających,

struktura filtrów, współczynniki wzmocnienia), co wymusza na osobie przygotowującej algorytm umiejętność poprawnego powiązania parametrów brzmieniowych z odpowiedzialnymi za nie segmentami kodu.

5.4. Wnioski z analizy programów prototypowych i motywacja dalszego postępowania

Trzy implementacje prototypowe przygotowano w celu oceny algorytmów i wyłonienia metody, która wykorzystana zostanie w ostatecznej wersji aplikacji. Realizacja prototypów i przygotowanie skryptów do operowania na rzeczywistych plikach dźwiękowych pozwoliły na przeprowadzenie dogłębnej analizy sposobu działania istniejących rozwiązań i w połączeniu z szeroko zakrojonym studium teoretycznym umożliwiło wiarygodną ocenę ich użyteczności w kontekście planowanego programu. Przy ocenie algorytmów brano pod uwagę takie czynniki jak:

- Brzmienie. (Czy symulacja pogłosu jest realistyczna? Czy nie występują artefakty lub zakłócenia pozwalające jednoznacznie zakwalifikować efekt jako nierealistyczny?)
- Możliwość parametryzacji. (W jaki sposób doprowadzić można do zmiany czasu pogłosu, tłumienia wysokich częstotliwości, proporcji sygnału bezpośredniego do przetworzonego? Czy da się to zrobić wewnątrz algorytmu?)
- Łatwość implementacji. (Czy algorytm wykazuje się dużym stopniem skomplikowania? Czy mnogość parametrów nie utrudnia utrzymania kontroli nad programem?)
- Wykorzystanie w profesjonalnych projektach. (Czy metoda znajduje praktyczne zastosowanie, czy jest raczej eksperymentalna? Czy użyta technologia spełnia standardy narzucone przez dzisiejszy rynek oprogramowania dźwiękowego?)

Analiza tych zagadnień dość szybko doprowadziła do wniosku, że pogłos oparty o filtry grzebieniowe i wszechprzepustowe w formie zaproponowanej przez Moorera nie będzie odpowiednim wyborem. Mimo, iż pozwala on na uzyskanie relatywnie autentycznie brzmiącej symulacji i istnieje w nim możliwość edycji parametrów brzmieniowych, stanowi rozwiązanie dość trywialne w świetle rozwoju dziedziny przez ostatnie 40 lat. Zapewnienie szerokiej stosowalności tego algorytmu wymaga w dzisiejszych czasach dość rozległych modyfikacji, które nie wnoszą wiele nowości do teoretycznych rozważań, są natomiast efektem słuchowego „dostrajania". W formie zaimplementowanej w toku niniejszej pracy projekt Moorera ciągle charakteryzuje się występowaniem nierozwiązanych problemów brzmieniowych i strukturalnych, których obecność w finalnej realizacji znacząco wpłynęłoby na jakość aplikacji jako całości.

Pogłos splotowy należy do najczęściej wykorzystywanych obecnie w komercyjnych aplikacjach algorytmów. Zapewnia ogromne możliwości kreacji brzmienia, zdecydowanie największe wśród badanych metod. Jest nieskomplikowany ideowo, a wykorzystanie wydajnych metod implementacji pozwala na jego pracę nawet w czasie rzeczywistym. Nie ma jed-

nak możliwości prostej edycji jego parametrów brzmieniowych, a w celu otrzymania sygnału wynikowego konieczne jest dysponowanie sygnałem zawierającym pożądaną odpowiedź impulsową.

Ostatecznie, za najodpowiedniejszy wybór uznano wykorzystanie sieci pogłosowych FDN. Metoda ta jest szeroko rozpowszechniona na rynku procesorów efektowych, ponieważ pozwala na uzyskanie bardzo dobrze, realistycznie brzmiących rezultatów. Jest ona nadal obiektem badań, wprowadzających coraz nowsze rozwiązania i ulepszenia i doprowadzających do jej prężnego rozwoju. Algorytm jest łatwy w parametryzacji, oparty na nieskomplikowanych elementach składowych i pozwala na mocną ingerencję użytkownika w finalny rezultat brzmieniowy. Działa także bardzo wydajnie i pozostawia szerokie pole do udoskonaleń w dziedzinie złożoności obliczeniowej. Wyzwania, które niesie za sobą użycie sieci pogłosowych FDN w finalnym programie to m. in. powiązanie parametrów używanych przez algorytm z parametrami udostępnionymi do regulacji użytkownikowi poprzez interfejs graficzny oraz odpowiedni dobór kontenerów wykorzystywanych do implementacji linii opóźniających („przetłumaczenie" algorytmu z języka Python na C++). W wyniku opracowania trafnych rozwiązań tych problemów finalny rezultat pracy powinien prezentować bardzo wysoką jakość brzmieniową i oferować możliwość intuicyjnej kontroli.

6. Docelowa implementacja procesora pogłosowego opartego o sieci pogłosowe FDN

Ostateczna implementacja programu realizującego efekt pogłosu została wykonana w oparciu o prototyp, opisany w poprzednim rozdziale. Skrypt testowy realizował jedynie wybrany algorytm – w czasie jednego uruchomienia dokonywano wczytania pliku, przetwarzania sygnału stereo w oparciu o zadane parametry, zapisu sygnału wyjściowego do pliku i wizualizacji wyników w celu ułatwienia analizy. Założeniem towarzyszącym docelowej realizacji było stworzenie programu, który umożliwi rozdzielenie tych zadań i przekaże kontrolę nad procesem ich wykonywania użytkownikowi poprzez czytelny interfejs graficzny. Aby osiągnąć ten cel, program musiał prezentować budowę modułową, w której każdy moduł odpowiedzialny jest za realizację innego zadania. Moduły musiały posiadać możliwość wymiany danych między sobą. Aby umożliwić zrealizowanie całego procesu stopniowego doboru odpowiednio brzmiącego efektu pogłosu do sygnału wejściowego, jednocześnie nie wymuszając zapisu każdego sygnału do osobnego pliku WAV, ustalono, że program musi zawierać następujące elementy (z wyłączeniem części odpowiedzialnej za sam efekt dźwiękowy):

- Moduł wczytujący plik, pozwalający na wybór pliku wejściowego poprzez okno dialogowe,
- Moduł zapisu danych do pliku, pozwalający na nadpisanie lub utworzenie nowego pliku WAV poprzez okno dialogowe,
- Moduł pozwalający na odtworzenie dźwięku bez zapisywania go do pliku w celu oceny wyniku działania algorytmu i stwierdzenia poprawności wczytania pliku.
- Moduł umożliwiający wizualizację kształtu fali obecnie rozpatrywanego sygnału, ułatwiający dobór wzmocnień wejściowych i wyjściowych i kontrolę nad utrzymaniem amplitudy sygnału w odpowiednim zakresie.

Program o takiej budowie umożliwi wygodną i intuicyjną pracę oraz nie będzie generował plików zawierających niesatysfakcjonujące użytkownika rezultaty. Oprócz opisanych modułów, interfejs użytkownika musi zawierać część odpowiedzialną za kontrolę nad parametrami brzmieniowymi pogłosu pod postacią wirtualnych potencjometrów.

Implementacja programu przeprowadzona została z użyciem platformy programistycznej JUCE 5.4. i środowiska programistycznego Microsoft Visual Studio 2019. JUCE jest szkieletem aplikacyjnym dla języka C++, który oferuje rozszerzony zakres bibliotek z dziedzin przetwarzania dźwięku i budowy interfejsów graficznych. Dzięki temu zyskał on dużą popularność wśród autorów aplikacji operujących na sygnałach audio, takich jak wtyczki VST, odtwarzacze multimediów czy wirtualne symulacje instrumentów. Jednym z głównych założeń platformy jest umożliwienie kompilacji i działania tego samego kodu źródłowego na wszystkich najpopularniejszych systemach operacyjnych: Windows, Linux i macOS. JUCE oferuje na swojej stronie internetowej bardzo rozległą dokumentację, która ułatwia posługiwanie się dostępnymi funkcjonalnościami, a także szeroki wachlarz samouczków, opisujących działanie

zaprezentowanych kodów realizujących często spotykane zadania [34]. Jednym z elementów platformy jest Projucer, czyli uproszczone środowisko programistyczne stworzone do pracy z projektami JUCE, umożliwiające zarządzanie strukturą projektu, selekcję wykorzystywanych modułów oraz eksport kodu do zewnętrznego edytora. Podczas tworzenia nowego projektu, Projucer pozwala na wybór typu planowanej aplikacji. Na podstawie decyzji użytkownika automatycznie generowany jest podstawowy kod, implementujący bazowe funkcjonalności związane z wyświetlaniem okna interfejsu graficznego i (w zależności od typu aplikacji) komunikacji karty dźwiękowej z programem. Projucer obsługuje w ten sposób m. in. aplikacje z interfejsem graficznym (ang. *GUI Application*), aplikacje audio (ang. *Audio Application*) czy wtyczki (ang. *Audio Plug-In*). Sposób rozszerzenia takich szkieletów o autorskie funkcjonalności opisany jest szeroko w dokumentacji platformy.

Panel sterowania oraz cała funkcjonalność wewnętrzna aplikacji została wykonana z wykorzystaniem zasobów bibliotek JUCE i opierała się na wbudowanej strukturze aplikacji audio. Program taki nie wymaga działania wewnątrz innego programu będącego gospodarzem, co odróżnia go od wtyczki. Niemożliwe jest użycie go w programie do obróbki dźwięku, a jedynie samodzielnie, niezależnie od reszty oprogramowania. W budowie aplikacji wykorzystano bazowe funkcje zaimplementowane przez platformę.

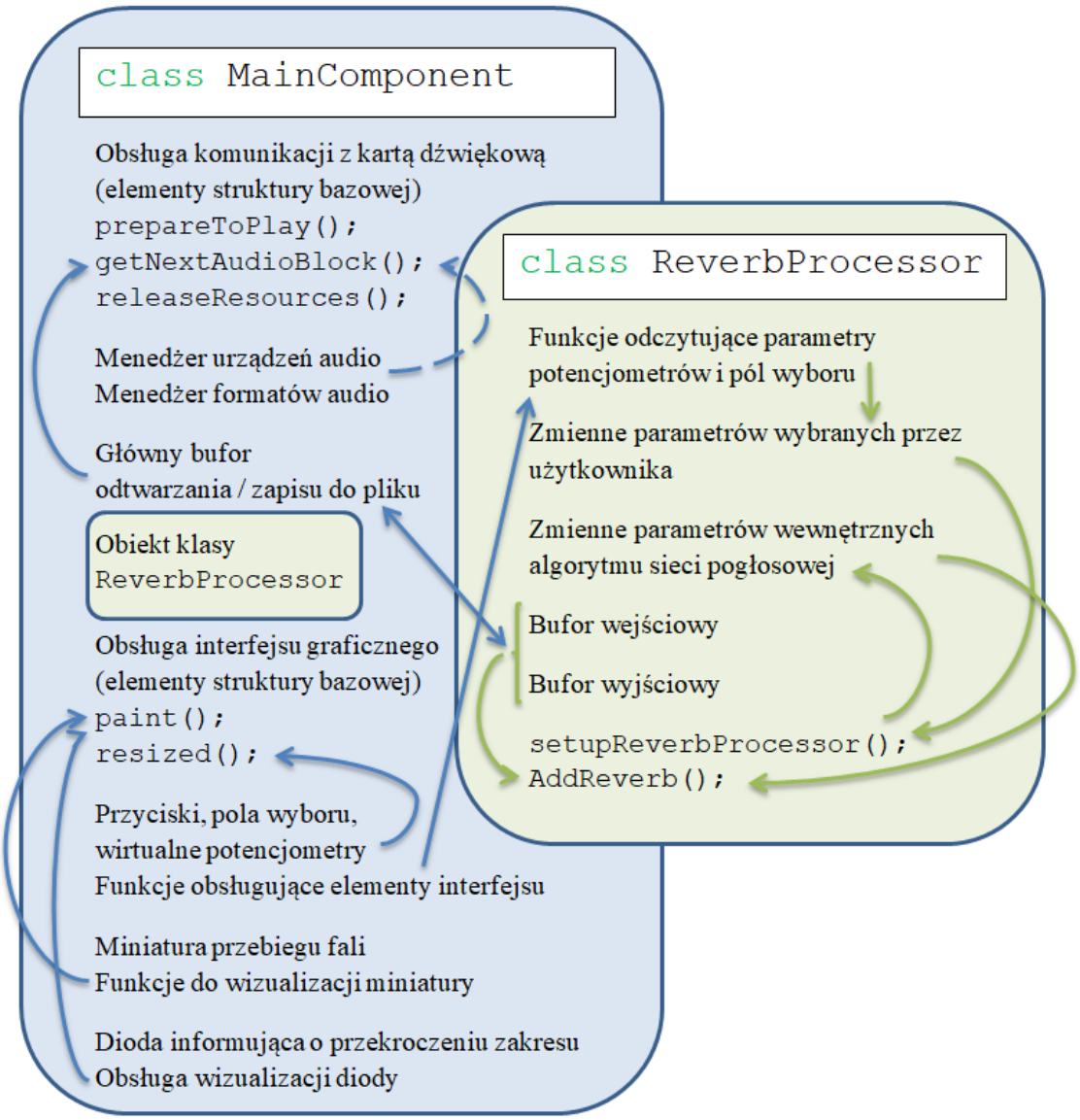
6.1. Całościowa struktura aplikacji w oparciu o paradygmat obiektowy

Na Rys. 31 przedstawiono schemat klas wykorzystanych do budowy programu. Działanie aplikacji i wymiana informacji wewnątrz niej opiera się na klasie `MainComponent`, która tworzona jest automatycznie wraz z utworzeniem projektu opartym na szkielecie aplikacji audio. Klasa ta zawiera oryginalnie pięć funkcji (z wyłączeniem konstruktora domyślnego i destruktora): trzy funkcje odpowiadające za komunikację programu z kartą dźwiękową urządzenia, na którym program działa oraz dwie funkcje obsługujące działanie interfejsu graficznego. Pierwsza z tych grup składa się z funkcji odziedziczonych przez `MainComponent` z klasy macierzystej `AudioAppComponent`. Komunikacja programu z kartą dźwiękową (lub inną docelową lokalizacją strumienia audio) jest oparta o wywołania zwrotne (ang. *callback*). Wywołanie takie realizowane jest w momencie nadejścia zdarzenia, informującego program o konieczności przekazania bloku danych dźwiękowych do bufora wyjściowego. Zadaniem programisty jest w tym wypadku zapewnienie przekazania poprawnych danych, ponieważ w innym wypadku dojść może do wystąpienia artefaktów w odtwarzanym dźwięku. Za realizację opisanego mechanizmu odpowiedzialne są następujące funkcje [35]:

- `prepareToPlay()`- funkcja wywoływana przed pierwszym wywołaniem zwrotnym, mająca na celu wprowadzenie mechanizmu w stan przygotowania do obsługi komunikacji. Oryginalnie przyjmuje dwa argumenty, specyfikujące częstotliwość próbkowania i rozmiar bloku danych dźwiękowych, który będzie wymagany do prze-

kazania. Wewnątrz niej należy umieścić wszystkie dane, których źródło potrzebuje do niezakłóconego przekazywania bloków próbek.

- `getNextAudioBlock()` - funkcja, która wykonywana będzie za każdym razem, gdy odebrany zostanie komunikat o konieczności przekazania nowego bloku danych do karty dźwiękowej. Jej argumentem jest `bufferToFill`, czyli pusty bufor, który należy zapełnić wewnątrz ciała funkcji. Ta funkcja zwrotna jest częścią wątku audio o bardzo wysokim priorytecie.
- `releaseResources()` - funkcja wykonywana po zakończeniu odbierania wywołań zwrotnych, wewnątrz której w zamyśle mają zostać zwolnione wszelkie zasoby używane podczas komunikacji z kartą dźwiękową.



Rys. 31 Schemat zależności klas w zrealizowanym programie.

Do funkcji niskiego poziomu obsługujących interfejs graficzny należą zaś [36]:

- `resized()` - wywołanie tej funkcji następuje, gdy zmieniony jest rozmiar głównego okna aplikacji. W jej wnętrzu należy umieścić kod specyfikujący rozmieszczenie elementów wewnątrz interfejsu oraz ich wzajemne zależności.
- `paint()`- funkcja realizująca wyświetlanie elementów na ekranie. Za jej pomocą można obsługiwać wszystkie niestałe elementy interfejsu. W czasie działania programu można posłużyć się funkcją `repaint()`, która odświeża aktualnie wyświetlane dane.

Składowymi klasy `MainComponent` są wszystkie elementy odpowiedzialne za wczytywanie plików dźwiękowych, zapisywanie ich do pliku, odtwarzanie oraz wizualizację danych na interfejsie graficznym. Ich szczegółowy opis zamieszczony jest w podrozdziale 6.2. Składową opisywanej klasy głównej jest także obiekt klasy `ReverbProcessor` – odpowiada ona za realizację algorytmu sieci pogłosowej FDN w sposób analogiczny do testowej implementacji w języku Python. Obiekt ten za pomocą funkcji ustawiających wartości (ang. *setter*) odczytuje obecnie wybrane przez użytkownika parametry efektu, dokonuje na ich podstawie kalkulacji współczynników koniecznych do wykonania algorytmu, a następnie stosuje go na próbkach bufora wejściowego (odpowiednio przygotowanej do przetwarzania kopii głównego bufora klasy `MainComponent`). W ten sposób uzyskiwany jest bufor wyjściowy, który przekazywany jest z powrotem do głównej klasy. Szczegółowy opis działania tego mechanizmu zamieszczony jest w podrozdziale 6.3.

6.2. Część interfejsowa – operacje wejścia / wyjścia i komunikacja z użytkownikiem

Obsługa aplikacji możliwa jest poprzez graficzny interfejs użytkownika. Prezentujący go zrzut ekranu przedstawiono na Rys. 32.

Rys. 32 Interfejs graficzny aplikacji.

Podczas projektowania interfejsu zwrócono szczególną uwagę na czytelność i zachowanie logicznego układu funkcji (ułożenie „od lewej do prawej", w którym operacje początkowe są umieszczone bliżej lewej strony, a operacje wyjściowe bliżej prawej). W celu zachowania konwencji, wszystkie etykiety interfejsu zamieszczono w języku angielskim. Działanie programu zainicjalizowane musi być otwarciem pliku – po naciśnięciu przycisku *Open a File…* (pol. Otwórz plik…) otwierane jest okno dialogowe, pozwalające na przeglądanie katalogów i wybór plików z rozszerzeniem .wav. Potwierdzenie wyboru powoduje wczytanie zawartości wybranego pliku do bufora audio, który jest składową klasy `MainComponent`. Wszystkie bufory realizowane są za pomocą specjalnie przygotowanej w tym celu klasy JUCE, `AudioBuffer`, która oferuje zoptymalizowane metody zapisu, odczytu i modyfikacji przechowywanych w niej danych. Wczytanie danych powoduje aktywację przycisków *Play* (pol. Odtwarzaj), *Add Reverb* (pol. Dodaj pogłos) oraz *Save To File* (pol. Zapisz do pliku), wybór domyślnej opcji rozdzielczości bitowej przy zapisie, która odpowiada rozdzielczości bitowej wczytanego pliku oraz wyświetlenie w przygotowanym oknie miniatury przebiegu fali obu kanałów. Wczytanie pliku realizowane jest za pomocą obiektu klasy `AudioFormatReader`, który tworzony jest przez będący składową `MainComponent` obiekt klasy `AudioFormatManager`. `AudioFormatReader` pozwala na uzyskanie dostępu do danych wybranego pliku dźwiękowego, takich jak jego długość, liczba kanałów, rozdzielczość bitowa oraz częstotliwość próbkowania, a co najważniejsze umożliwia przepisanie próbek do głównego bufora za pomocą metody `read()`. Program pozwala na pracę z nieskompresowanymi plikami monofonicznymi lub stereofonicznymi, o dowolnej rozdzielczości bitowej i częstotliwości próbkowania 44,1 kHz. Miniatura jest przygotowywana i wyświetlana przy pomocy oferowanej przez platformę klasy `AudioThumbnail`. Konstruktor obiektu tej klasy przyjmuje jako argument instancję klasy `AudioFormatManager`, dzięki której pobiera dane o kształcie fali z wybranego pliku, a także liczbę całkowitą specyfikującą liczbę próbek, które ulegają uśrednieniu w celu uzyskania wartości jednej próbki na miniaturze. W implementacji użyto liczby 32, która pozwala na dość szczegółową wizualizację. Miniatura jest odświeżana w dwóch przypadkach – wczytania nowego pliku lub dodania efektu. W tym drugim przypadku jest najłatwiejszym sposobem na ocenę, czy finalny sygnał nie jest zbyt słaby (zbyt mała amplituda) lub zbyt silny (komunikowane przez czerwony kolor diody *Clip* (pol. Przesterowanie)).

Wciśnięcie przycisku *Play* rozpoczyna odtwarzanie zawartości bufora klasy `MainComponent`. Reagowanie programu na naciśnięcie przycisku jest również obsługiwane za pomocą mechanizmu wywołań zwrotnych. Każdy obiekt klasy `Button` posiada składnik `OnClick`, który zawierać powinien instrukcje realizowane w przypadku wystąpienia zdarzenia – wciśnięcia przycisku. Obsługę wywołania zwrotnego można przeprowadzić poprzez przypisanie do `OnClick` wyrażenia lambda, w ciele którego realizowana jest funkcja związana z przeznaczeniem przycisku. Kod ten przedstawia się następująco:

```
playButton.onClick = [this] { playButtonClicked(); };
```

Linijka ta znajduje się w konstruktorze klasy `MainComponent`. Wyrażenie lambda przechwytuje tworzoną instancję klasy, wewnątrz której znajduje się wywoływana funkcja `playButtonClicked()`. Do każdego przycisku interfejsu przypisana jest osobna funkcja.

Odtwarzanie zawartości bufora oparte jest o prostą maszynę stanów. Naciśnięcie przycisku *Play* inicjuje stan odtwarzania (*Playing*). Jako, że wywołania zwrotne pochodzące od karty dźwiękowej wysyłane są nieustannie, możliwość zapełnienia ich danymi powinna być zapewniona jedynie w tym stanie. W tym celu wprowadzono zmienną logiczną `stopFlag`, której wartość 0 umożliwia odtwarzanie (spełniony jest warunek wykonania kodu wewnątrz `getNextAudioBlock()`). Flaga ta ustawiana jest na 0 wewnątrz funkcji `playButtonClicked()`. W takiej sytuacji realizowana jest pętla, przepisująca dane z bufora głównego do bufora wyjściowego, odczytywanego przez kartę dźwiękową. W każdym obiegu pętli przepisywany jest domyślnie blok określonej wielkości, ustalonej w `prepareToPlay()`. Nie ma jednak gwarancji, że będzie to za każdym razem taki sam rozmiar. Kod musi być na tyle elastyczny, aby obsłużyć odmienne przypadki [35]. Przepisanie danych odbywa się za pomocą metody klasy `AudioBuffer`, `copyFrom()`. Naciśnięcie przycisku *Stop* (pol. Zatrzymanie) zmienia stan na stan zatrzymania (*Stopped*) i wartość flagi na 1. Odtwarzanie zostaje wtedy wtrzymane. Warto nadmienić, że przekazywanie wartości zmiennej `stopFlag` odbywa się pomiędzy tzw. wątkiem informacyjnym (o niskim priorytecie) i wątkiem audio (o wysokim priorytecie). Generalnie, takie działanie jest ryzykowne i może prowadzić do wystąpienia zakłóceń w odtwarzanym dźwięku. Aby zapewnić bezpieczeństwo takiej operacji, wykorzystano klasę JUCE `Atomic`, która bazuje na szablonie biblioteki standardowej C++, `std::atomic`. Typ ten stworzony został specjalnie do obsługi zmiennych, których wartość przekazywana jest pomiędzy wątkami i zapewnia, że rezultat takich operacji będzie zdefiniowany [37]. Ponadto, odczytywanie wartości flagi dokonywane jest w wywołaniach zwrotnych realizowanych przez obiekt klasy `Timer`. Klasa ta pozwala na wykonywanie określonej akcji w regularnych odstępach czasu. Jeżeli w którymś wywołaniu, wykonywanym co 10 ms, `Timer` zarejestruje zmianę wartości `stopFlag` na 1 (dzieje się tak w przypadku zakończenia odtwarzania, tzn. przekazania wszystkich bloków w buforze do bufora wyjściowego), stan zmieniany jest na stan zatrzymania. W ten sposób w odtwarzanym dźwięku nie występują artefakty w postaci nieprzyjemnych trzasków (występowały zaś przed implementacją z wykorzystaniem `Atomic`).

Zapis do pliku wykonywany jest poprzez naciśnięcie przycisku *Save To File…*Podobnie jak w przypadku otwierania pliku, zapis wiąże się z wyświetleniem okna dialogowego, pozwalającego na wybranie lokalizacji i nazwy docelowego pliku WAV. Również analogicznie do odczytu, zadanie wykonywane jest poprzez użycie obiektu klasy `AudioFormatWriter`. Konstruktor tego obiektu wymaga specyfikacji liczby kanałów, częstotliwości próbkowania

oraz rozdzielczości bitowej. Ten ostatni parametr ustalany jest na podstawie wyboru użytkownika, umożliwionego poprzez pola wyboru umieszczone nad przyciskiem. Dzięki tej funkcji, aplikacja może działać jako prosty program do konwersji rozdzielczości bitowej plików WAV. Zapis prowadzony jest z bufora głównego w klasie `MainComponent` za pomocą metody `writeFromAudioSampleBuffer()`.

Oprócz opisanych elementów, interfejs użytkownika zawiera elementy wykorzystywane w specyfikacji parametrów efektu pogłosu oraz przycisk wywołujący algorytm przetwarzania sygnałów. Wirtualne potencjometry są obiektami klasy `Slider`. Ich stan nie jest monitorowany na bieżąco, gdyż aplikacja nie działa w czasie rzeczywistym. Każdy z nich ma wartość domyślną, a podwójne kliknięcie powoduje wyśrodkowanie. Dla każdego potencjometru indywidualnie dobrany został zakres i krok, w zależności od kontrolowanego parametru. Dokonano tego z użyciem funkcji `setRange()`, która jako argument przyjmuje wartość minimalną, maksymalną i krok. W przypadku pól wyboru, dla każdego parametru może być wybrana tylko jedna wartość w danej chwili. Takie zachowanie zapewnione jest przez przekazanie tej samej liczby całkowitej do metody `setRadioGroupId()` wybranych obiektów klasy `ToggleButton`.

6.3. Część funkcjonalna – implementacja toru przetwarzania sygnału

Operacja wygenerowania efektu pogłosu dla sygnału znajdującego się w głównym buforze inicjowana jest poprzez naciśnięcie przez użytkownika przycisku *Add Reverb*. Przycisk działa na tej samej zasadzie co przyciski opisane w poprzednim podrozdziale i powoduje ostatecznie wykonanie funkcji `processReverbButtonClicked()`, która stanowi metodę klasy `MainComponent`. Wewnątrz tej funkcji, w pierwszej kolejności wywoływane są funkcje ustawiające zmienne zawierające aktualne wartości czterech potencjometrów. Są to publiczne metody klasy `ReverbProcessor`, które wykorzystują metodę `getValue()` każdego z obiektów klasy `Slider` i przypisują te wartości prywatnym zmiennym wewnątrz swojej klasy. Na tej samej zasadzie ustawiane są zmienne powiązane z polami wyboru, w których wybrane przez użytkownika pole odszukiwane jest z użyciem funkcji `getToggleState()`. Program umożliwia kontrolę nad następującymi parametrami efektu:

- Wzmocnienie wejściowe i wzmocnienie wyjściowe (ang. *Input Gain, Output Gain*) - Aplikacja umożliwia osłabienie sygnału na wejściu algorytmu i wzmocnienie go na wyjściu w celu optymalnego dopasowania amplitudy do obsługiwanego przez plik WAV zakresu. Do wyboru jest 5 dB, 10 dB, 20 dB i 40 dB zarówno przy tłumieniu, jak i wzmocnieniu. Testy aplikacji wykazały, że takie wartości umożliwiają dość swobodne dopasowanie amplitudy dla kilku zróżnicowanych dynamicznie sygnałów.

- Potencjometr balansu (ang. *Balance*) – umożliwia płynną kontrolę nad proporcjami sygnału bezpośredniego i efektu pogłosu w sygnale wyjściowym. Działa w zakresie od -1 do 1 z krokiem 0,05. Wartość 0 powoduje uzyskanie równych proporcji. Wartości ujemne prowadzą do osłabienia efektu, a dodatnie do osłabienia sygnału bezpośredniego. Skrajne wartości pozwalają na uzyskanie sygnału wynikowego bez efektu (ang. *Dry*) lub bez dźwięku bezpośredniego (ang. *Wet*). Domyślna wartość to 0.
- Potencjometr czasu pogłosu (ang. *Reverb Time*) – pozwala na ustawienie wartości $t_{60}(0)$ w sekundach. Potencjometr działa w zakresie 0 s do 5 s z krokiem 0,05 s. Domyślna wartość to 1,5 s.
- Potencjometr tłumienia (ang. *Dampening*) – umożliwia kontrolę nad stopniem tłumienia wysokich częstotliwości w ogonie pogłosowym. Dla ułatwionej obsługi jest skalibrowany w procentach, gdzie 0% i 100% wprowadzają odpowiednio zerowe i najwyższe tłumienie. Wartości zmieniają się co 1%, a domyślna wartość to 75%.
- Potencjometr opóźnienia początkowego (ang. *Initial Delay*) – pozwala na przesunięcie ogona pogłosowego względem dźwięku bezpośredniego, co umożliwia uzyskanie wrażenia większej przestrzeni. Regulacja występuje w zakresie 0 – 500 ms z krokiem 1 ms.

Po odczytaniu wartości ustawionych przez użytkownika parametrów, wykonywane są dwie funkcje składowe klasy `ReverbProcessor` – `setupReverbProcessor()` i `addReverb()`. Funkcje te odpowiadają bezpośrednio za wykonanie algorytmu sieci pogłosowej FDN. Pierwsza z tych metod służy do wyznaczenia wartości współczynników wymaganych przez algorytm na podstawie odczytanych wcześniej parametrów. Zgodnie z Rys. 31, `setupReverbProcessor()` doprowadza do ustawienia wartości zestawu innych zmiennych prywatnych swojej klasy. Dopiero te zmienne są brane pod uwagę przez funkcję dodającą efekt do sygnału. Niektóre z nich są ustawiane już na etapie działania konstruktora klasy, jako że pozostają niezmienne niezależnie od parametrów użytkownika. Tymi składowymi są: dwunastoelementowa tablica liczb całkowitych zawierająca długości linii opóźniających w próbkach (identyczne jak w Tab. 4) oraz macierz rozpraszająca, zaimplementowana jako obiekt klasy JUCE `dsp::Matrix`.

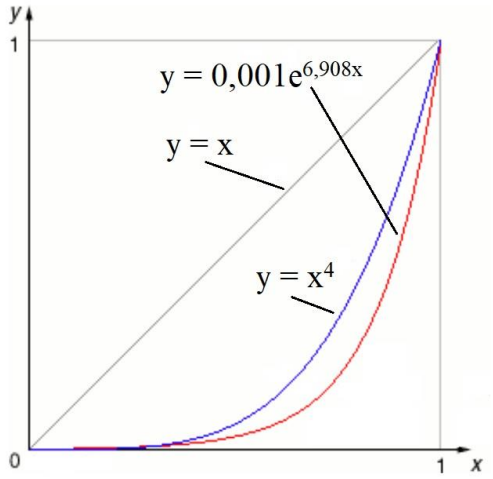
W pierwszej kolejności ustawiane są bufory, na których operuje algorytm – wejściowy i wyjściowy. Bufor wejściowy jest kopią bufora głównego. W funkcji obliczającej parametry dokonywane jest jego wydłużenie, a dodane próbki wypełniane są wartością 0. W celu „zmieszczenia" ogona pogłosowego, bufor wejściowy przedłużany jest o liczbę próbek odpowiadającą sumie czasu pogłosu i ustawionego opóźnienia wstępnego oraz dodatkowych 250 ms. Bufor wyjściowy, który w momencie wywołania funkcji jest pusty (zapełniony wartościami 0) jest także ustawiany na identyczną długość w próbkach. W tym celu wykorzystywana jest metoda `setSize()`.

Kolejnym etapem jest ustawienie współczynników wzmocnienia. Jako, że zmienne wybierane przez użytkownika są w skali decybelowej, na tym etapie konieczne jest wyznaczenie na ich podstawie parametrów w postaci liczb dziesiętnych. JUCE oferuje w tym celu klasę `Decibels`, posiadającą metodę `decibelsToGain()`. Metoda ta realizuje następujące działanie matematyczne:

$$coeff = 10^{\frac{decibels}{20}} \tag{54}$$

Gdzie:
coeff – współczynnik wzmocnienia w formie liczby dziesiętnej,
decibels – współczynnik wzmocnienia w decybelach.

Tak obliczony współczynnik wzmocnienia wejściowego jest bezpośrednio przypisywany do parametru *b* sieci FDN, zaś współczynnik wzmocnienia wyjściowego wraz z wartością balansu służą do obliczenia parametrów *c* i *d*. W zależności od tego, czy użytkownik wybrał dodatni czy ujemny balans, jeden z tych parametrów otrzymuje wartość 1, podczas gdy drugi równy jest czwartej potędze absolutnej wartości balansu. Obecność tego przelicznika wynika z konieczności zapewnienia potencjometrowi nieliniowej charakterystyki [38]. Aby percepcyjnie zmiana poziomu efektu bądź sygnału bezpośredniego odbierana była jako liniowa, współczynnik realizujący osłabianie sygnału powinien zmieniać się w sposób wykładniczy. Dobrym przybliżeniem takiego stanu rzeczy jest użycie funkcji potęgowej, przy czym czwarta potęga odpowiada krzywej dla zakresu dynamicznego kontrolowanego sygnału równego 60 dB. Wykres przedstawiający tą zależność jest przedstawiony na Rys. 33.



Rys. 33 Porównanie charakterystyk potencjometrów regulujących poziom sygnału [38].

Następnym etapem było przygotowanie filtrów, które używane są przez algorytm sieci pogłosowej. Platforma JUCE zawiera moduł `dsp`, w który zamieszczone są realizacje struktur wykorzystywanych w przetwarzaniu sygnałów. Jedną z nich jest szablon klasy `Filter`, która zawiera implementację filtra o nieskończonej odpowiedzi impulsowej. Implementacja ta opiera się o strukturę transponowaną realizacji bezpośredniej układu IIR, a więc tą samą, któ-

ra wykorzystana została w bibliotece SciPy. Jej opis zamieszczony został w podrozdziale 5.2, a schemat przepływu sygnału na Rys. 16. Jako, że współczynniki filtrów, a co z tego wynika obiekty klasy `Filter` są parametrami zmiennymi w czasie działania programu i zależnymi od parametrów wybranych przez użytkownika, muszą być one tworzone i likwidowane dynamicznie. W celu usprawnionej obsługi takich obiektów, wykorzystano klasy JUCE `OwnedArray` oraz `OptionalScopedPointer`. Przechowują one inteligentne wskaźniki (w pierwszym przypadku tablicę takich wskaźników). Inteligentny wskaźnik pozwala na zapewnienie indywidualnego dostępu do danego miejsca w pamięci. Jako, że jest obiektem klasy, przed jego zniszczeniem wywoływany jest destruktor, w którym umieszczony jest kod zwalniający zaalokowaną pamięć [39]. W ten sposób inteligentne wskaźniki zwalniają programistę z obowiązku każdorazowego używania `delete` podczas alokacji pamięci, likwidując przechowywane obiekty w momencie, gdy wykonany zostanie kod definiujący ich zasięg. Takie wskaźniki zostały wykorzystane do przechowywania obiektów implementujących filtr korekcyjny, a także obiektów filtrów tłumiących (dwunastoelementowa tablica).

Podczas testów aplikacji natrafiono na znaczący problem. W wyniku doboru niskiej wartości parametru α przy długim czasie pogłosu t_{60}, w programie występowała pętla nieskończona, która uniemożliwiała uzyskanie wyniku. Przeprowadzony proces debugowania pozwolił stwierdzić, że przyczyną takiego zachowania aplikacji jest występowanie wartości któregoś z parametrów p_i większej bądź równej 1. Analizując równanie (46) można stwierdzić, że taka sytuacja doprowadza do utraty stabilności filtru i stanu nieskończonego wzrostu amplitudy wyjściowej, aż do przepełnienia zakresu numerycznego zmiennych. Rozwiązaniem tej sytuacji jest wprowadzenie dla danego czasu t_{60} parametru α_{min}, najmniejszej wartości α pozwalającej na zachowanie stabilności układu. Parametr ten jest uzyskiwany poprzez następujące rozumowanie:

$$\frac{\ln(10)}{4}\left(-\frac{3M_iT}{t_{60}}\right)\left(1-\frac{1}{\alpha^2}\right)<1 \tag{55}$$

$$\frac{4t_{60}}{-3M_iTln(10)}<1-\frac{1}{\alpha^2} \tag{56}$$

$$1-\frac{1}{\alpha_{min}^2}=x \tag{57}$$

$$\alpha_{min}=\sqrt{\frac{1}{1-x}};\ x<1 \tag{58}$$

$$x=\frac{4t_{60}}{-3M_{max}Tln(10)}<1 \tag{59}$$

Gdzie:
M_{max} – największa długość linii opóźniającej, w przypadku opisywanej implementacji równa 4999 próbek.

Wartość ustawiona na potencjometrze tłumienia jest w takiej sytuacji kalkulowana na wartość α zgodnie z następującym równaniem:

$$\alpha = \alpha_{min} + (100 - d)\left(\frac{1 - \alpha_{min}}{100}\right) \tag{60}$$

Gdzie:
d – wartość ustawiona na potencjometrze tłumienia, w zakresie 0 – 100.

Parametr czasu pogłosu jest używany przez algorytm bezpośrednio i nie wymaga przeliczania. W ostatnim kroku w funkcji `setupReverbProcessor()` tworzone są filtry, których współczynniki uzyskiwane są zgodnie z równaniami (47) i (48).

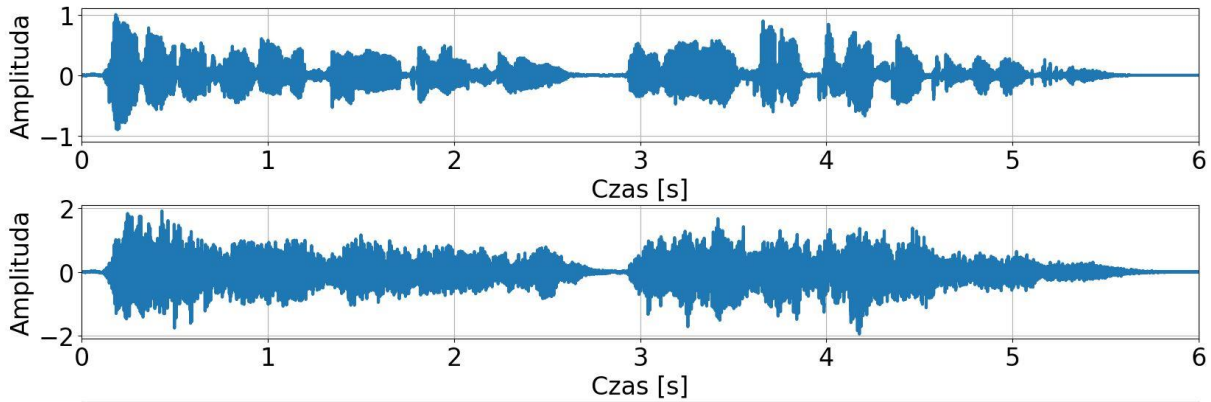
W następnej kolejności wywoływana jest funkcja `addReverb()`. Mechanizm realizacji algorytmu sieci pogłosowej FDN oparty jest na działaniu na pojedynczych próbkach, więc głównym elementem implementacji jest zagnieżdżona pętla, iterująca po wszystkich próbkach bufora wejściowego w każdym z jego kanałów. Najważniejszym elementem tej funkcji, obok opisanych wcześniej filtrów, są linie opóźniające, które oparte są na kolejce dwustronnej (ang. *double – ended queue*). Wybór tej struktury danych wynika z faktu, iż pozwala ona na dodawanie i usuwanie elementów zarówno z początku, jak i z końca wektora [40]. Wykorzystano implementację zawartą w bibliotece standardowej języka C++, `std::deque`. Linia opóźniająca realizowana jest poprzez dodawanie każdej przetworzonej próbki na początek kolejki z wykorzystaniem metody `push_front()`, podczas gdy w tej samej iteracji próbką wykorzystywaną do dalszych operacji jest próbka odczytana z końca kolejki za pomocą metody `back()`. Długość linii jest równa długości opóźnienia, które chce się uzyskać i jest utrzymywana na stałym poziomie przez usuwanie z kolejki próbki, która została już odczytana. W taki sposób działa linia wprowadzająca opóźnienie początkowe. Jej długość jest przeliczana z milisekund na próbki z wykorzystaniem częstotliwości próbkowania (44100 Hz). Kolejka dwustronna tworzona jest na początku funkcji `addReverb()`. Poniższy fragment kodu wprowadza opóźnienie na wyjściu:

```
1. initDelayLine.push_front(outSample);
2. initDelayLine.pop_back();
3. outputWrite[sample] = ((inputRead[sample] * bGain * balanceDry)
4.                        + initDelayLine.back()) * outputGain;
```

`outSample` jest próbką wyjściową filtra korekcyjnego, `inputRead` jest buforem wejściowym, a `outputWrite` buforem wyjściowym. W każdej iteracji pętli sprawdzany jest warunek, czy wartość próbki mieści się w przedziale < -1,0; 1,0 >, na takich wartościach ope-

ruje bowiem klasa `AudioBuffer`. Jeżeli tak nie jest, zmienna logiczna `isClipping` przybiera wartość 1, co powoduje zmianę koloru diody *Clip* na interfejsie użytkownika na kolor czerwony. Zmiana ta wykonywana jest w funkcji `paint()`. Jest to informacja o tym, że należy zmniejszyć wzmocnienie sygnału. Po zakończeniu wykonywania pętli, oczekiwany sygnał wyjściowy znajduje się w buforze wyjściowym klasy `ReverbProcessor`. Funkcja `addReverb()` zwraca więc referencję do obiektu klasy `AudioBuffer`, która na koniec operacji algorytmu przypisywana jest do głównego bufora klasy `MainComponent`. Dzięki takiemu rozwiązaniu, sygnał z pogłosem może zostać odtworzony lub zapisany do pliku w identyczny sposób jak wczytany plik, bez konieczności uwzględniania dodatkowych funkcjonalności w kodzie. Warto nadmienić, że bufor wejściowy pozostaje bez zmian i zawiera nadal sygnał wejściowy, co pozwala na przygotowanie nowej symulacji bezpośrednio po zakończeniu działania funkcji `processReverbButtonClicked()`, bez konieczności ponownego wczytywania pliku początkowego. Ostatnim zadaniem wykonywanym po naciśnięciu przycisku *Add Reverb* jest zaktualizowanie miniatury, która powinna od tej chwili zawierać obraz fali wygenerowanego właśnie sygnału. Do wykonania tej czynności wykorzystano metodę klasy `AudioThumbnail`, `addBlock()`.

Zbudowana w ten sposób aplikacja cechuje się intuicyjnym sposobem obsługi i względnie prostą strukturą, umożliwiającą korzystanie z niej bez posiadania wcześniejszego doświadczenia w pracy z aplikacjami audio. Program zapewnia uzyskanie wyniku przy każdej konfiguracji parametrów i umożliwia bardzo dokładne dopasowanie ustawień brzmieniowych do preferencji użytkownika. Dzięki platformie JUCE interfejs graficzny jest bardzo responsywny i cechuje się atrakcyjnym wyglądem. Współpraca między częścią interfejsową a funkcjonalną przebiega sprawnie, rezultaty działania wszystkich elementów kontrolnych i przycisków są powtarzalne. Jeśli chodzi o brzmienie efektu, z powodu implementacji tego samego schematu przepływu sygnału co w programie prototypowym jest ono tożsame z tym opisanym w podrozdziale 5.3. Na Rys. 34 przedstawiono wynik działania programu.



Rys. 34 Wynik działania finalnej aplikacji. Od góry: prawy kanał sygnału podstawowego, fragmentu męskiej mowy; prawy kanał sygnału wynikowego, uzyskanego dla $t_{60}(0)$ = 0,7 s, α = 0,8 i jednostkowych współczynników wzmocnień.

7. Podsumowanie i wnioski

W toku niniejszej pracy opracowano metodę uzyskania cyfrowej symulacji akustycznego zjawiska pogłosu, która oparta jest na założeniu percepcyjnej dokładności i umożliwia użytkownikowi kontrolę nad parametrami brzmieniowymi. Metoda ta została wykorzystana w implementacji aplikacji dźwiękowej, która pozwala na wzbogacenie o odpowiednio dobrany sztuczny pogłos sygnału akustycznego przechowywanego w nieskompresowanym pliku WAV. Proces tworzenia finalnej aplikacji poprzedzony został procedurą doboru odpowiedniego do zastosowania algorytmu. Procedura ta oparta była na rozległym przeglądzie literaturowym i implementacji wybranych rozwiązań programistycznych w formie skryptu w języku Python. W oparciu o te wnioski można stwierdzić, że cel niniejszej pracy został całkowicie zrealizowany. Planowana początkowo aplikacja działa i spełnia swoje założenia.

Wprowadzenie tematu rozpoczęto od przedstawienia fizycznej interpretacji zjawiska rewerberacji i jego najważniejszego parametru – czasu pogłosu. Nawiązanie do mechanizmu powstawania opisywanego efektu w rzeczywistych pomieszczeniach umożliwiło dogłębne i merytorycznie poprawne spojrzenie na sposoby wytwarzania symulacji – pozwoliło na zrozumienie zasadności występowania w nich m. in. linii opóźniających i filtrów dolnoprzepustowych jako elementów bezpośrednio modelujących zachowanie fal dźwiękowych. Dokonany w następnej kolejności opis etapów wyróżnianych w pogłosowej odpowiedzi impulsowej pomieszczeń umożliwił zaś świadome odniesienie struktur występujących w algorytmach modelujących do zjawiska, do którego się odnoszą (np. blok opóźniający w procesorze Moorera emulujący etap wczesnych odbić). Znajomość zaprezentowanych w pracy podstawy teoretycznych opisywanego tematu ułatwiła zrozumienie zasady działania algorytmów i metod elektromechanicznych, a poznane na tym etapie pojęcia okazały się nieodłącznym elementem dalszych badań.

Kolejną część pracy stanowił opis istniejących metod symulacji pogłosu – zarówno wykorzystywanych w czasach powszechności technik analogowych, jak i współczesnych, opartych na operacjach wykonywanych przez maszyny obliczeniowe. Poznanie mnogości opracowanych już rozwiązań umożliwiło bardziej całościowe podejście do problemu ostatecznej implementacji. Mimo zbieżności efektów brzmieniowych większości metod, sama problematyka cechuje się istnieniem bardzo wielu silnie różniących się od siebie ideowo koncepcji. Pomimo to, rozwój dziedziny jest metodyczny. Naukowcy i twórcy komercyjnych aplikacji korzystają intensywnie z dokonań poprzedników, dzięki czemu postęp dziedziny efektów dźwiękowych jest po dziś dzień niezwykle dynamiczny. Przeprowadzono głęboki przegląd istniejących rozwiązań i na podstawie stopnia ich zaawansowania, testów odsłuchowych, opinii badaczy i użytkowników oraz wkładu w rozwój tematyki dokonano wyboru 3 rozwiązań cyfrowych, które szczegółowo opracowano w dalszej części pracy. Do metod tych należały: pogłos splotowy, pogłos oparty o filtry grzebieniowe i wszechprzepustowe oraz sieci pogłosowe FDN. Każda z tych metod cechuje się zupełnie innym podejściem i co za tym idzie od-

miennymi efektami brzmieniowymi. Są one w swojej podstawowej koncepcji proste, jednak lata badań i użytkowania doprowadziły do powstania wielu złożonych rozszerzeń. Każda z metod zaimplementowana została w oparciu o modelową strukturę, zaprezentowaną jako wynik badań naukowych i kolektywnie uznaną za wzór do dalszych poszukiwań przez środowisko związane z przetwarzaniem dźwięku. Rozwiązanie problemów stojących przed programistą realizującym te algorytmy umożliwiło dogłębne zbadanie rzeczywistego sposobu ich operacji. Wykonanie części prototypowej projektu pozwoliło również na usystematyzowanie wiedzy dotyczącej podstawowych działań na plikach dźwiękowych, ich struktury wewnętrznej oraz warunków, które muszą być spełnione, aby uzyskać poprawną reprezentację sygnału akustycznego. Jako, że implementacje testowe wykonane były w języku Python bez wykorzystania specjalistycznych bibliotek do przetwarzania dźwięku, odczyt i zapis plików nie był wspomagany przez już zdefiniowane funkcje i odbywał się w oparciu o niskopoziomowe operacje. Wykonanie prototypów wymagało wykorzystania wiedzy programistycznej w zakresie m.in. działań numerycznych i wizualizacji danych. Sam proces analizy gotowych prototypów wymagał chwilowego odejścia od zagadnień czysto inżynierskich i skupienia się na problematyce samego brzmienia efektu – nie istnieją bowiem miary służące ocenie, czy symulacja efektu jest percepcyjnie dokładna, czy nie. Jedynym sposobem na uzyskanie takiej wiedzy jest test odsłuchowy i testy porównawcze z innymi dostępnymi symulacjami i rzeczywistym pogłosem. Należało mieć również na uwadze, że w zamyśle efekty takie wykorzystywane są w dziedzinach, w których niebanalne znaczenie ma estetyka, np. produkcja muzyczna czy produkcja filmowa. Finalny efekt dźwiękowy musi więc, poza brzmieniową dokładnością, cechować się byciem „przyjemnym dla ucha”. Taka dwoistość wymagań była czynnikiem wymuszającym w toku pracy wyjątkowe położenie akcentu na trening słuchu i zebranie doświadczeń poprzez analizę brzmienia wielu symulacji różnego pochodzenia. Ostateczną selekcję i decyzję dotycząca doboru algorytmu użytego w finalnym programie podjęto w dużej mierze na podstawie takich właśnie przesłanek brzmieniowych. Zadecydowano, że wyartykułowane wcześniej założenia najlepiej spełnia algorytm sieci pogłosowych FDN, który był również najbardziej technicznie rozwiniętym i najdokładniej zdefiniowanym teoretycznie rozwiązaniem ze wszystkich poddanych analizie. W kontekście samego zadania programistycznego, sieci pogłosowe FDN okazały się dość wymagające. Na ich strukturę składa się kilka niezależnych elementów, które połączone są w dość nieintuicyjną architekturę pętli sprzężenia zwrotnego. W czasie pracy nad aplikacją zaprojektowano dwie metody realizacji sieci pogłosowych, każda charakteryzująca się podejściem reprezentatywnym dla używanego języka programowania (operacje na pełnych wektorach w języku Python z użyciem biblioteki NumPy, iteracyjne przetwarzanie próbek w języku C++). Obie wersje pozwalały na uzyskanie identycznych wyników końcowych, mimo dość znacznych różnic w budowie wewnętrznej. Uzyskanie w pełni funkcjonalnego i brzmieniowo zadowalającego algorytmu okazało się czasochłonne, zarówno ze względu na konieczność przeniesienia teorii cyfrowego przetwarzania sygnałów na grunt użyteczności w ostatecznym skrypcie, ale także przeprowadzenia wymagających testów odsłuchowych i procesu „strojenia” algorytmu, czyli doboru najodpowiedniej-

szych parametrów odpowiedzialnych za charakterystykę efektu dźwiękowego. W zadaniu tym pomocne okazały się materiały i wskazówki opracowane przez inżynierów dźwięku, programistów aplikacji audio oraz użytkowników procesorów efektowych.

Najbardziej wymagającym etapem pracy okazało się stworzenie ostatecznej aplikacji. W tej części pracy, algorytm musiał zostać połączony z pobocznymi funkcjonalnościami w celu zapewnienia możliwości intuicyjnej obsługi przez użytkownika. W celu wydajnej i efektywnej implementacji posłużono się platformą programistyczną JUCE. Ten szkielet aplikacyjny oferuje niezwykle szeroki wybór funkcjonalności, które znacząco ułatwiają pracę nad programami dźwiękowymi, jest jednak dość wymagający w użytkowaniu. Cechuje się złożoną zależnością klas i metod, której znajomość jest często absolutnie wymagana do skorzystania z oferowanych przez platformę zasobów. Bardzo pomocnym materiałem okazały się samouczki i forum użytkowników JUCE, gdzie znaleźć można porady autorstwa samych twórców bibliotek. JUCE oferuje klasy implementujące zarówno bazowe funkcjonalności, jak i wykonujące wysoce specyficzne zadania, związane m.in. z zarządzaniem pamięcią, obsługą operacji numerycznych czy programowaniem wielowątkowym. Jest to narzędzie pozwalające na tworzenie zaawansowanych programów bez konieczności gromadzenia zestawu bibliotek realizujących odmienne zadania, co jest niezwykłą zaletą tej platformy.

Ostateczna aplikacja jest wysoce zadowalającą implementacją podstawowych procesów związanych z symulacją percepcyjnych efektów akustycznych. Jak na indywidualną aplikację dźwiękową posiada dość szeroki zakres funkcjonalności. W toku jej realizacji udało się rozwiązać szereg problemów wynikających z dość subtelnych zależności programistycznych, m.in. wielowątkowości aplikacji czy komunikacji między modułem interfejsowym a funkcjonalnym. Mimo to, oczywistym jest, że obecny kształt programu oferuje wiele możliwości modyfikacji. Możliwymi do wprowadzenia ulepszeniami jest np. poszerzona obsługa błędów i wyjątków, zwiększenie możliwości odtwarzacza plików audio i modułu wizualizacji kształtu fali czy polepszenie wydajności implementacji algorytmu sieci pogłosowej. W dalszej perspektywie taki program mógłby zyskać możliwość działania w czasie rzeczywistym i zostać poddany konwersji na działającą w programie – gospodarzu (aplikacji do cyfrowej obróbki dźwięku) wtyczkę VST.

Tworzenie symulacji efektów dźwiękowych jest zagadnieniem interdyscyplinarnym, łączącym zagadnienia z zakresu przetwarzania sygnałów, programowania i akustyki. Konieczna jest w nim umiejętność przeprowadzania szczegółowych testów odsłuchowych i obycie z terminologią i sposobem użytkowania oprogramowania dźwiękowego. Ważnym aspektem w toku projektowania aplikacji jest zdystansowanie końcowego użytkownika od szczegółów implementacyjnych i udostępnienie mu kontroli jedynie nad ogólnie rozpoznawalnymi i ściśle określonymi parametrami. Programy wytwarzające symulacje efektów dźwiękowych powinny cechować się wydajnością i elastycznością, a umiejętności potrzebne programiście do zapewnienia tych warunków zdobywane są w toku długoletniej pracy. Przedstawiony w niniejszych rozważaniach projekt stanowi obiecujący punkt startowy dla takiej właśnie ścieżki rozwoju.

8. Bibliografia

[1] Alton Everest F., *Podręcznik Akustyki*, Wydawnictwo Sonia Draga, Katowice 2004.

[2] Serwis edukacyjny Katedry Systemów Multimedialnych Politechniki Gdańskiej, *Technika nagłaśniania. Czas pogłosu pomieszczenia.*, https://sound.eti.pg.gda.pl/student/elearning/cp.htm [dostęp 9 grudnia 2019].

[3] Kuttruff H., *Acoustics. An Introduction*, Taylor & Francis, Londyn i Nowy Jork 2006.

[4] Sabine W. C., *Collected Papers on Acoustics*, Harvard University Press, Cambridge 1922

[5] Drobner M. *Akustyka Muzyczna*, Polskie Wydawnictwo Muzyczne, Kraków 1973.

[6] Moorer J. A., *About This Reverberation Business*, Computer Music Journal Vol. 3, No. 2, lipiec 1979, s. 13 – 28.

[7] Serwis edukacyjny Katedry Systemów Multimedialnych Politechniki Gdańskiej, *Technika nagłaśniania. Czas pogłosu pomieszczenia. Pomiar czasu pogłosu.,* https://sound.eti.pg.gda.pl/student/elearning/cp_pom.htm [dostęp 9 grudnia 2019].

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

[9] Makarewicz R., Wykład nt. echa, pogłosu, dźwięku i hałasu w komorze bezechowej, https://www.youtube.com/watch?v=dKywjLt9hAs [dostęp 9 grudnia 2019].

[10] Brown A. D., Stecker G. C., Tollin D. J., *The Precedence Effect in Sound Localization*, Journal of the Association for Research in Otolaryngology : JARO Vol. 16, No. 1, luty 2015, s. 1 – 28.

[11] Jot J. – M., Chaigne A., *Digital Delay Networks for Designing Artificial Reverberators*, 90th Convetion of Audio Engineering Society, luty 1991, przeddruk 3030.

[12] Smyth T., *Frequency Dependance of Reverb Time*, wykład *Music 175: Time and Space* http://musicweb.ucsd.edu/~trsmyth/space175/Frequency_Dependence_Reverb.html [dostęp 9 grudnia 2019].

[13] Gardner W. G., *Reverberation Algorithms.*, [w:] *Applications of Digital Signal Processing to Audio and Acoustics.*, Kahrs M., Brandenburg K., The International Series in Engineering and Computer Science, Vol. 437, Springer, Boston 1998.

[14] Przedpełska – Bieniek M., *Sztuka Dźwięku. Technika i Realizacja*, Wydawnictwo Wojciech Marzec, Warszawa 2017.

[15] Przedpełska – Bieniek M., *Dźwięk w Filmie*, Agencja Producentów Filmowych, Warszawa 2006.

[16] Moore F. R., *An Introduction to the Mathematics of Digital Signal Processing: Part II: Sampling, Transforms, and Digital Filtering*, Computer Music Journal Vol. 2, No. 2, wrzesień 1978, s. 38 – 60.

[17] Roads C., *The Computer Music Tutorial*, The MIT Press, Cambridge 1996.

[18] Kostek B., wykład *Modelowanie pola akustycznego* na przedmiocie Technika Nagłaśniania w Katedrze Systemów Multimedialnych Politechniki Gdańskiej, https://sound.eti.pg.gda.pl/student/tnagl/Modelowanie.pdf [dostęp 9 grudnia 2019].

[19] Schroeder M. R., *Natural Sounding Artificial Reverberation*, Journal of the Audio Engineering Society Vol. 10, No. 3, lipiec 1962, s. 219 – 223.

[20] Gardner W. G., *The Virtual Acoustic Room*, praca dyplomowa magisterska na Massachusetts Institute of Technology, Cambridge 1992.

[21] Stautner J., Puckette M., *Designing Multi - Channel Reverberators*, Computer Music Journal Vol. 6, No. 1, kwiecień 1982, s. 52 – 65.

[22] Smith J. O. III, *Physical Audio Signal Processing: for Virtual Musical Instruments and Digital Audio Effects*, wyd. online, 2010, http://ccrma.stanford.edu/~jos/pasp/ [dostęp 9 grudnia 2019].

[23] Weisstein E. W., *Hadamard Matrix*, [w:] MathWorld – A Wolfram Web Resource, http://mathworld.wolfram.com/HadamardMatrix.html [dostęp 9 grudnia 2019].

[24] Schroeder M. R., Logan B. F., *"Colorless" Artificial Reverberation*, Journal of the Audio Engineering Society Vol. 9, No. 3, lipiec 1961, s. 192 – 197.

[25] Rocchesso D., Smith J. O. III, *Circulant and Elliptic Feedback Delay Networks for Artificial Reverberation*, IEEE Transactions on Speech and Audio Vol. 5, No. 1, styczeń 1996, s. 51 – 60.

[26] Moduł *wave* w dokumentacji standardowej biblioteki języka Python, https://docs.python.org/3/library/wave.html [dostęp 9 grudnia 2019].

[27] Moduł *struct* w dokumentacji standardowej biblioteki języka Python, https://docs.python.org/3/library/struct.html [dostęp 9 grudnia 2019].

[28] Free Reverb Impulse Responses, Voxengo, https://www.voxengo.com/impulses/ [dostęp 9 grudnia 2019].

[29] Funkcja *scipy.signal.convolve* w dokumentacji biblioteki SciPy, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve.html [dostęp 9 grudnia 2019].

[30] Funkcja *scipy.signal.lfilter* w dokumentacji biblioteki SciPy, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html [dostęp 9 grudnia 2019].

[31] Smith J. O. III, *Introduction to Digital Filters with Audio Applications*, wyd. online, 2007, http://ccrma.stanford.edu/~jos/filters/ [dostęp 9 grudnia 2019].

[32] Zając W., wykład *Dyskretna transformacja kosinusowa* na przedmiocie Cyfrowe przetwarzanie i kompresja danych w Instytucie Sterowania i Systemów Informatycznych Uniwersytetu Zielonogórskiego, http://staff.uz.zgora.pl/wzajac/CPIK%20W04.pdf [dostęp 9 grudnia 2019].

[33] Plik *revmodel.cc* w kodzie źródłowym aplikacji Freeverb, udostępnionym poprzez platformę GitHub, https://github.com/tim-janik/beast/blob/master/plugins/freeverb/revmodel.cc [dostęp 9 grudnia 2019].

[34] Strona główna platformy programistycznej JUCE, https://juce.com/ [dostęp 9 grudnia 2019].

[35] Klasa *AudioAppComponent* w dokumentacji platformy programistycznej JUCE, https://docs.juce.com/master/classAudioAppComponent.html [dostęp 9 grudnia 2019].

[36] Klasa *Component* w dokumentacji platformy programistycznej JUCE, https://docs.juce.com/master/classComponent.html [dostęp 9 grudnia 2019].

[37] Szablon klasy *std::atomic* w dokumentacji biblioteki standardowej języka C++, https://en.cppreference.com/w/cpp/atomic/atomic [dostęp 9 grudnia 2019].

[38] Thomas A., *Programming Volume Controls*, https://www.dr-lex.be/info-stuff/volumecontrols.html#table1 [dostęp 9 grudnia 2019].

[39] Zaawansowane C++ / Wykład 10: Inteligentne wskaźniki, http://wazniak.mimuw.edu.pl/index.php?title=Zaawansowane_CPP/Wyk%C5%82ad_10:_Inteligentne_wska%C5%BAniki [dostęp 9 grudnia 2019].

[40] Szablon klasy *std::deque* w dokumentacji biblioteki standardowej języka C++, https://en.cppreference.com/w/cpp/container/deque [dostęp 9 grudnia 2019].