AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI

PRACA DYPLOMOWA
magisterska

Zastosowanie procesorów graficznych do obliczeń akustycznych

Implementation of graphics processing units for acoustic calculations

Autor: **Dominik Szymaniak**
Kierunek studiów: Inżynieria Akustyczna

Opiekun pracy: **dr Marek Pluta**

. . . . . .
podpis

Kraków, rok 2019

OŚWIADCZENIA STUDENTA

Kraków, dnia 19 września 2019

Dominik Szymaniak
Imiona i nazwisko studenta

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

Wydział Inżynierii Mechanicznej i Robotyki
Nazwa wydziału

dr Marek Pluta
Imiona i nazwisko opiekuna pracy dyplomowej

Ja niżej podpisany oświadczam, że:

jako twórca/współtwórca* pracy dyplomowej magisterskiej pt.
Zastosowanie procesorów graficznych do obliczeń akustycznych

1. **uprzedzony o odpowiedzialności karnej** na podstawie art. 115 ust 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz. U. z 2018 r. poz. 1991, z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie", **a także uprzedzony o odpowiedzialności dyscyplinarnej** na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (tj. Dz. U. z 2018 r. poz. 1668, z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta." **niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i nie korzystałem ze źródeł innych niż wymienione w pracy;**
2. praca dyplomowa jest wynikiem mojej twórczości i nie narusza praw autorskich innych osób;
3. wersja elektroniczna przedłożonej w wersji papierowej pracy dyplomowej jest wersją ostateczną, która będzie przedstawiona komisji przeprowadzającej egzamin dyplomowy;
4. praca dyplomowa nie zawiera żadnych informacji podlegających ochronie na podstawie przepisów o ochronie informacji niejawnych ani nie jest pracą dyplomową, której przedmiot jest objęty tajemnicą prawnie chronioną;
5. [TAK]** udzielam nieodpłatnie Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie licencji niewyłącznej, bez ograniczeń czasowych, terytorialnych i ilościowych na udostępnienie mojej pracy dyplomowej w sieci Internet za pośrednictwem Repozytorium AGH.

. . . . . .
czytelny podpis studenta

Jednocześnie uczelnia informuje, że:

1. zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt. 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668, z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez 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.

Akademia Górniczo-Hutnicza im. Stanisława Staszica Kraków, 19 września 2019
Wydział Inżynierii Mechanicznej i Robotyki

Kierunek studiów: Inżynieria Akustyczna

Specjalność: Drgania i Hałas w Technice i Środowisku (D)

Dominik Szymaniak
Imiona i nazwisko studenta

Praca dyplomowa magisterska
Zastosowanie procesorów graficznych do obliczeń akustycznych
(tytuł pracy)

Opiekun: dr Marek Pluta

STRESZCZENIE

Praca stanowi przegląd dostępnych środowisk programistycznych wykorzystujących platformy heterogeniczne i opisuje, wykonany w ramach pracy, program komputerowy korzystający ze środowiska OpenCL. Autor przedstawia założenia przetwarzania heterogenicznego oraz założenia metody źródeł pozornych, które zostały użyte do napisania aplikacji. Głównym celem pracy jest implementacja metody źródeł pozornych w postaci programu na heterogeniczne platformy w taki spoób, aby wykorzystać możliwości tych platform i skrócić czas obliczeń. Aplikacja została napisana w technologii OpenCL ze względu na dużą uniwersalność środowiska. Umożliwia wyznaczenie siatki źródeł pozornych dla przygotowanego modelu geometrii pomieszczenia. Dodatkowo aplikacja pozwala na wizualizację poszczególnych promieni dźwiękowych przy użyciu programu GeoGebra. Program posłużył do przeprowadzenia obliczeń testowych. Obliczenia dla różnych zestawów danych przedstawiają możliwości i użyteczność metody źródeł pozornych. Autor wykonuje również obliczenia na różnych architekturach urządzeń, które wykazują przewagę heterogenicznych platform nad procesorami CPU w szybkości obliczeń dla danego algorytmu.

AGH University of Science and Technology
Faculty of Mechanical Engineering and Robotics

Kraków, September 19, 2019

Field of Study: Acoustic Engineering

Specialization: [Specialisation]

Dominik Szymaniak
(First name and family name of the student)

Engineer Diploma Thesis
Implementation of graphics processing units for acoustic calculations
(Thesis title)

Supervisor: dr Marek Pluta

SUMMARY

The diploma is an overview of available programming environments using heterogeneous platforms and describes the computer program made as part of the diploma which is using the OpenCL environment. The author presents the assumptions of heterogeneous processing and assumptions of the image source method that were used to write the application. The main purpose of the work is to implement the image source method in the form of a program on heterogeneous platforms in such a way as to take advantage of the capabilities of these platforms and reduce computation time. The application was written in OpenCL technology due to the high universality of the environment. It allows to determine the grid of image sources for the prepared model of room geometry. In addition, the application allows you to visualize individual sound rays using the GeoGebra program. The program was used to perform test calculations. Calculations for different data sets show the possibilities and usability of the image source method. The author also performs calculations on various device architectures that shows the advantage of heterogeneous platforms over CPUs in the speed of calculation for a given algorithm.

Kraków, dnia 19 września 2019

AKADEMIA GÓRNICZO-HUTNICZA
WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI

TEMATYKA PRACY DYPLOMOWEJ

Dominik Szymaniak
imię i nazwisko studenta

Tytuł pracy dyplomowej:
Zastosowanie procesorów graficznych do obliczeń akustycznych

Promotor pracy:	dr Marek Pluta
Recenzent pracy:	dr hab. Inż. Mariusz Giergiel, prof. AGH

. . . . . .
Podpis dziekana

PLAN PRACY DYPLOMOWEJ:

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

Praktyka (dyplomowa):
[Praktyka dyplomowa]

Kraków, ...
data *podpis dyplomanta*

. . . . . .
podpis promotora

Termin oddania do dziekanatu: . . . . . 20. . . r.

"Więc z punktu, mając na uwadze,
że ewentualna krytyka może być...
tak musimy zrobić,
żeby tej krytyki nie było...
tylko aplauz i zaakceptowanie (...)"
- "Rejs" M. Piwowskiego

Spis treści

1. Wstęp

1.1. Wprowadzenie

Wiele dziedzin z zakresu akustyki, a w szczególności akustyka pomieszczeń, wymaga analizy pola akustycznego. Zdefiniowanie przez Wallace Clement Sabine'a pojęcia czasu pogłosu i ustalenie empirycznego wzoru rozpoczęło nową erę akustyki architektonicznej i rozwój metod analizy pola akustycznego. Wzór Sabine'a [1] oraz jego kolejne przekształcenia nie uwzględniały wielu zjawisk akustycznych. Zarówno wzory empiryczne jak i metody statystyczne wyznaczały jedynie niektóre parametry pola akustycznego i częstowprowadzały duży błąd w odniesieniu do rzeczywistych wartości. Analityczne rozwiązanie równania falowego jak i wykorzystanie metody elementów skończonych były zbyt kosztowne lub niemożliwe. Jednymi z prostszych do implementacji metod analizy pola akustycznego były metody geometryczne. W 1948 roku L. Cremer w publikacji „Geometrische Raumakustik" [2] przedstawia geometryczną interpretację fali akustycznej. Praca ukazuje mechanizm odbicia fali od powierzchni płaskiej i stanowi wprowadzenie do geometrycznych metod analizy pola.

Powstałe metody numeryczne stosowane były w wielu dziedzinach analizy pól wektorowych. Rozwój technologii komputerowej doprowadził do optymalizacji i opłacalnego wykorzystania tych metod. Początkowo, ze względu na małe zapotrzebowanie na adaptacje akustyczne, metody analizy pola akustycznego nie były znacznie rozwijane. W tym czasie szybko rozwijającą się dziedziną stała się grafika komputerowa. Próby wygenerowania realistycznych obrazów przy użyciu technologii cyfrowych wymagały zamodelowania zjawiska rozchodzenia się światła. W 1968 Arhtur Appel w swojej publikacji [3] zamodelował punktowe źródło światła jako nieskończony zbiór półprostych odbijających się od powierzchni płaskich pod takim samym kątem, jak kąt padania. Powyższa metoda promieniowa (ang. Ray Tracing) zaczęła być wykorzystywana do renderowania grafiki trójwymiarowej. Poprzez podobieństwo rozchodzenia się fali dźwiękowej do świetlnej, w publikacji "Journal of Sound and Vibration " [4], przedstawiono wykorzystanie tej metody do obliczenia czasu pogłosu pomieszczenia. Dostrzeżono opłacalność metody promieniowej i zaczęto ją modyfikować, co doprowadziło do powstania kolejnych metod geometrycznych. Jedną z osób przyczyniającą się do rozwoju metod geometrycznych był W. Straszewicz. W swojej pracy [5] wykorzystuje on podstawy metody źródeł pozornych (ang. Image-Source). W 1979 J. Allen i D. Berkley zaimplementowali metodę źródeł pozornych przy użyciu technik cyfrowych [6]. Wysokie zapotrzebowanie na grafikę komputerową, rozwój gier komputerowych i niski koszt obliczeniowy doprowadziły do rozwoju powyższych metod symulacji fali. Należy jednak pamiętać, że metody geometryczne często nie uwzględniają wielu zjawisk falowych, co wymusza uzupełnianie ich o inne metody numeryczne.

1.2. Cel i zakres pracy

Metoda źródeł pozornych jest obecnie wykorzystywana w popularnych programach do symulacji akustycznych [7] [8]. W przypadku tych programów stanowi ona uzupełnienie innych metod i pozwala na wyznaczenie źródeł pozornych niskich rzędów. Jako niezależna metoda, przy niewydajnych obliczeniach, nie jest w stanie dokładnie odwzorować badanego środowiska akustycznego. Obliczenia przy użyciu tej metody dla źródeł pozornych wysokich rzędów są czasochłonne. Problem ten można rozwiązać zrównoleglając obliczenia przy użyciu kart graficznych. **Głównym założeniem poniższej pracy jest skrócenie czasu obliczeń dla danej metody poprzez implementację jej w postaci programu komputerowego wykorzystującego platformy heterogeniczne.** Aplikacja jako dane wejściowe ma przyjmować współrzędne przestrzenne punktu źródła dźwięku i punktu odbioru, tablicę powierzchni odbijających wraz z ich współczynnikami pochłaniania dźwięku oraz rząd obliczanych źródeł pozornych. Dane wyjściowe zostaną uzyskane w postaci siatki źródeł pozornych wraz z informacją o ilości pochłoniętej podczas odbić energii dla każdego źródła pozornego. Prezentacja danych ma zostać wykonana przez interfejs graficzny w programie GeoGebra w postaci ścieżki promienia dźwiękowego dla wybranego źródła pozornego oraz siatki źródeł pozornych.

Pomysł wykorzystania kart graficznych do implementacji metody źródeł pozornych pojawia się w publikacji [9], która wykorzystuje środowisko CUDA wspierające karty graficzne NVIDIA. Kolejną implementacją metody źródeł pozornych w środowisku kart graficznych jest aplikacja Wayverb [10], przeznaczona jedynie dla systemów typu macOS. **Ze względu na ograniczenia systemowe powyższych prac, kolejnym założeniem pracy autora jest wykonanie uniwersalnego programu, który może zostać uruchomiony na wiele popularnych środowiskach.** W tym celu autor wykorzystuje bibliotekę OpenCL, która umożliwia wykonanie kodu na większości platform heterogenicznych. Aplikacja została wykonana na systemy z rodziny Windows i Linux ze względu na popularność tych środowisk. Przy tych założeniach aplikacja może stanowić bazę do wykorzystania metody źródeł pozornych w innych aplikacjach.

2. Opis metody źródeł pozornych

2.1. Wprowadzenie

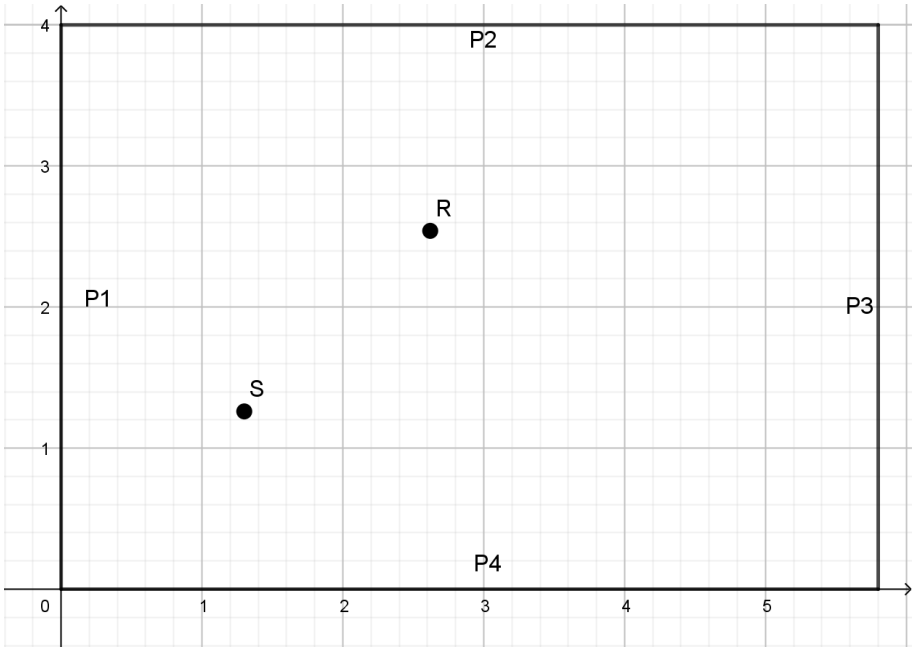
Metoda źródeł pozornych jest jedną z metod często wykorzystywanych w akustyce architektonicznej i cyfrowym przetwarzaniu sygnałów. Przy użyciu tej metody możemy wygenerować odpowiedź impulsową pomieszczenia, zbadać dyfuzyjność pola czy prześledzić ścieżkę propagacji dźwięku. Ze względu na liczne uproszczenia, w metodzie nie jest uwzględnianych wiele zjawisk falowych. W jej najprostszej implementacji pomijane są takie zjawiska jak dyfrakcja, interferencja czy rozproszenie fali. W wielu przypadkach, aby ta metoda była skuteczna, należy przeprowadzić złożone obliczenia lub wykorzystać ją równolegle z innymi metodami numerycznymi.

2.2. Główne założenia metody

W przypadku metod geometrycznych falę dźwiękową modelujemy jako prosty obiekt przestrzenny. W metodzie źródeł pozornych punktowe źródło zastępujemy nieskończonym zbiorem półprostych. Każdy taki promień dźwiękowy reprezentuje pewną część składową fali i jej kierunek propagacji. Każda składowa odbija się od powierzchni zgodnie z prawem Snella, a przy odbiciu zostaje pochłonięta część jej energii proporcjonalna do współczynnika pochłaniania dźwięku dla danego materiału. Kolejnym założeniem jest, że powierzchnie odbijające są powierzchniami płaskimi, a w polu występuje jedno punktowe źródło i jeden punkt odbioru. W przypadku większej ilości punktów obserwacji lub większej ilości źródeł należałoby skorzystać z szerszych metod, jakimi są metoda obrazów pozornych i metoda pozornych obrazów punktu obserwacji. Przy powyższych założeniach każdą ścieżkę propagacji promienia dźwiękowego można zastąpić pozornym źródłem. Źródło pozorne dla danej składowej fali powstaje w wyniku odbicia lustrzanego punktu źródła względem powierzchni odbijającej tą składową. W przypadku większej ilości odbić punkt źródła należy odbić lustrzanie względem każdej kolejnej powierzchni odbijającej. Zbiór wyznaczonych w ten sposób źródeł nazywamy siatką źródeł pozornych, która reprezentuje warunki akustyczne analizowanego pomieszczenia dla ściśle określonych punktów źródła i odbioru. Siatka źródeł pozornych może być podstawą do wyznaczenia echogramu i czasu pogłosu pomieszczenia. Ze względu na dużą złożoność obliczeniową metody często uzyskiwane są wyniki, które nie wystarczają do analizy pola akustycznego. W tym wypadku stosuje się połączenie kilku metod numerycznych [11] [12] [13] [14].

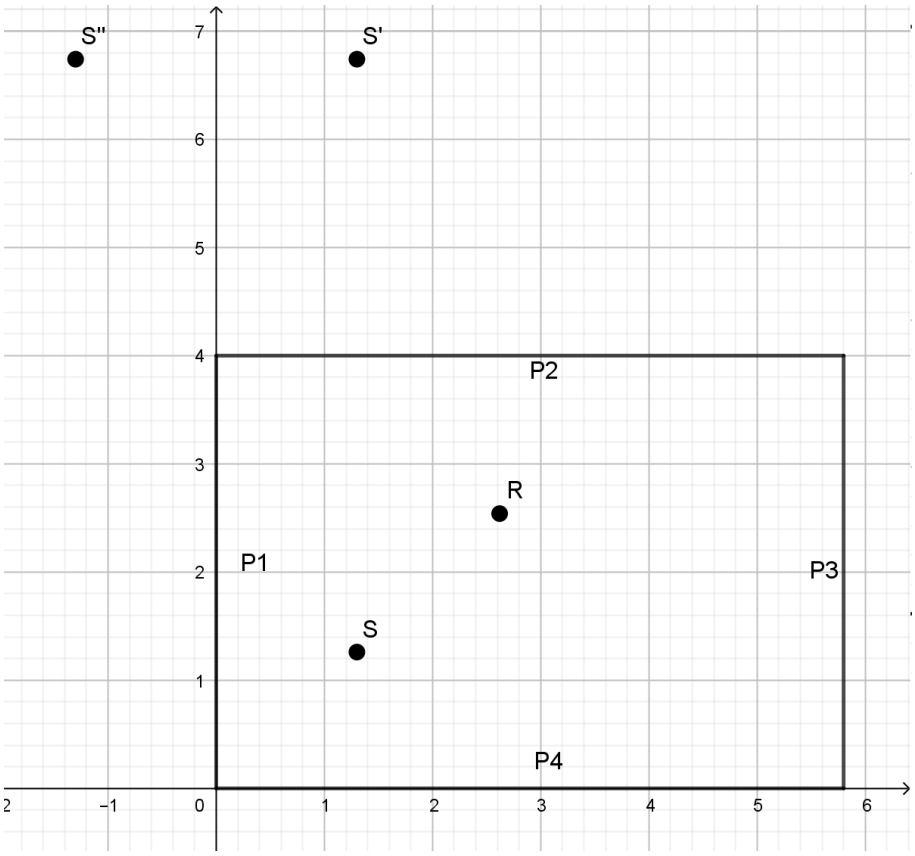
2.3. Wyznaczanie siatki źródeł pozornych

W celu wyznaczenia siatki źródeł pozornych N-tego rzędu zdefiniujmy punkt źródła dźwięku S, punkt odbioru R oraz K-liczny zbiór powierzchni odbijających P_i gdzie i oznacza numer powierzchni (Rys. 2.1.).



Rys. 2.1. Model geometrii pomieszczenia (S - punkt źródła dźwięku, R - punkt obserwacji, Pn - kolejne powierzchnie odbijające).

Do znalezienia wszystkich źródeł pozornych N-tego rzędu należy wygenerować wszystkie K^N wariacje z powtórzeniami zbioru P. Na wstępie możemy pominąć wariacje, w których ta sama powierzchnia jest przynajmniej dwoma kolejnymi elementami ciągu, ponieważ promień dźwiękowy nie może dwukrotnie z rzędu odbić się od tej samej powierzchni. Każda wariacja reprezentuje jeden promień dźwiękowy, który odbija się kolejno od każdej powierzchni w wariacji. Aby znaleźć źródło pozorne dla danego promienia dźwiękowego należy odbić punkt źródła symetrycznie względem każdej z płaszczyzn w wariacji (Rys. 2.2.).



Rys. 2.2. Pierwsze i drugie odbicie względem powierzchni (S - punkt źródła dźwięku, S' i S" - kolejne odbicia symetryczne punktu S względem powierzchni odbijających, R - punkt obserwacji, Pn - kolejne powierzchnie odbijające).

Dla uzyskanego punktu należy zweryfikować, czy promień dźwiękowy jest w stanie dotrzeć od punktu źródła do punktu odbiornika w danym układzie geometrycznym. W tym celu należy prześledzić ścieżkę promienia dźwiękowego wstecz – zaczynając od punktu obserwacji. Początkowy kierunek promienia znajduje się wyznaczając wektor rozpięty od punktu źródła pozornego do punktu obserwacji. Prosta przechodząca przez te punkty powinna przecinać ostatnią powierzchnię z wariacji. Kąt pomiędzy wektorem wyznaczającym kierunek ścieżki promienia dźwiękowego a wektorem normalnym płaszczyzny, na której leży powierzchnia odbijająca, powinien być mniejszy niż 90 stopni. Odcinek łączący punkt obserwacji z punktem przecięcia powyższej prostej z powierzchnią odbijającą stanowi fragment ścieżki promienia dźwiękowego. Kolejny fragment ścieżki promienia dźwiękowego wyznacza się odbijając symetrycznie poprzedni odcinek względem kolejnej powierzchni odbijającej. Z każdym krokiem należy sprawdzić czy odcinek ścieżki promienia dźwiękowego przecina płaszczyzny względem których się odbija. Jeśli uda się prześledzić ścieżkę promienia dźwiękowego od punktu obserwacji do punktu źródła, można sprawdzany punkt uznać za źródło pozorne i uwzględnić w siatce źródeł pozornych.

2.4. Uwzględnienie współczynnika pochłaniania

W akustyce architektonicznej, przy symulacji warunków akustycznych w pomieszczeniu, istotne jest uwzględnienie współczynnika pochłaniania dźwięku (Wzór 2.1.).

$$\alpha = \frac{E_{poch}}{E_{pad}} \tag{2.1}$$

gdzie

E_{poch} – energia fali pochłoniętej
E_{pad} – energia fali padającej

Definiując dla każdej powierzchni odbijającej współczynnik pochłaniania α_i, gdzie i jest indeksem powierzchni, możemy dla każdej z nich wyznaczyć współczynnik odbicia R_i. Mnożąc przez siebie współczynniki odbić fali dla kolejnych powierzchni i przemnażając przez energię źródła otrzymujemy energię źródła pozornego odpowiadającego odbiciom od powyższych powierzchni (Wzór 2.2).

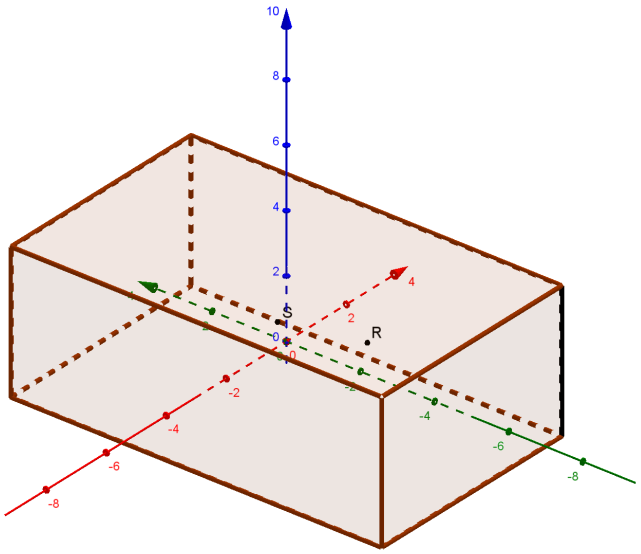
$$E_{IS} = E_S \cdot \prod_{i=1}^{N} (1 - \alpha_i) \tag{2.2}$$

gdzie

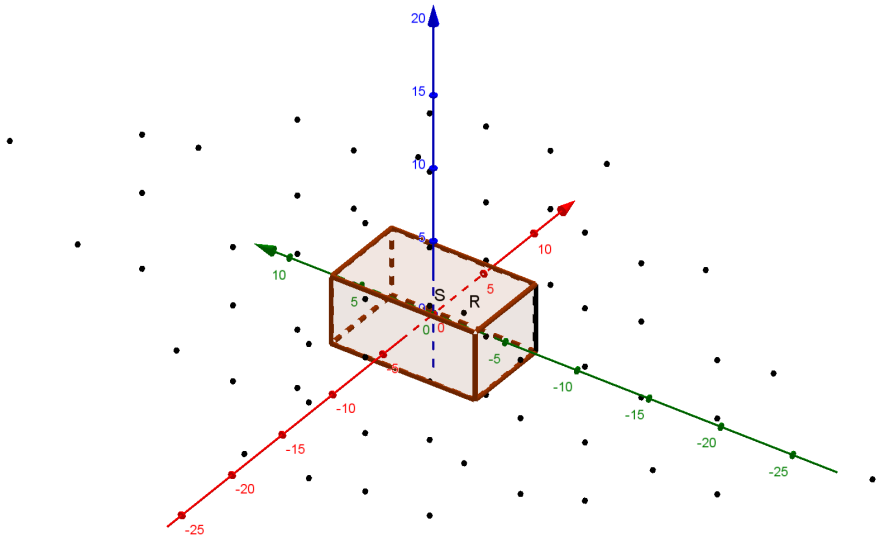
E_S – energia źródła dźwięku
N – liczba powierzchni odbijających
α_i – współczynniki pochłaniania kolejnych powierzchni odbijających

2.5. Przykładowe użycie metody

Przyjmując prostopadłościenne pomieszczenie z umieszczonym punktowym źródłem dźwięku i punktem obserwacji (Rys. 2.3.), wyznaczamy siatkę źródeł pozornych (Rys. 3.4.). Dla przejrzystości rysunku siatka została wyznaczona dla maksymalnie trzeciego odbicia.

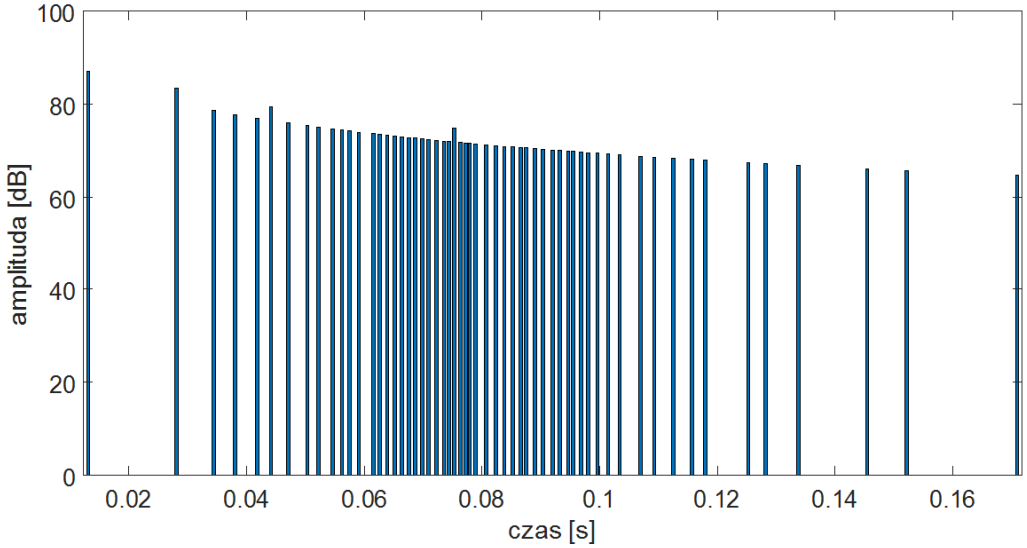


Rys. 2.3. Model geometrii pomieszczenia (S - punkt źródła dźwięku, R - punkt obserwacji).



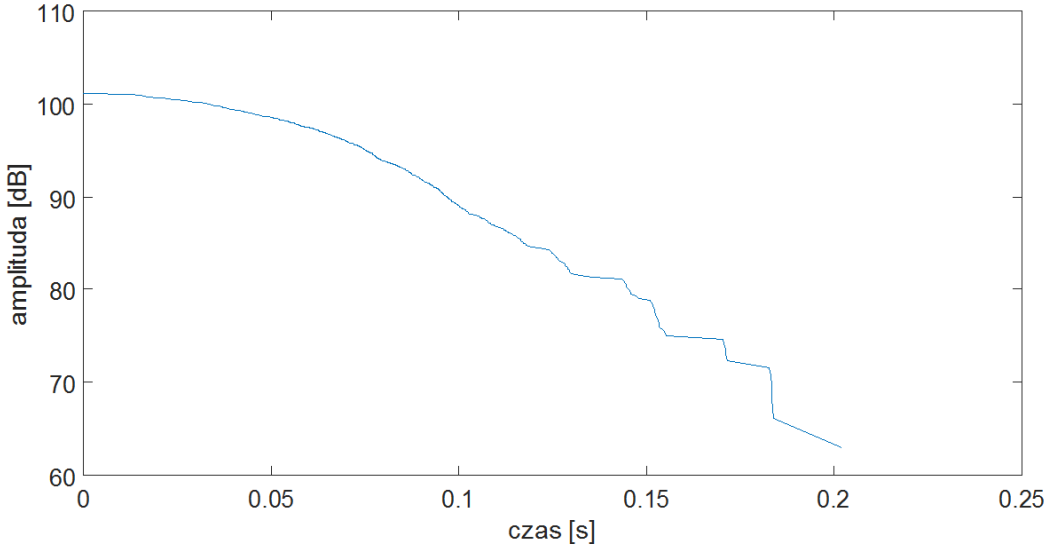
Rys. 2.4. Siatka źródeł pozornych (S - punkt źródła dźwięku, R - punkt obserwacji).

Położenia źródeł pozornych w siatce dają informację o kierunkach promieni dźwiękowych dochodzących do punktu obserwacji. Uwzględniając straty energii pochłoniętej przez odbicia oraz starty energii związanej z rozchodzeniem się fali kulistej możemy wyznaczyć ilość energii i czas, w jakim dotrze ona do punktu obserwacji dla każdego źródła pozornego. Zależność energii dochodzącej do punktu odbioru od czasu przedstawiona jest na poniższym echogramie (Rys. 2.5.).



Rys. 2.5. Echogram.

Uzyskany echogram lub jego część mogą być użyte do obliczenia wskaźników C50, C80, D50. Poprzez całkowanie wsteczne echogramu można uzyskać krzywą zaniku energii dźwięku w pomieszczeniu (Rys. 2.6.).



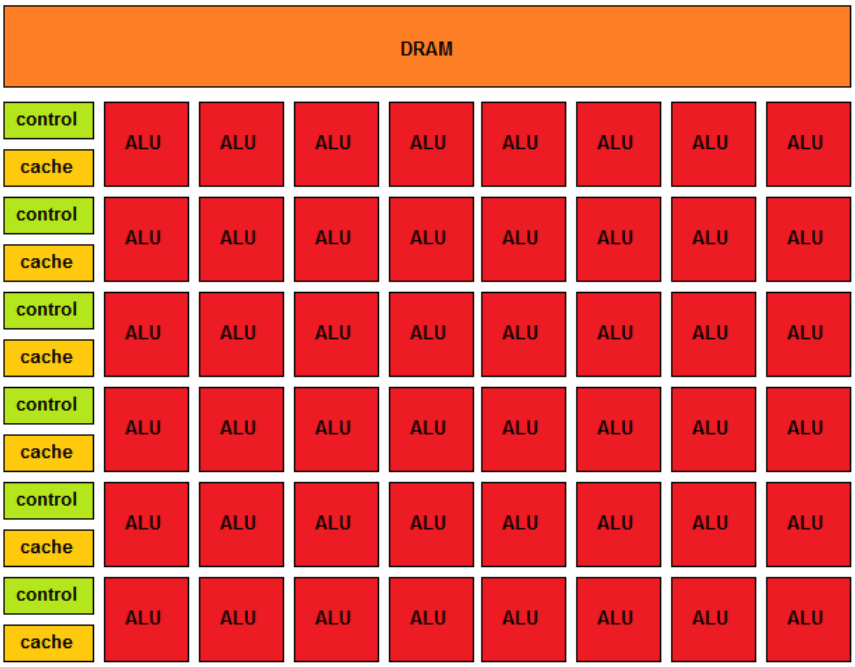
Rys. 2.6. Krzywa zaniku dźwięku.

Na jej podstawie można wyznaczyć czas pogłosu pomieszczenia. W porównaniu do metody źródeł pozornych, w metodzie promieniowej, ze względu na mniejszą złożoność obliczeń, uzyskuje się dłuższe echogramy. Zaletą metody źródeł pozornych jest uzyskiwanie dokładnych ścieżek promieni dźwiękowych, a nie, jak w przypadku metody promieniowej, ścieżek trafiających jedynie w okolice punktu obserwacji. Ze względu na zalety obu tych metod często stosuje się je łącznie [13] [14].

3. Przetwarzanie heterogeniczne

3.1. Wprowadzenie

Lata 50-te XX wieku były przełomowym okresem w dziedzinie elektronicznego przetwarzania danych. Opracowana w 1945 roku *Architekura von Neumana* [15] pozwoliła na uruchomienie pierwszych komputerów ogólnego przeznaczenia. Mimo, że *Architektura Harwardzka* [16] została opracowana 6 lat wcześniej, *Architektura von Neumana* była łatwiejsza w implementacji przez przechowywanie danych wraz z programem na jednej wspólnej pamięci. Pierwszym komputerem opartym na pomyśle Neumana, który wykonywał instrukcje zapisane w fizycznej pamięci, był powstały w 1948 roku Small-Scale Experimental Machine. Stał się on bazą do rozwijania kolejnych urządzeń. W ten sposób w 1949 roku powstał EDSAC (akronim od ang. Electronic Delay Storage Automatic Calculator). Został uznany jako pierwszy komputer wykorzystywany w praktyce do obliczeń naukowych. EDSAC rozbudowany był o dodatkowe układy peryferyjne. W celu odczytu danych zastosowano w nim dalekopis – aparat drukujący dane w postaci alfanumerycznej. Skonstruowanie komputerów zerowej, pierwszej i drugiej generacji znacznie rozwinęło moc obliczeniową tych urządzeń. W dalszym ciągu jednak stosowano niewygodne formy prezentacji danych – wyświetlacze złożone z szeregu lamp, perforowane karty. W 1975 roku, w jednym z pierwszych komputerów osobistych IBM 5100, zastosowano kineskopowy wyświetlacz, który mógł wyświetlać 16 linii po 64 znaków. 6 lat później, w kolejnym modelu IBM 5150, wprowadzono możliwość instalacji kart rozszerzeń ISA. Zastosowano w nim pierwszą kartę graficzną Monochrome Display Adapter (MDA). Rozpoczęło to rozwój peryferyjnych układów komputera, które stały się niezależnymi platformami z własnym procesorem i pamięcią. Początkowo karty graficzne były w stanie wyświetlać jedynie znaki alfanumeryczne przechowywane w pamięci karty. Kolejne generacje kart pozwalały na rysowanie obrazów przy użyciu pojedynczych pikseli, a nowoczesne układy graficzne pozwalały na akceleracje 2D i 3D, korzystając z wbudowanych funkcji do generowania obrazu. W najnowszych procesorach grafiki umożliwiono użytkownikowi zaprogramowanie ich w dowolny sposób. Charakterystyka obliczeń przy przetwarzaniu obrazów wymusiła architekturę procesorów graficznych w postaci dużej ilości jednakowych jednostek ALU (ang. Arithmetic Logic Units), potrafiących wykonać równolegle wiele prostych operacji (Rys. 3.1.).



Rys. 3.1. Architektura procesora GPU (ALU - jednostka arytmetyczno-logiczna, DRAM - pamięć dynamiczna, control - jednostka sterująca, cache - pamięć cache).

Taka budowa kart graficznych pozwoliła na wykorzystanie ich nie tylko do obliczeń związanych z generowaniem grafiki, ale także innych obliczeń przetwarzania danych, co doprowadziło do powstania kart ogólnego przeznaczenia (GPGPU).

3.2. Heterogeniczne platformy obliczeniowe

Początkowo GPGPU były wykorzystywane do zaawansowanego generowania grafiki. Dążenie do realizmu w grach komputerowych rozwinęło karty graficzne o algorytmy wymagające przetwarzania równoległego, takie jak shading [17] czy ray-tracing [3]. W celu odciążenia procesora od złożonych obliczeń, na kartach graficznych zaczęto implementować fizykę obiektów – obliczenia związane z mechaniką klasyczną, symulacje zachowania cieczy, zachowania układów ciągłych i inne efekty cząsteczkowe. Dla ułatwienia programistom wykorzystania możliwości GPGPU, NVIDIA w 2007 roku wprowadza platformę CUDA. Jest to środowisko programistyczne i biblioteka umożliwiająca wykorzystanie kart graficznych produkowanych przez firmę NVIDIA. Umożliwia ona pisanie kodu opartego na C/C++ wykonywanego na procesorze karty i mającego bezpośredni dostęp do jej pamięci. Ułatwienie implementacji algorytmów na GPGPU rozwinęło wykorzystywanie tych kart w różnych dziedzinach naukowych, takich jak kryptografia, fizyka kwantowa, ekonomia czy medycyna. Wykonywanie skomplikowanych obliczeń przy użyciu CUDA stało się powszechne na komputerach domowych, ale było ograniczone przez wymagania sprzętowe. 2 lata po wprowadzeniu CUDA firma Apple Inc wprowadza OpenCL (ang. Open Computing Language). W przeciwieństwie do produktu NVIDA, OpenCL umożliwia pisanie programów na heterogeniczne platformy – układy złożone z różnego rodzaju procesorów. Daje to możliwość pisania aplikacji na komputery z układami większości

popularnych producentów, lub na dowolne układy złożone z różnych procesorów (m. in. CPU, GPU, DSP, FPGA), oraz zapewnia przenośność programów.

3.3. Modele obliczeń równoległych

Obliczenia równoległe to typ obliczeń komputerowych, w których wiele instrukcji wykonywanych jest jednocześnie [14]. Złożone problemy często mogą być rozłożone na mniejsze zadania, które mogą być wykonywane w tym samym czasie. Daje to możliwość szybszego rozwiązywania problemów bez zwiększania częstotliwości taktowania procesora. W uproszczeniu energia wydzielana przez procesor, a w większości ciepło, wzrasta wraz z kwadratem częstotliwości taktowania. Ze względu na te fizyczne ograniczenia procesora wielordzeniowe procesory i obliczenia równoległe stały się dominującym paradygmatem architektury komputerów.

Obliczenia równoległe dzielą się na cztery różne modele:
- równoległość bitów
- równoległość instrukcji
- równoległość danych
- równoległość zadań.

Modele obliczeń równoległych mogą być implementowane oddzielnie lub w połączeniu ze sobą.

3.3.1. Równoległość bitów

Równoległość na poziomie bitów jest modelem przetwarzania równoległego opartym na zwiększaniu ilości bitów w pojedyńczym słowie procesora. Zwiększenie pojemności słowa zmniejsza liczbę instrukcji, które procesor musi wykonać, aby wykonać operację na zmiennych o rozmiarach większych niż długość słowa. Początkowo komputery opierały się na jednobitowych procesorach. Pierwszym nieszeregowym komputerem był 16-bitowy Whirlwind z 1951 roku. Obecnie w komputerach osobistych funkcjonuje standard 32 i 64-bitowych architektur procesorów i systemów operacyjnych. Większa szerokość słowa wymagałaby znacznie większej zewnętrznej szyny danych, co jest kosztowne do wykonania przy obecnych procesach technologicznych. 64-bitowe słowo w większości jest wystarczające do procesowania zmiennych wykorzystywanych na komputerach osobistych. Przy złożonych obliczeniach inżynierskich duże dane przechowuje się na kilku zmiennych i przetwarza przy użyciu kilku instrukcji procesora.

3.3.2. Równoległość instrukcji

Równoległość na poziomie instrukcji jest oparta na równoległym wykorzystaniu kilku jednostek wykonawczych procesora. Większość procesorów składa się z kilku jednostek arytmetyczno-logicznych ALU, odpowiadających za wykonywanie operacji arytmetycznych i logicznych przez procesor oraz kilku jednostek zmiennoprzecinkowych FPU, odpowiadających za wykonywanie operacji arytmetycznych na liczbach zmiennoprzecinkowych. Wszystkie jednostki ALU i FPU mogą wykonywać operacje równolegle. Przykładowo, w poniższym kodzie (Kod ...), operacja z linijki 4 może być wykonana na jednostce ALU, a operacja z linijki 5 na jednostce FPU. Wykorzystując równoległość na poziomie instrukcji obie operacje mogą być wykonane przy użyciu jednej instrukcji procesora.

Kod źr. 1. Plik wejściowy programu

```
1 float a~= 1.0, b = 2.0, c;
2 int x = 1, y = 2, z, w;
3
4 c = a~+ b;
5 z = z * y;
```

W zależności od architektury komputera, rozdzielenie operacji na jednostki wykonawcze procesora może być wykonane dynamicznie w trakcie wykonywania programu lub statycznie na poziomie kompilacji. Dynamicznie ustawiana równoległość instrukcji wykorzystywana jest w powszechnie dostępnych procesorach CPU. Zwiększa ona szybkość wykonywania programu oraz odciąża programistę i kompilator od procesu rozdzielenia instrukcji. Rozmieszczenie instrukcji na etapie kompilacji wykorzystywane jest w procesorach DSP i GPU. Pozwala to optymalizację kodu przed jego wykonaniem i obliczenie dokładnego czasu w jakim program będzie się wykonywał.

3.3.3. Równoległość danych

Równoległość na poziomie danych wykorzystywana jest przy obliczeniach na dużych zbiorach danych, gdzie obliczenia dla pojedyńczych danych są od siebie niezależne. W tym modelu zbiór danych dzielony jest na mniejsze zbiory i definiowany jest zestaw zadań, które zostaną wykonane dla pojedyńczego zbioru. Mniejsze zbiory danych wysyłane są do oddzielnych jednostek sterujących, które wykonują na nich równolegle zdefiniowany zestaw zadań. Równoległość danych jest najwydajniejsza dla dużych zbiorów podobnych danych (np. wektory, macierze), dla prostych operacji takich mnożenie macierzy, transformata Fouriera. Szybkość wykonywania programu opartego na tym modelu zależy od ilości jednostek sterujących, dlatego paralelizm danych często implementowany jest na procesorach graficznych, gdzie ilość tych jednostek sięga kilku tysięcy.

3.3.4. Równoległość zadań

Równoległość na poziomie zadań to model, gdzie równolegle wykonywane są różne zadania na tych samych lub różnych zbiorach danych. Najczęściej wykorzystywany jest na wielordzeniowych procesorach CPU, gdzie każdy rdzeń może wykonywać bardziej złożone operacji niż w przypadku modelu równoległości danych. Rozdzielenie zadań pomiędzy rdzenie wykonywane jest na poziomie pisania kodu aplikacji lub obsługiwane przez system operacyjny. Większość współczesnych systemów operacyjnych wykorzystuje równoległość zadań do wykonywania procesów na oddzielnych wątkach.

3.4. Błędy obliczeniowe

3.4.1. Błędy obliczeń numerycznych

Wykorzystując do obliczeń cyfrowe układy należy mieć na uwadze, że niemal wszystkie obliczenia numeryczne są obarczone błędem. Źródłami tych błędów dla metod numerycznych są:

- błędy wejściowe,

- błędy obcięcia,

- błędy zaokrągleń.

Błędy wejściowe związane są błędami danych wejściowych do algorytmu obliczeniowego. Jeśli dana wejściowa jest wynikiem pomiaru wielkości fizycznej, to jest obarczona błędem tego pomiaru. Dodatkowo zapisanie jej w formacie cyfrowym wprowadza błąd wstępnego zaokrąglenia lub urwania. Wykorzystywane w obliczeniach stałe, które są niewymierne, również są obarczone błędem wstępnego zaokrąglenia lub urwania. Błąd zaokrąglenia jest mniejszy od błędu urwania, ale nie zawsze zaokrąglenie jest możliwe do wykonania. Przy pomiarach czasu rzeczywistego niektóre przetworniki analogowo-cyfrowe są w stanie przybliżyć wartość pomiary jedynie przez operację urwania.

Błędy obcięcia występują przy uproszczeniach formuł matematycznych wykorzystywanych w algorytmie metody numerycznej lub w obliczaniu wartości funkcji i stałych. W środowisku cyfrowym większość stałych niewymiernych wyznaczana jest przy użyciu nieskończonych szeregów. Uwzględniając w obliczeniach skończoną ilość wyrazów szeregu uzyskujemy przybliżoną wartość stałej, a jej błąd uzależniony jest od ilości obliczonych wyrazów.

Błędy zaokrągleń powstają podczas zaokrąglania wyników wykonywanych operacji arytmetycznych. Każdy kolejny wynik zapisywany jest z pewnym zaokrągleniem, więc każda kolejna operacja wprowadza coraz większy błąd. Wartość tego błędu można zmienić w zależności od liczby wykonywanych lub przy zmianie kolejności działań.

3.4.2. Cyfrowa reprezentacja liczb

Technologia w jakiej wykonywana jest fizyczna pamięć komputerów wymusza do przechowywania liczb i danych stosowanie systemu binarnego. O ile liczby całkowite do pewnego zakresu mogą być przechowywane z dokładną wartością, o tyle wiele liczb niecałkowitych wymaga zaokrąglenia ze względu na ograniczoną szerokość słowa bitowego. Nawet liczba mająca skończone rozwinięcie po przecinku w systemie dziesiętnym może mieć nieskończone rozwinięcie w systemie binarnym. Dla zrównoważenia wielkości zakresu i precyzji reprezentacji liczb rzeczywistych często używa się typu liczb zmiennoprzecinkowych. Obecnie najczęściej używanym standardem jest IEEE 754 (Wzór 3.1).

$$-1^{s} \cdot m \cdot 2^{e} \tag{3.1}$$

gdzie

s – znak liczby (0 - dla liczby ujemnej, 1 - dla liczby dodatniej)
m – mantysa liczby
e – wykładnik, który odpowiada za pozycję przecinka

Powyższy standard definiuje trzy typy danych:

- liczba pojedynczej prezycji (ang. signle precision),

- liczba podwójnej precyzji (ang. double precision),

- liczba o rozszerzonej precyzji (ang. extended precision).

W zależności szerokości bitowej danego typu liczba zmiennoprzecinkowa ma różny zakres i precyzję (Tabela 3.1).

Tab. 3.1. Typy danych w standardzie IEEE 754.

precyzja	szerokość	mantysa	wykładnik
pojedyncza	32 bitów	23 bity	8 bitów
podwójna	64 bity	52 bity	11 bitów
rozszerzona	80 bitów	64 bity	15 bitów

Zakres danych możliwy do zapisania na poszczególnych typach obrazuje Tabela 3.2, w której przedstawione są najmniejsze i największe dodatnie liczby możliwe do zapisania na tych typach.

Tab. 3.2. Przybliżone wartości minimalne i maksymalne dodatnich liczb zmiennoprzecinkowych standardu IEEE 754.

precyzja	w. min.	w. maks.
pojedyncza	1,18 E-38	3,40 E+38
podwójna	2,23 E-308	1,90 E+308
rozszerzona	3,65 E-4951	1,18 E+104932

W standardzie IEEE 754 zostały zdefiniowane następujące metody zaokrąglania liczb:

- Zaokrąglanie do najbliższej wartości (ang. round to nearest),
- zaokrąglanie w kierunku zera (ang. round toward 0),
- zaokrąglanie w stronę dodatniej nieskończoności (ang. round toward +infinity),
- zaokrąglanie w stronę ujemnej nieskończoności (ang. round toward -infinity).

Standard wykorzystuje zaokrąglanie do najbliższej wartośc jako domyślną metodę. W zależności od typu, zaokrąglenie może dać różny wynik. W zależności od procesora lub kompilatora operacje arytmetyczne mogą być wykonywane przy użyciu różnych funkcji procesora, które operują na różnych typach danych. Mimo obliczeń na zmiennych o jednolitym typie danych, wynik może się różnić w zależności od platformy sprzętowej.

Błąd zaokrąglenia na maszynie cyfrowej zdefiniowany jest przy użyciu dokładnści maszynowej ϵ. Dokładność maszynowa to różnica pomiędzy 1 a najbliższą jej wartością większą od niej. Zaokrąglenie liczby rzeczywistej do zmiennoprzecinkowej następuje z błędem względnym $\frac{\epsilon}{2}$. Wartości błędu względnego dla poszczególnych typów danych przedstawia Tabela 3.3.

Tab. 3.3. Wartości błędu względnego ϵ zaokrąglenia dla liczb zmiennoprzecinkowych standardu IEEE 754.

precyzja	ϵ
pojedyncza	2 E-23
podwójna	2 E-52
rozszerzona	2 E-64

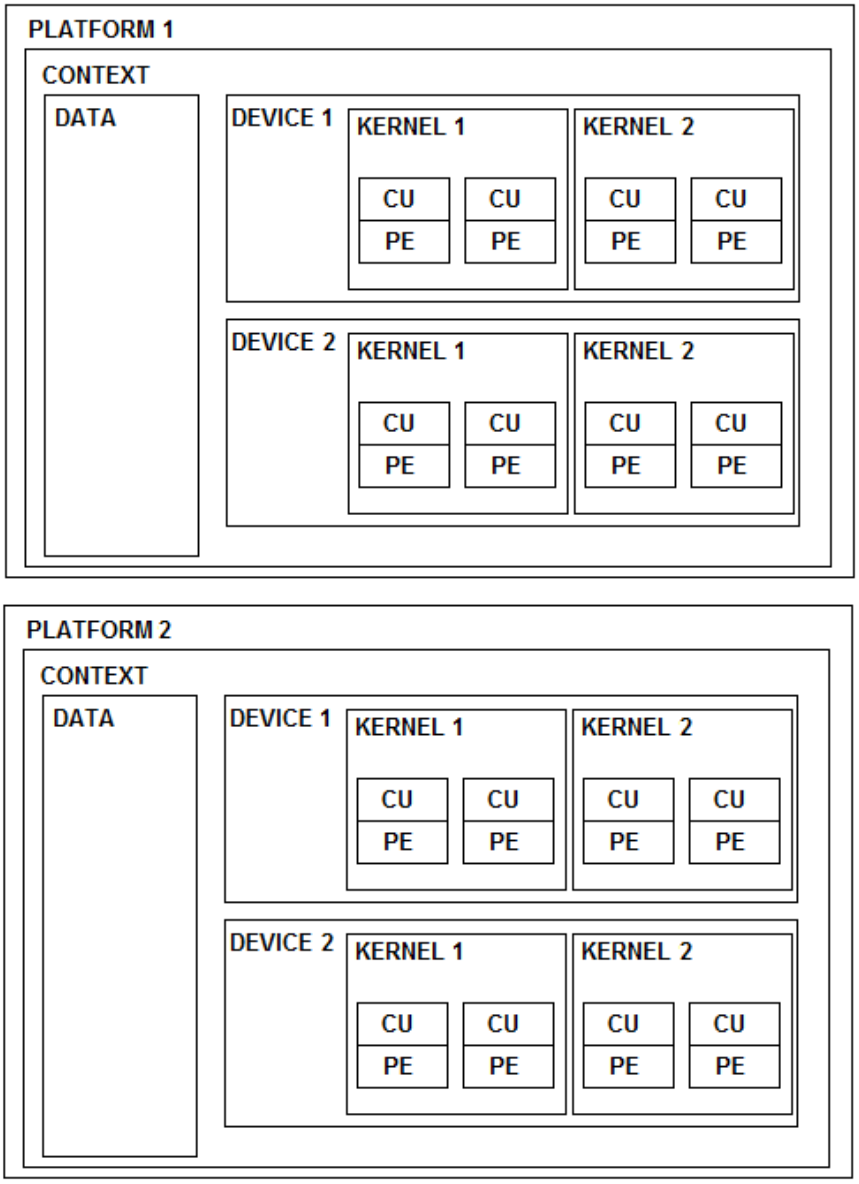
4. Środowisko OpenCL

4.1. Wprowadzenie

OpenCL [18] jest platformą programistyczną opracowaną w 2009 roku przez Apple Inc, a następnie utrzymywaną przez Khronos Group. Umożliwia programowanie heterogenicznych układów procesorowych w języku opartym na C99 i C++11. Wykorzystanie OpenCL nie ogranicza się jedynie do procesorów graficznych. Jego główną zaletą jest przenośność na większość popularnych procesorów CPU, DSP, a nawet układów FPGA. Każdy procesor z instrukcjami SSE2 może zostać wykorzystany jako urządzenie OpenCL. Standard OpenCL dostarcza interfejs programistyczny (API), biblioteki dla wielu popularnych języków programowania oraz sterowniki do urządzeń.

4.2. Platforma

Aplikacja pisana w OpenCL opiera się na jednostkach zwanych platformami. Każda platforma odpowiada za przygotowanie danych do obliczeń i rozdzielenie zadań. W nowszych standardach kod platformy może być pisany w wielu popularnych językach (m. in. Python, Matlab). Każda platforma zawiera kontekst, w którym definiowane są dane wejściowe, kolejka zadań i urządzenia do wykonywania obliczeń. Każde urządzenie zawiera jednostki obliczeniowe (ang. compute units, CU), które składają się z elementów przetwarzania (ang. processing elements, PEs)(Rys. 4.1.).



Rys. 4.1. Architektura programu napisanego w bibliotece OpenCL (CU - jednostka obliczeniowa, PE - element przetwarzania).

Pojedyńcza platforma odnosi się do poszczególnego sterownika. Przykładowo na komputerze z procesorem Intel i kartą graficzną AMD mozemy uruchomić dwie platformy, jedną dla OpenCL™ Runtimes for Intel® Processors [19] i drugą dla wyspecjalizowanego sterownika AMD dla danej karty graficznej. Każda platforma może obsługiwać więcej niż jedno urządzenie, w szczególnych przypadkach może zawierać złożone układy kart graficznych.

Standard OpenCL definiuje obsługę mieszanych wersji platform (ang. Platform Mixed Version Support). Oznacza to, że w jednej aplikacji możemy użyć różnych wersji platform, urządzeń i języka OpenCL. Domyślnie standard używa najnowszej wersji języka dostępnej na danym urządzeniu.

4.3. Model wykonywania programu

Wykonywanie programu OpenCL następuje w dwóch krokach: uruchomienie kodu głównego na urządzeniu gospodarza i wykonywanie kodu kernela na urządzeniach OpenCL. Kod gospodarza definiuje platformy i zarządza wykonywaniem kerneli. Dla każdego kernela definiowana jest przestrzeń indeksowania. Każdy element przestrzeni indeksowania jest unikalnym identyfikatorem i dla każdego z nich wykonywany jest kod kernela. Instancję kernela wywołaną w obrębie jednego indeksu nazywamy wątkiem (ang work-item). Każdy wątek wykonuje ten sam kod, ale jego działanie może zależeć od przydzielonego mu indeksu. Posługując się identyfikatorem z przestrzeni indeksowej możemy dla każdego wątku wybrać inny zakres danych do obliczeń lub wykonać inny typ obliczeń. Ta własność, w połączeniu z możliwością wykonywania różncych kerneli w jednej aplikacji, umożliwia pisanie programów zarówno w modelu równoległości danych, jak i modelu równoległości zadań.

4.4. Przestrzeń indeksowania

Przestrzeń indeksowania w standardzie OpenCL definiowana jest przez typ NDRange. Jest to jedno-, dwu- lub trójwymiarowa tablica liczb całkowitych. Cała przestrzeń dzieli się na grupy o takim samym wymiarze jak wymiar całej przestrzeni (ang. work-groups). Każdy wątek może być zidentyfikowany po jego globalnym indeksie lub po kombinacji indeksu grupy i lokalnego indeksu wewnątrz grupy. Wątki wewnątrz jednej grupy wykonywane są na jednym elemencie przetwarzania. Podział na grupy może być zrealizowany ze względu na ilość jednostek przetwarzania lub charakter obliczeń. Przestrzeń ideksowania stanowią rosnące wartości liczb całkowitych z zadanym przesunięciem, więc może ona reprezentować indeksy wektora, macierzy lub dziedzinę funkcji.

4.5. Kontekst obliczeniowy

Definiowany na urządzeniu gospodarza na pojedyńczej platformie kontekst obliczeniowy odpowiada za przygotowanie następujących zasobów:

- Urządzenia OpenCL - zbiór urządzeń podpiętych pod urządzenie gospodarza, na których wykonywane są kernele.

- Kernele - funkcje napisane w języku OpenCL, które wykonywane są na poszczególnych urządzeniach.

- Obiekty programu - źródła i pliki wykonywalne programu gospodarza.

- Obiekty pamięci - przestrzeń pamięci na urządzeniu gospodarza, w której przechowywane są dane wejściowe i wyjściowe wykonywania kernela.

Kontekst pisany jest zgodnie z interfejsem OpenCL, który udostępnia kolejkę zadań (ang. command-queue). Kolejka zadań zawiera listę zadań do wykonania, które postępują asynchronicznie względem urządzenia gospodarza. Zadaniami mogą być:

- Zadanie wykonania kernela - wykonanie kernela na konkretnym elemencie przetwarzania na konkretnym urządzeniu.

- Zadanie pamięci - przeniesienie danych pomiędzy pamięcią gospodarza, a pamięcią urządzenia, lub przekazanie wskaźnika na adres pamięci gospodarza do urządzenia.

- Zadanie synchronizacji - wymuszenie kolejności wykonywania zadań z kolejki.

Wykonywanie zadań z kolejki może być wykonywane w kolejności (ang. In-order execution) lub poza kolejnością (ang. Out-order execution). W pierwszym wariancie kolejne zadanie z kolejki może być wykonywane dopiero po zakończeniu poprzedniego zadania. Zadanie wykonywane poza kolejnością rozpoczyna działanie przed zakończeniem poprzedniego zadania. Synchronizacja pomiędzy zadaniami jest wykonywana przez programistę wykorzystując komendy synchronizacyjne z interfejsu OpenCL.

W jednym kontekście może być zdefiniowana więcej niż jedna kolejka. Takie kolejki wykonywane są jednocześnie i niezależnie. Standard OpenCL nie udostępnia mechanizmu dedykowanego do synchronizacji pomiędzy kolejkami.

4.6. Kernel

Funkcja kernela jest kodem napisanym w języku OpenCL wykonywanym na urządzeniu OpenCL. Zdefiniowa jest poprzez kwalifikator `__kernel`. Język kernela oparty jest na C99 i C++11 i w zależności od wersji OpenCL może wykorzystywać niektóre funkcje biblioteczne z tych języków. Przykładowo od wersji OpenCL 1.2 w kodzie kernela możliwe jest użycie funkcji printf.

Rozróżniane są dwa typy kerneli:

- Kernele OpenCL - napisane w języku OpenCL C i skompilowane przy użyciu kompilatora OpenCL.

- Natywne kernele - funkcje skompilowane przez inny kompilator niż OpenCL. Najczęściej są to funkcje wyeksportowane z zewnętrznych bibliotek.

Kod kernela kompilowany jest do obiektu kernela podczas wykonywania kodu głównego na urządzeniu gospodarza. Zbudowanie obiektu kernela wytwarza program oparty na jego kodzie i powiązany z przypisanymi mu danymi wejściowymi. Dane wejściowe mogą być przekazane do kernela w postaci wskaźnika na adres pamięci urządzenia gospodarza lub przeniesione do lokalnej pamięci urządzenia. Formy przekazywania pamięci i jej typy przedstawione są w rozdziale 4.9.

4.7. Urządzenia OpenCL

Urządzenie OpenCL stanowi interfejs do zbioru jednostek obliczeniowych. W generalnym przypadku są to karty GPU, wielordzeniowe procesory CPU, DSP i procesory typu Cell/B.E.

Standard OpenCL udostępnia 5 różnych rodzajów urządzeń:

- `CL_DEVICE_TYPE_CPU` - urządzenie gospodarza na którym, oprócz kodu głównego, mogą być wykonywane kernele.

- `CL_DEVICE_TYPE_GPU` - urządzenie GPU podłączone pod urządzenie gospodarza.

- `CL_DEVICE_TYPE_ACCELERATOR` - dowolne urządzenie peryferyjne wspierane przez standard OpenCL.

- `CL_DEVICE_TYPE_DEFAULT` - dowolne urządzenie dostępne na danej platformie.

- `CL_DEVICE_TYPE_ALL` - wszystkie dostępne urządzenia dostępne na danej platformie.

Przy użyciu typu urządzenia możemy pobrać uchwyty do wybranych urządzeń dostępnych na platformie. Pobrane urządzenia mogą być przekazane do kontekstu gdzie definiowane jest ich użycie. Platforma OpenCL umożliwia pobranie informacji o urządzeniu takich jak typ urządzenia czy ilość jednostek obliczeniowych i elementów przetwarzania. Pozwala to na planowanie podziału danych, grup i przestrzeni indeksowania na poziomie pisania kodu i dynamiczne przydzielanie tych wartości w czasie wykonywania programu w zależności od urządzeń, na jakich jest on wykonywany.

4.8. Jednostki obliczeniowe i elementy przetwarzania

Jednostki obliczeniowe (ang, Compute Unit, CU) i elementy przetwarzania (ang. Processing Element, PE) reprezentują różne fizyczne elementy procesora w zależności od jego rodzaju. Dla procesorów CPU CU jest równoważne PE i stanowi pojedyńczy rdzeń procesora. W kartach graficznych NVIDIA CU stanowi multiprocesor strumieniowy, a PE jego rdzenie. Karty graficzne AMD definiują CU jako jednostkę SIMD (ang. Single Instruction, Multiple Thread), a PE jako poszczególne linie jednostki SIMD. Na pojedyńczym elemencie przetwarzania wykonywany jest jeden wątek. Mimo że w zależności od urządzenia CU i PE reprezentują różne elementy procesora, odwołują się do nich w ten sam sposób. Umożliwia to wykonywanie programu gospodarza i kernela o jednakowej składni na różnych typach urządzeń. Kernele wykonywane są według kolejki zadań i rozdzielone pomiędzy jednostki obliczeniowe mogą być wykonywane równolegle na wielu elementach przetwarzania.

4.9. Dostep do pamieci

OpenCL umożliwia dynamiczny dostęp do pamięci urządzeń oraz pamięci gospodarza. Każdy wątek ma dostęp do czterech typów pamięci:

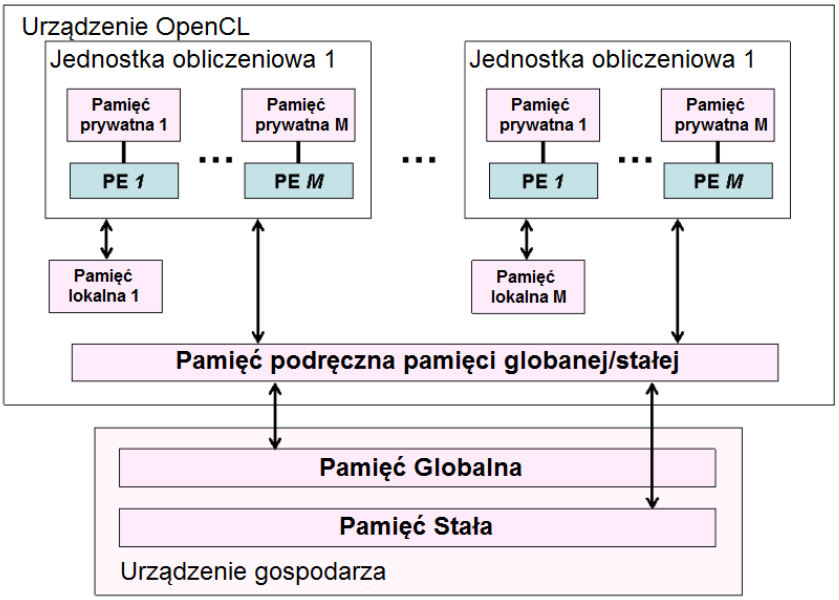
- Pamięci globalnej – pamięci dostępnej przez wszystkie wątki w obrębie jednego kontekstu.

- Pamięć stałej – pamięci tylko do odczytu, dostępnej globalnie.

- Pamięci lokalnej – pamięci dostępnej w obrębie jednej grupy wątków.

- Pamięci prywatnej – pamięci dostępnej w obrębie jednego wątku.

Przepływ danych pomiędzy urządzeniem gospodarza a urządzeniem OpenCL może odbyć się poprzez operację kopiowania lub odwzorowania z pamięci platformy do pamięci urządzenia. Rozmieszczenie typów pamięci i ich wzajemne relacje przedstawione są na Rysunku 4.2.



Rys. 4.2. Architektura pamięciu na urządzeniach OpenCL (PE - jednostka przetwarzania).

Operacja kopiowania danych pomiędzy pamięcią gopodarza a pamięcią urządzenia jest wywoływana przez zadanie przeniesienia danych znajdujące się w kolejce zadań. Zadanie przeniesienia może być blokujące lub nieblokujące. Przeniesienie blokujące zabrania dostępu do pamięci urządzenia gospodarza aż do czasu zakończenia przenoszenia danych. Dostęp do pamięci urządzenia gospodarza przez wskaźnik na adres pamięci umożliwia modyfikację danych w tej pamięci przez wiele kerneli jednocześnie. Może to być wykorzystywane przy operacjach na wspólnym zbiorze danych takich jak funkcja splotu czy mnożenie macierzy.

Standard OpenCL opiera się na luźnym modelu spójności pamięci (ang. relaxed consistency memory model). Oznacza to, że stan pamięci widoczny przez poszczególne wątki nie jest spójny w każdej chwili. OpenCL gwarantuje spójność pamięci jedynie wewnątrz grupy. Spójność pamięci pomiędzy różnymi grupami musi być obsłużona poprzez punkty synchronizacji.

Dużym ograniczeniem wydajności obliczeń jest czas dostępu do pamięci. Pamięć o szerszym zakresie ma wolniejszy czas dostępu, dlatego najwydajniejsze obliczenia wykonują się dla niezależnych wątków działających równolegle na prywatnej pamięci.

4.10. Synchronizacja

Istnieją dwie domeny synchronizacji w OpenCL:
- pomiędzy wątkami w obrębie jednej grupy,
- pomiędzy zadaniami z kolejki lub kolejek zadań w obrębie jednego kontekstu.

Synchronizacja pomiędzy wątkami w obrębie jednej grupy jest wykonywana przy użyciu bariery grupy (ang. work-group barrier). Wszystkie wątki z danej grupy muszą dojść do punktu bariery grupy zanim zacznie się wykonywanie programu poza barierą. Poprawne wykonanie programu jest możliwe tylko gdy wszystkie lub żaden wątek nie

osiągnie bariery grupy. Standard nie udostępnia mechanizmu synchronizacji pomiędzy grupami.

Synchronizacja w obrębie kolejek zadań może odbywać przy użyciu dwóch mechanizmów:

- Bariera kolejki zadań. Bariera ta zapewnia, że wszystkie znajdujące się przed nią zadania w kolejce zostaną zakończone. Wszystkie obiekty pamięci, które są wykorzystywane przez późniejsze zadania, muszą zostać ustalone zanim zaczną się wykonywać zadania znajdujące się w kolejce po barierze. Bariera odnosi się tylko do jednej kolejki zadań i nie może być użyta do synchronizacji pomiędzy kolejkami.

- Oczekiwanie na wydarzenie. Każda funkcja z interfejsu OpenCL, która kolejkuje zadania, zwraca wydarzenie identyfikujące to zadanie i wszystkie obiekty pamięci do których się odnosi. Dodatkowe zadanie w kolejce może czekać na wybrane wydarzenia z różnych zadań i stanowić punkt synchronizacji. Ten mechanizm może być użyty do synchronizacji pomiędzy kilkoma kolejkami.

4.11. Obiekty pamięci

Obiekty pamięci przechowują dane, które mogą być przenoszone pomiędzy pamięcią urządzenia gospodarza a pamięcią urządzenia OpenCL. Obiekt pamięci jest obsługiwany przez typ `cl_mem` i może być przekazany jako parametr wejściowy lub wyjściowy do funkcji kernela.

Standard definiuje dwa typy obiektów pamięci: bufor (ang. buffer object) i obraz (ang. image object). Obiekt bufora przechowuje jednowymiarowy zbiór elementów, natomiast obiekt obrazu może przechowywać dwu- lub trójwymiarowy zbiór elementów albo zbiór buforów. Podstawową różnicą między typami poza wymiarem jest ułożenie danych w pamięci. Elementy w buforze są ułożone w ciągłej pamięci i możemy się odwoływać do nich poprzez wskaźnik. Rozmieszczenie elementów obrazu nie jest widoczne dla użytkownika, a dostęp do nich odbywa się poprzez funkcje wbudowane w język OpenCL C. Elementami bufora mogą być dowolne typy danych obsługiwane przez język kernela, natomiast elementy obrazu stanowią czteroelementowe wektory liczb całkowitych lub zmiennoprzecinkowych. Stanowi to wygodny format do przechowywania plików graficznych. W przypadku przekazywania innego typu danych jako obraz, w funkcji kernela należy wykonać odpowiednią konwersję i interpretację danych.

4.12. Inne środowiska programowania heterogenicznego

Wzrost popularności wykorzystania kart graficznych do ogólnego użytku doprowadził do powstania wielu środowisk wykorzystujących GPGPU. OpenCL jako jedyny wspiera tak dużą ilość języków i urządzeń. Umożliwia to wykorzystanie OpenCL do pisania aplikacji, możliwych do uruchomienia zarówno na urządzeniach domowych jak i specjalistycznych. Wysoki poziom abstrakcji tego środowiska pozwala na pewien uniwersalizm, jednak ogranicza to niskopoziomowy dostęp do urządzeń i możliwość optymalizacji na wybrane urządzenia. Obecnie dostępne są podobne środowiska programistyczne, takie jak The Xcelerit SDK [20], DirectCompute GPU computing API [21], Alea GPU [22], Close to Metal [23] czy C++ AMP [24]. Popularne w środowisku naukowym oprogramowanie MATLAB również wspiera obliczenia na GPGPU przy użyciu biblioteki Parallel

Computing Toolbox and MATLAB Distributed Computing Server. Powyższe środowiska w większości są przeznaczone na tylko jedną platformę sprzętową, mają ograniczony interfejs do wybranego zbioru operacji lub wykorzystują specyficzny język programowania. Wybór środowiska może ograniczać użycie aplikacji, ale również zwiększyć jej wydajność.

Obok OpenCL dominującym na rynku środowiskiem jest CUDA utworzone przez Nvidia. Jest to pierwsze i najdłużej rozwijane środowisko wspierające programowanie na GPGPU. Architektura programów pisanych w CUDA jest znacząco podobna do OpenCL - istnieje wiele odpowiedników poszczególnych struktur, funkcji i typów zmiennych. Firma Nvidia produkuje dedykowane karty graficzne dla technologii CUDA co pozwala na lepszą optymalizację kodu na dane urządzenia. W wielu przypadkach technologia CUDA i OpenCL pozwala na uzyskanie zliżonej wydajności [25]. Jedynie przy użyciu wyspecjalizowanych kart CUDA pozwala na uzyskanie większej wydajności niż w konkurencyjnych środowiskach. Znacznym ograniczeniem CUDA jest możliwość obliczeń jedynie na urządzeniach Nvidia. Umożliwia to pisanie aplikacji tylko na wyspecjalizowane platformy i nie zapewnia takiej przenośności jak w przypadku OpenCL.

5. Realizacja założeń projektowych

5.1. Wprowadzenie

Promowany w latach 80-tych ruch wolnego oprogramowania [26] znacząco wpłynął na rozwój aplikacji i algorytmów. Powszechny dostęp do Internetu umożliwił upublicznianie kodów źródłowych, co zrzeszało duże społeczności pracujące wspólnie nad rozwojem programów komputerowych. Otwarte bazy aplikacji i algorytmów pozwoliły niezależnym programistom na wykorzystanie ich do rozwoju kolejnych w wydajniejszy sposób. Realizacja projektu niniejszej pracy stanowi bazę, która może zostać wykorzystana w podobny sposób. Kod tej aplikacji może zostać łatwo zaadaptowany do połączenia jej z innym oprogramowaniem. Możliwe też jest wprowadzenie modyfikacji algorytmu metody źródeł pozornych lub zaimplementowanie dodatkowych metod wewnątrz aplikacji.

Aplikacja została napisana przy użyciu OpenCL w wersji 1.2. Zdecydowano się na starszą wersję ze względu na kompatybilność z wieloma popularnymi urządzeniami. Platforma została napisana przy użyciu języka C++11. Zarówno dane wejściowe jak i wyjściowe transferowane są między plikiem wykonywalnym a plikami tekstowymi. Dane przechowywane są w zmiennych o pojedynczej precyzji. Zmniejsza to czas obliczeń na urządzeniach o architekturze 32-bitowej i wprowadza wystarczająco mały błąd dla tego typu obliczeń.

5.2. Implementacja algorytmu

Aplikacja do obliczeń używa jednej platformy (Kod źr. 2.), która definiuje jeden kontekst obliczeniowy i jedno urządzenie.

Kod źr. 2. Definiowanie platformy

```
1 cl_platform_id* platforms = new cl_platform_id[numPlatforms];
2 status = clGetPlatformIDs(numPlatforms, platforms, NULL);
```

Dane wejściowe alokowane są w pamięci po stronie platformy. Następnie tworzony jest kontekst obliczeniowy (Kod źr. 3.), który przy użyciu predyrektywy `CL_PRESENT_DEVICE` wybiera , jaki typ urządzenia zostanie wybrany.

Kod źr. 3. Definiowanie kontekstu obliczeniowego

```
cl_context_properties cps[3] = { CL_CONTEXT_PLATFORM,
   (cl_context_properties)platform, 0 };
context = clCreateContextFromType(
                                    cps,
                                    CL_PRESENT_DEVICE,
                                    NULL,
                                    NULL,
                                    &status);
```

Następuje poszukiwanie urządzenia o zadanym typie i zapisanie informacji o nim (Kod źr. 4.).

Kod źr. 4. Pobieranie listy dostępnych urządzeń

```
clGetContextInfo(
                  context,
                  CL_CONTEXT_DEVICES,
                  deviceListSize,
                  devices,
                  NULL);
```

Kolejnym krokiem jest przygotowanie kolejki zadań (Kod źr. 5.). Posłuży ona do stworzenia przestrzeni indeksowania dla wątków.

Kod źr. 5. Przygotowanie kolejki zadań

```
commandQueue = clCreateCommandQueue(
                                     context,
                                     devices[0],
                                     0,
                                     &status);
```

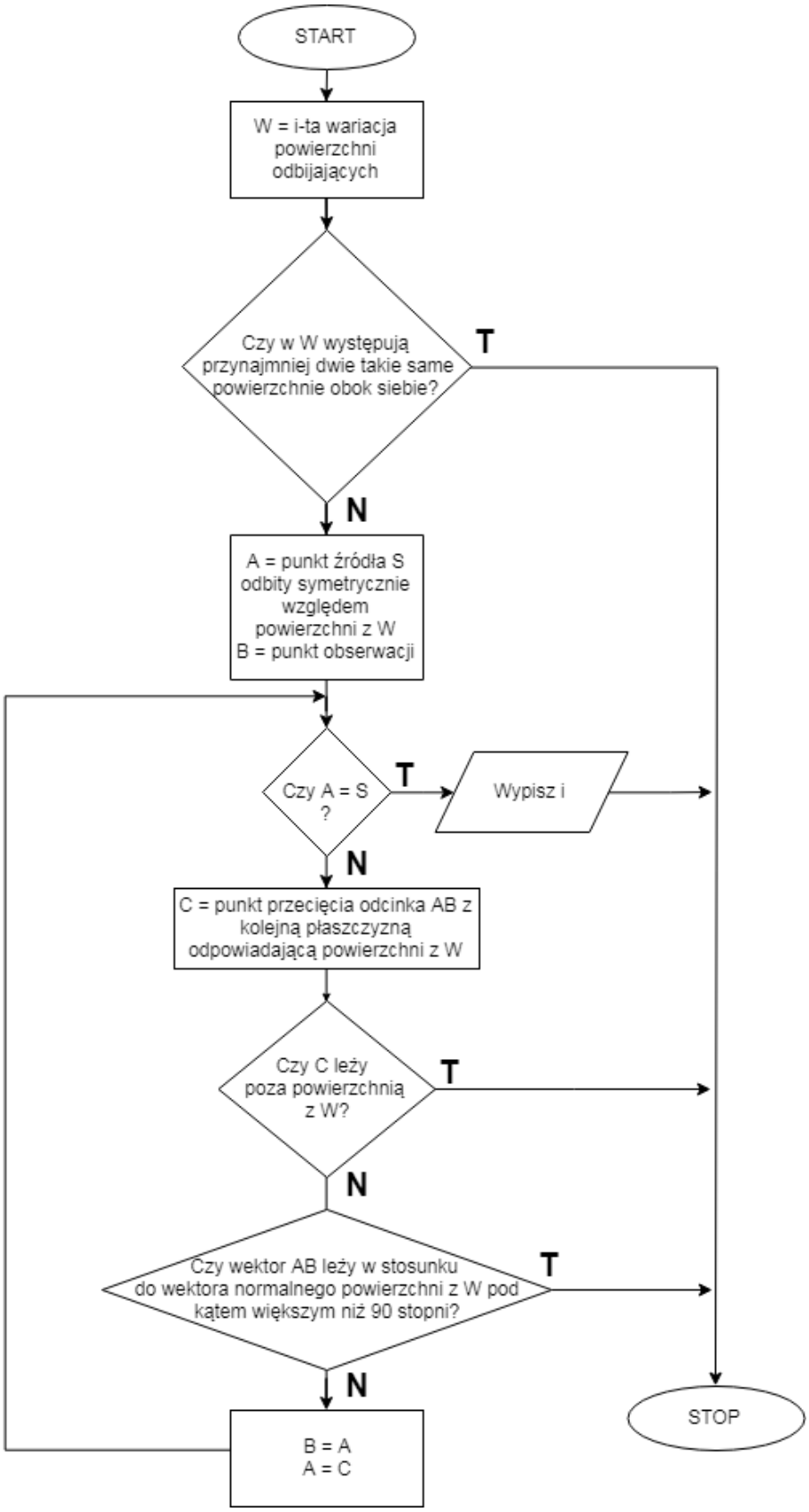
Dane wejściowe przekazywane są do kontekstu obliczeniowego poprzez bufory (Kod źr. 6.). Oddzielne bufory dla każdej danej wejściowej transferują owe dane do zdefiniowanego przez kernel miejsca w pamięci.

Kod źr. 6. Definicja buforu danych dla parametru plA

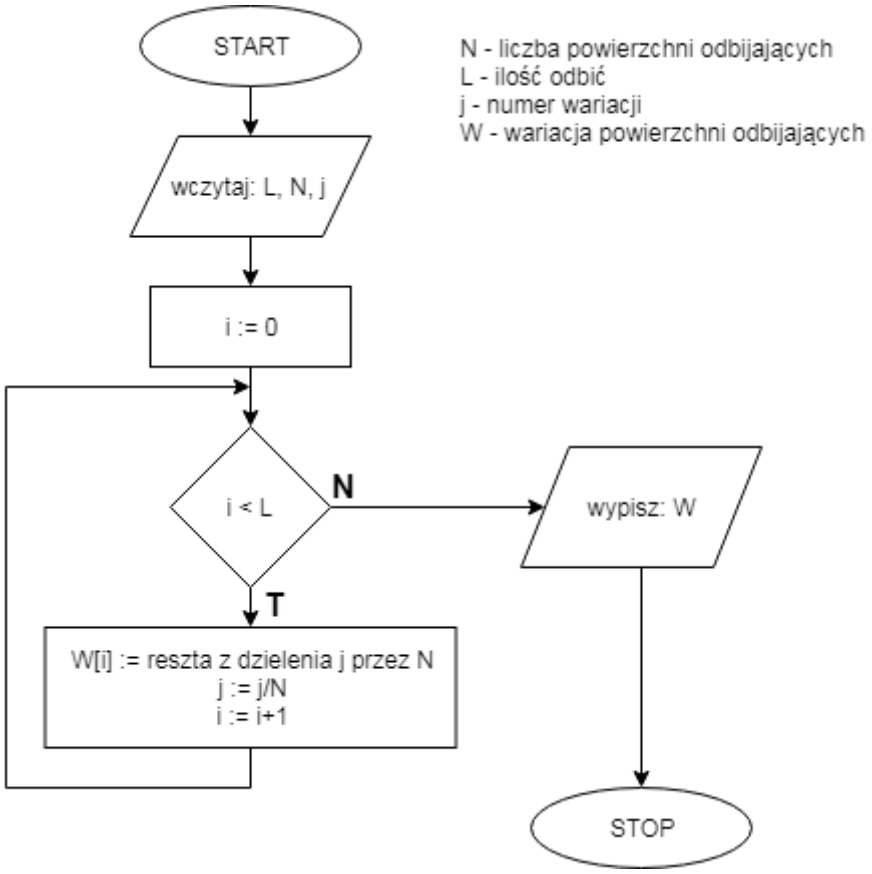
```

plABuffer = clCreateBuffer(
                            context,
                            CL_MEM_READ_WRITE | CL_MEM_USE_HOST_PTR,
                            sizeof(cl_float) * walls,
                            plA,
                            &status);
```

Kod kernela przechowywany jest w postaci oddzielnego pliku tekstowego (załącznik A). Jest on kompilowany przez bibliotekę OpenCL podczas wykonywania głównego programu. Plik ten jest wczytywany przez platformę, a następnie budowany do

obiektu (Kod źr. 7.), który później będzie wywoływany przez poszczególne wątki. Kod kernela zawiera implementację algorytmu metody źródeł pozornych (Rysunek 5.1.), który na podstawie danych z kontekstu obliczeniowego wykonuje obliczenia dla jednej wariacji powierzchni odbijających. Ilość wszystkich wariacji powierzchni odbijających dla N powierzchni i L rzędu źródeł pozornych wynosi N^L. Przestrzeń indeksowania stanowi kolejne liczby od 1 do N^L. Dla każdego wątku indeks przedstawiany jest w postaci liczby w systemie liczbowym o podstawie N. Każda kolejna cyfra takiej liczby odpowiada poszczególnej płaszczyźnie, dzięki czemu dla każdego indeksu uzyskujemy unikatową wariację powierzchni odbijających (Rysunek 5.2.).



Rys. 5.1. Schemat blokowy algorytmu metody źródeł pozornych dla pojedyńczej wariacji powierzchni odbijających.



Rys. 5.2. Schemat blokowy wyznaczania j-tej wariacji powierzchni odbijających.

Kod źr. 7. Budowanie obiektu kernela

```
program = clCreateProgramWithSource(
                                    context,
                                    1,
                                    &source,
                                    sourceSize,
                                    &status);
status = clBuildProgram(program, 1, devices, NULL, NULL, NULL);
kernel = clCreateKernel(program, "templateKernel", &status);
```

W zależności od wykorzystywanego urządzenia należy dobrać odpowiednio wielkości grup wątków. Zostały one zdefiniowane w zmiennych globalThreads i localThreads. Należy zwrócić uwagę, aby nie przekraczały one maksymalnych dopuszczalnych wartości dla danego urządzenia. W tym celu pobierane są informacje z bieżącego urządzenia o maksymalnych dopuszczalnych wartościach (Kod źr. 8.).

Kod źr. 8. Pobranie informacji o zasobach urządzenia

```
clGetDeviceInfo(
                devices[0],
                CL_DEVICE_MAX_WORK_GROUP_SIZE,
                sizeof(size_t),
                (void*)&maxWorkGroupSize,
                NULL);
clGetDeviceInfo(
                devices[0],
                CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS,
                sizeof(cl_uint),
                (void*)&maxDims,
                NULL);
clGetDeviceInfo(
                devices[0],
                CL_DEVICE_MAX_WORK_ITEM_SIZES,
                sizeof(size_t)*maxDims,
                (void*)maxWorkItemSizes,
                NULL);
```

Przed uruchomieniem kernela przekazywany jest do niego bufor danych (Kod źr. 9.).

Kod źr. 9. Przekazanie bufora plABuffer do kernela

```

clSetKernelArg(
               kernel,
               1,
               sizeof(cl_mem),
               (void *)&plABuffer);
```

Następuje uruchomienie kolejki zadań i czekanie na ich wykonanie (Kod źr. 10.). Po wykonaniu zadań kolejka zostaje zwolniona.

Kod źr. 10. Uruchomienie kolejki zadań dla bufora wejściowego

```
clEnqueueNDRangeKernel(
                       commandQueue,
                       kernel,
                       1,
                       NULL,
                       globalThreads,
                       localThreads,
                       0,
                       NULL,
                       &events[0]);
clWaitForEvents(1, &events[0]);
clReleaseEvent(events[0]);
```

Następnie dane wyjściowe odczytywane są z bufora wyjściowego (Kod źr. 11.)

Kod źr. 11. Uruchomienie kolejki zadań dla bufora wyjściowego

```
clEnqueueReadBuffer(
                      commandQueue,
                      outputBuffer,
                      CL_TRUE,
                      0,
                      width * sizeof(cl_uint),
                      output,
                      0,
                      NULL,
                      &events[1]);
```

5.3. Obsługa aplikacji

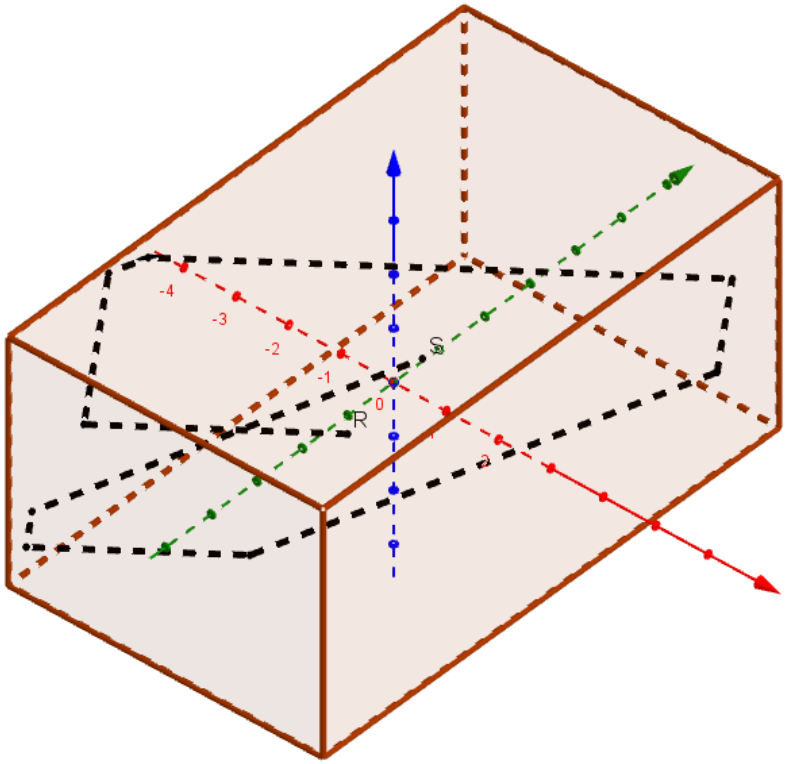
Przed kompilacją programu należy przygotować kod pod urządzenia, na jakich będzie on wykonywany. W zależności od typu procesora należy zdefiniować dyrektywę `CL_PRESENT_DEVICE` jako `CL_DEVICE_TYPE_CPU` dla procesorów, lub `CL_DEVICE_TYPE_GPU` dla kart graficznych. W zależności od parametrów urządzenia należy odpowiednio zdefiniować rozmiary grup wątków w zmiennych localThreads i globalThreads, a następne skompilować do pliku wykonywalnego dołączając biblioteki standardu OpenCL. Plik wykonywalny przyjmuje dane w postaci pliku wejściowego o nazwie input.txt. W danym pliku należy zdefiniować w kolejnych liniach rząd pozornych źródeł, współrzędne punktu źródła dźwięku i punktu obserwacji, współczynniki płaszczyzn powierzchni odbijających w postaci ogólnej, granice powierzchni dla poszczególnych osi oraz współczynniki pochłaniania dźwięku dla każdej powierzchni (Kod źr. 12).

Kod źr. 12. Plik wejściowy programu

```
3               %rząd siatki źródeł pozornych
0.2, 0.4, 0.3   % współrzędne punktu źródła
0.6, -1.7, 0.4 % współrzędne punktu obserwacji
0, 0, 1, 3.3    % współczynniki płaszczyzny dla pierwszej powierzchni
    odbijającej
-2, -7, -3.3    % dolne granice powierzchni dla osi x, y i~z
2, 7, -3.3      % gorne granice powierzchni dla osi x, y i~z
0.71            % współczynnik pochłaniania powierzchni
```

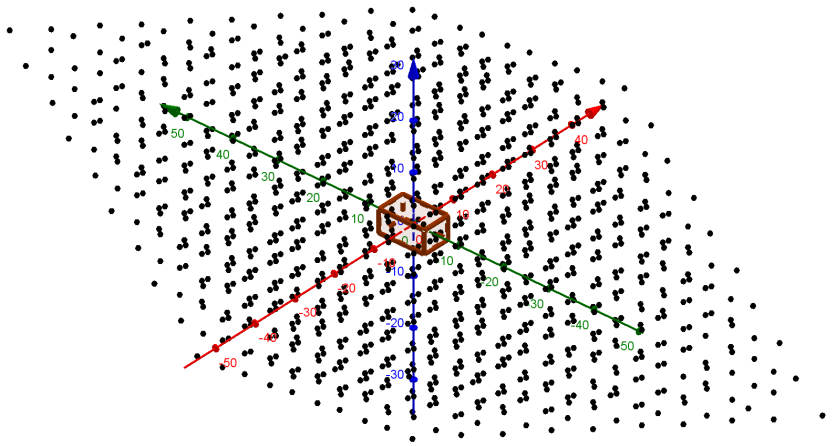
Dane wyjściowe otrzymywane są w postaci pliku output.txt. Po wykonaniu obliczeń w pliku znajdują się współrzędne punktów siatki źródeł pozornych wraz z ich energią.

Przy ustawieniu dyrektywy `write_file` na *true* i wybraniu kombinacji powierzchni odbijających program wygeneruje plik ze skryptem wejściowym do programu GeoGebra. Wygenerowany skrypt rysuje trójwymiarową grafikę (Rys. 5.3.), na której znajdują się powierzchnie odbijające, punkt źródła dźwięku, punkt pozornego źródła, punkt obserwacji oraz ścieżka promienia dźwiękowego.



Rys. 5.3. Ścieżka promienia dźwiękowego (S - punkt źródła dźwięku, R - punkt obserwacji).

Dane z pliku output.txt mogą posłużyć do wygenerowania siatki źródeł pozornych (Rys. 5.4.).



Rys. 5.4. Siatka źródeł pozornych.

6. Obliczenia testowe

6.1. Wprowadzenie

Poza realizacją algorytmu autor wykonuje testy napisanej aplikacji. Testy poprawności implementacji zostały wykonane na podstawie testów jednostkowych. Przy użyciu ów testów autor sprawdza poprawność przekształceń geometrycznych i transferu danych między pamięciami urządzeń. Test poprawności całego modelu wykonano wyznaczając algebraicznie siatki źródeł pozornych dla kilku początkowych rzędów dla różnych układów geometrycznych pomieszczeń i porównując je z wynikami uzyskanymi przez obliczenia przy użyciu aplikacji. Testy dla źródeł pozornych wyższych rzędów zostały wykonane dla pojedynczych punktów.

W rozdziale 6.2 autor przedstawia przykładowe wykorzystanie aplikacji do analizy pola akustycznego. W tym celu przygotowane zostały geometryczne modele pomieszczeń o zróżnicowanych wymiarach i parametrach pochłaniania dźwięku, a następnie porównane ze sobą. Aplikacja umożliwia jedynie obliczenia na bryłach wypukłych z powierzchniami prostopadłymi do osi układu współrzędnych. Ogranicza to model geometrii do prostopadłościanu z możliwością naniesienia dodatkowych powierzchni o innym współczynniku pochłaniania na ściany boczne tego prostopadłościanu.

W celu weryfikacji jednego z założeń pracy dotyczącego wydajności aplikacji autor uruchamia program na różnych urządzeniach. Do testów wykorzystane zostały urządzenia zróżnicowane pod względem wydajności i ilości jednostek arytmetyczno-logicznych. Autor porównuje czasy obliczeń na danych urządzeniach dla różnych rzędów siatki źródeł pozornych.

6.2. Użycie aplikacji

6.2.1. Obliczenia na pomieszczeniach zróżnicowanych geometrycznie

Do obliczeń dla zróżnicowanych pomieszczeń zdefiniowano 3 modele geometryczne (Tabele 6.1-6.3).

Tab. 6.1. Geometryczny model pomieszczenia 1

	x	y	z
Wymiary pomieszczenia	6	10	4.6
Źródło	0.2	0.4	0.33
Punkt obserwacji	0.6	-1.7	0.4

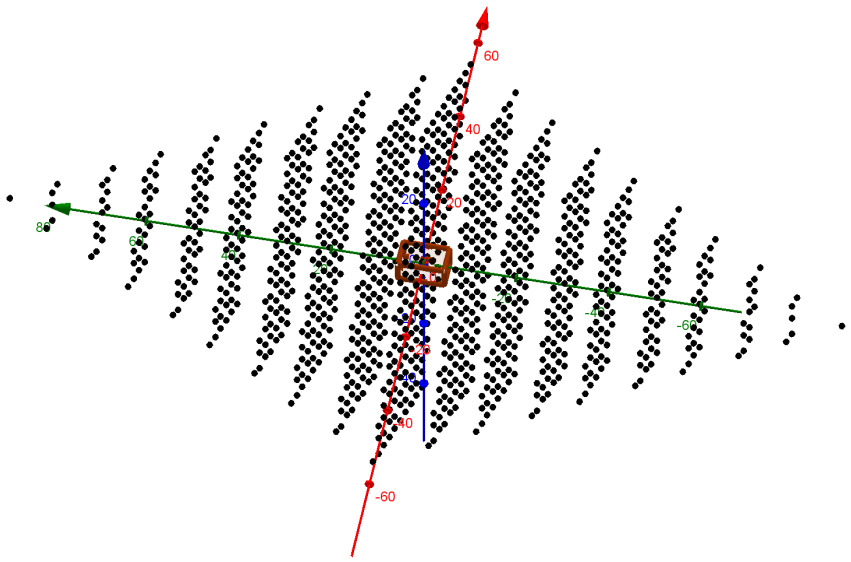
Tab. 6.2. Geometryczny model pomieszczenia 2

	x	y	z
Wymiary pomieszczenia	6	10	4.6
Źródło	-2.9	-4.9	-2.2
Punkt obserwacji	-2.87	-4.39	-2.08

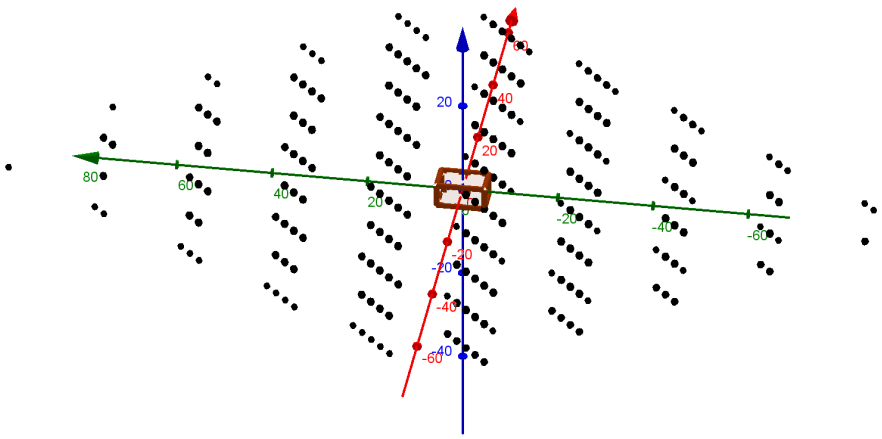
Tab. 6.3. Geometryczny model pomieszczenia 3

	x	y	z
Wymiary pomieszczenia	20	2	2
Źródło	-9	0.2	0.3
Punkt obserwacji	8	-0.4	0.1

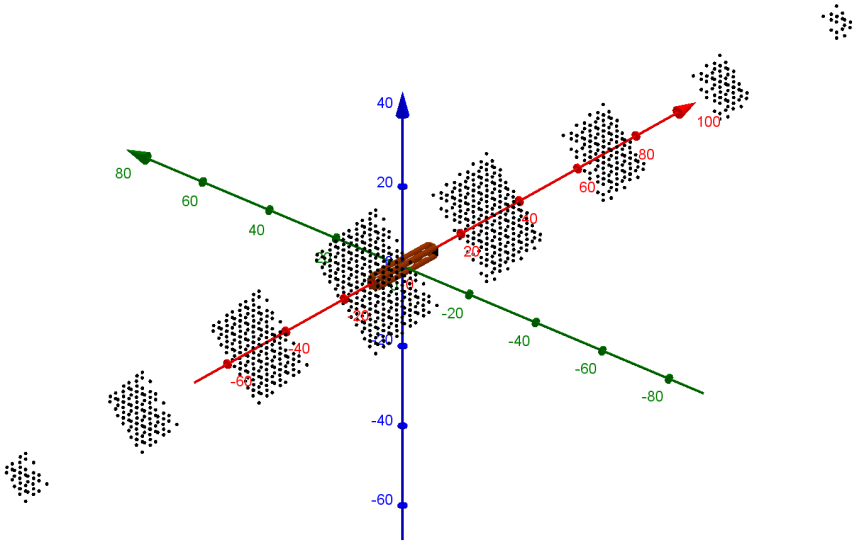
Dla danych modeli zostały wykonane wyliczenia siatek źródeł pozornych do 12 rzędu (Rys. 6.1-6.3).



Rys. 6.1. Siatka źródeł pozornych modelu 1.

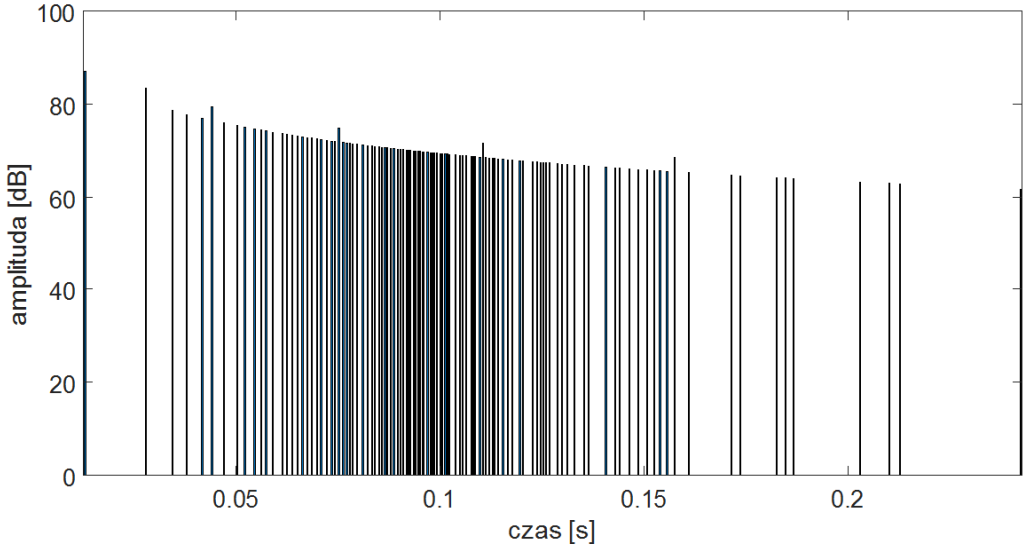


Rys. 6.2. Siatka źródeł pozornych modelu 2.

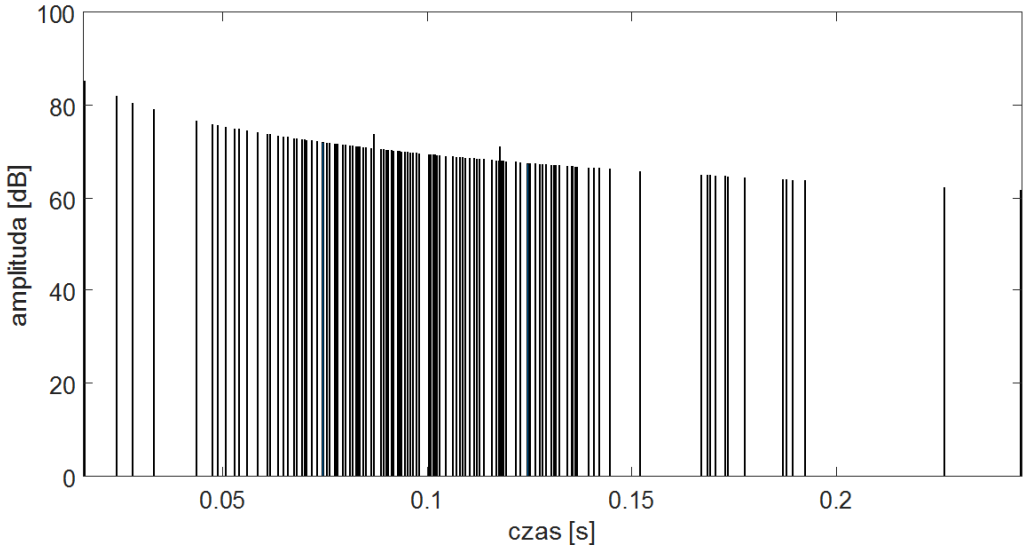


Rys. 6.3. Siatka źródeł pozornych modelu 3.

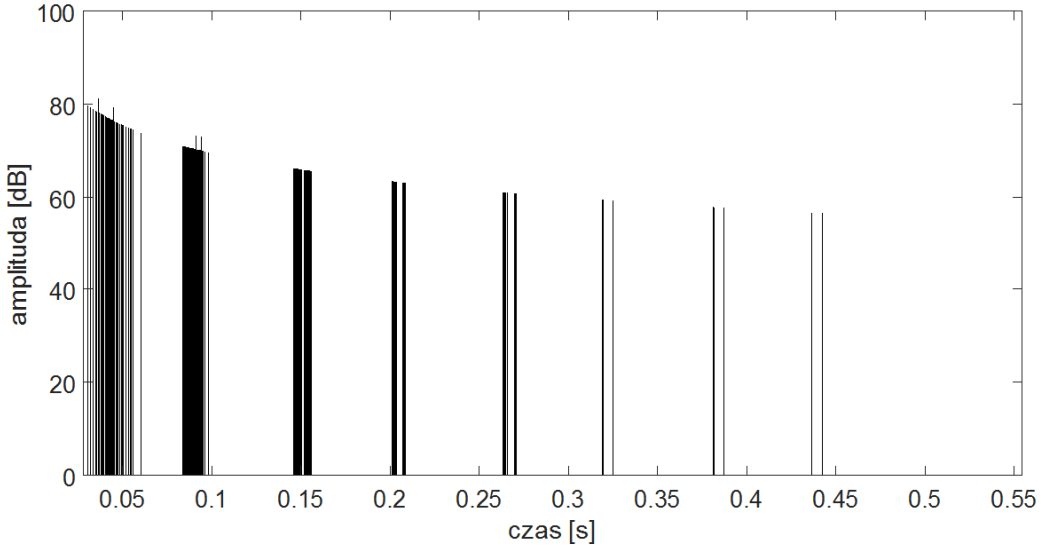
Na podstawie siatek źródeł pozornych zostały wyznaczone echogramy oraz krzywe zaniku dźwięku (Rys. 6.4-6.9).



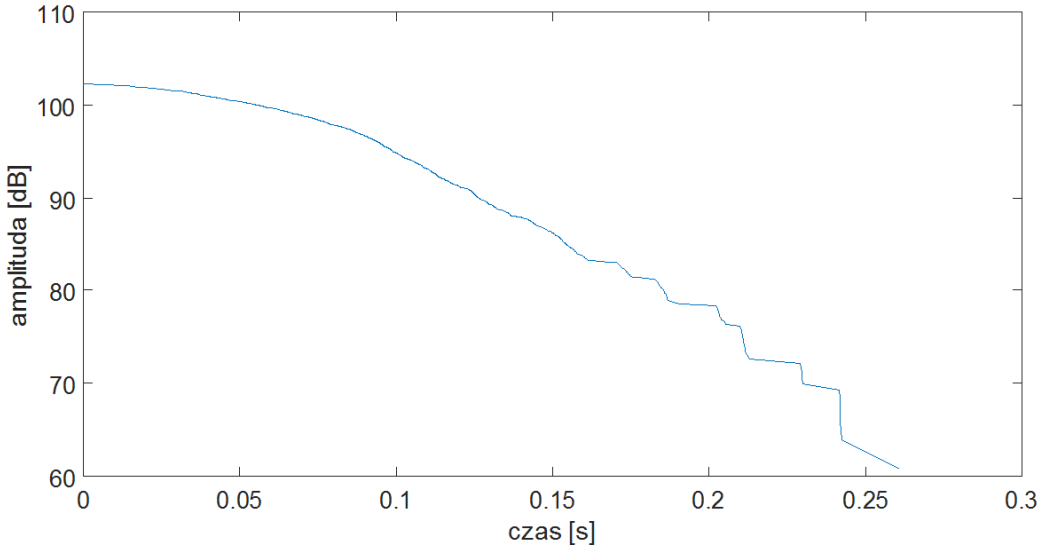
Rys. 6.4. Echogram modelu 1.



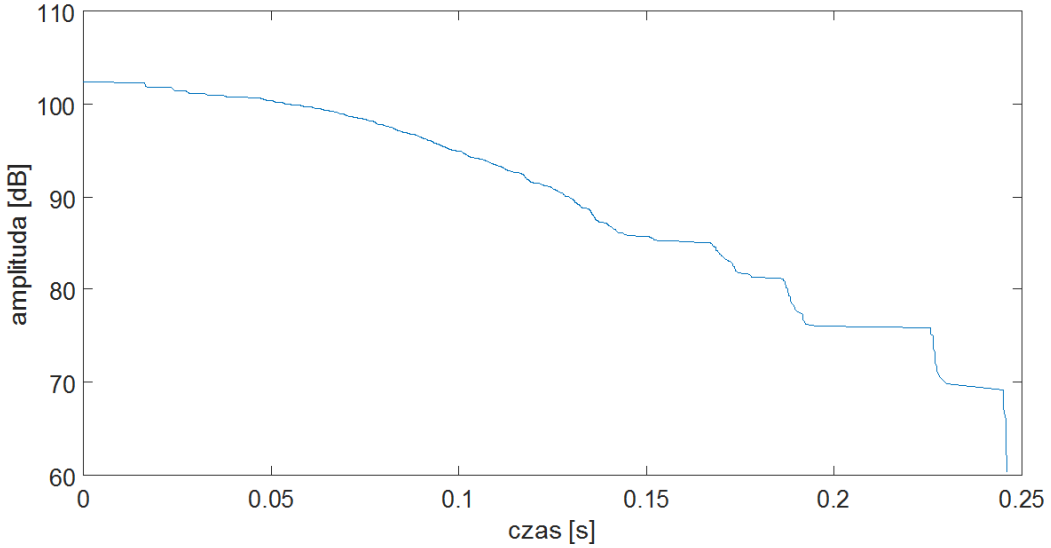
Rys. 6.5. Echogram modelu 2.



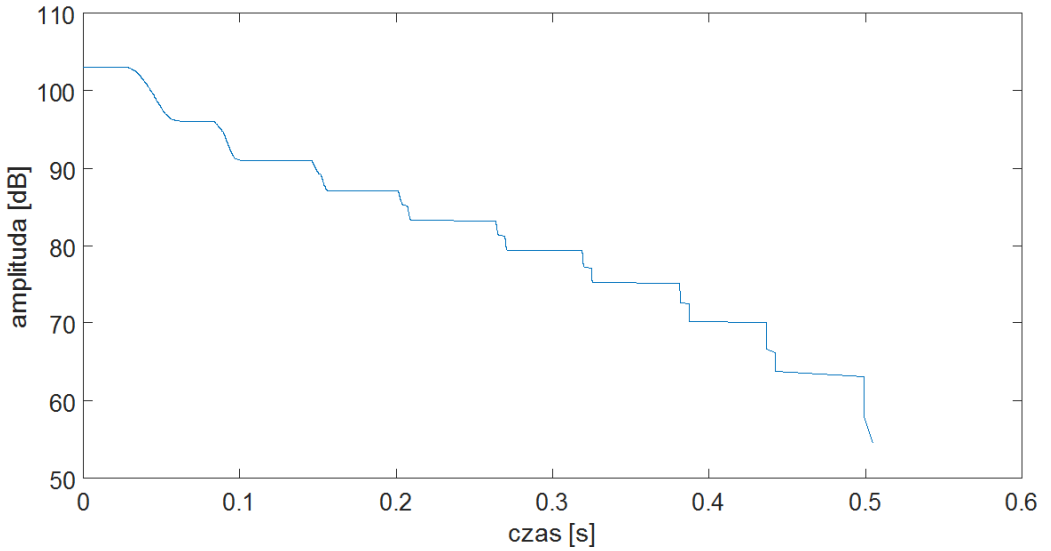
Rys. 6.6. Echogram modelu 3.



Rys. 6.7. Krzywa zaniku dźwięku modelu 1.



Rys. 6.8. Krzywa zaniku dźwięku modelu 2.



Rys. 6.9. Krzywa zaniku dźwięku modelu 3.

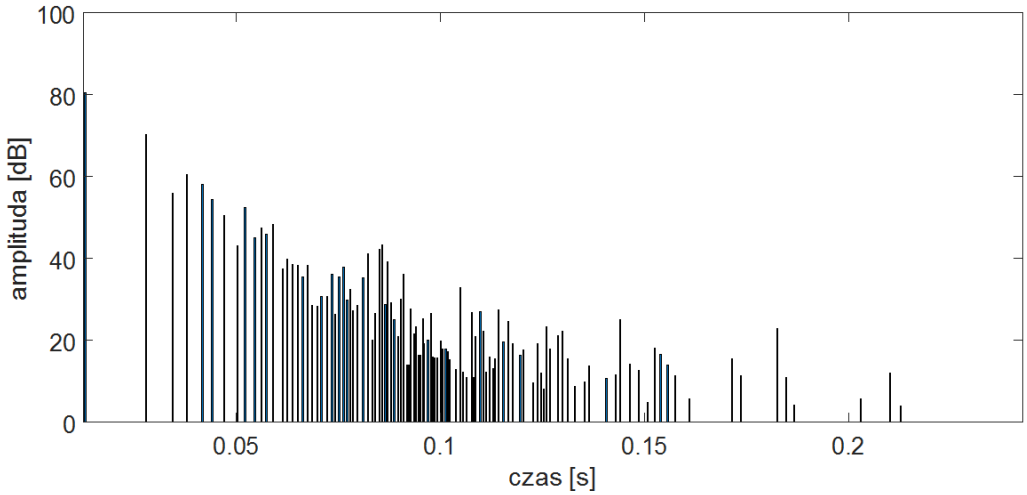
6.2.2. Obliczenia na pomieszczeniach o różnych parametrach pochłaniania

Do obliczeń dla zróżnicowanych współczynników pochłaniania wykorzystano model geometryczny 1 z rozdziału 6.2.1. Dla danego modelu zdefiniowano 3 różne zestawy współczynników pochłaniania (Tabela 6.4).

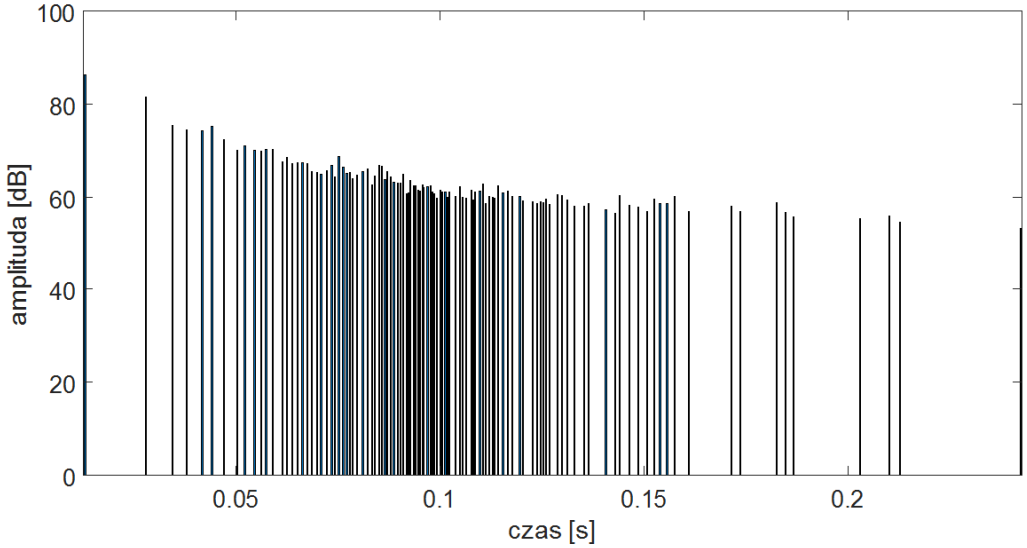
Tab. 6.4. Wartośći współczynników pochłaniania dźwięku dla poszczególnych powierzchni w różnych zestawach danych

powierzchnia	zestaw 1	zestaw 2	zestaw 3
góra	0.71	0.21	0.71
dół	0.78	0.18	0.78
lewo	0.85	0.25	0.02
prawo	0.72	0.12	0.72
przód	0.84	0.24	0.84
tył	0.61	0.21	0.01

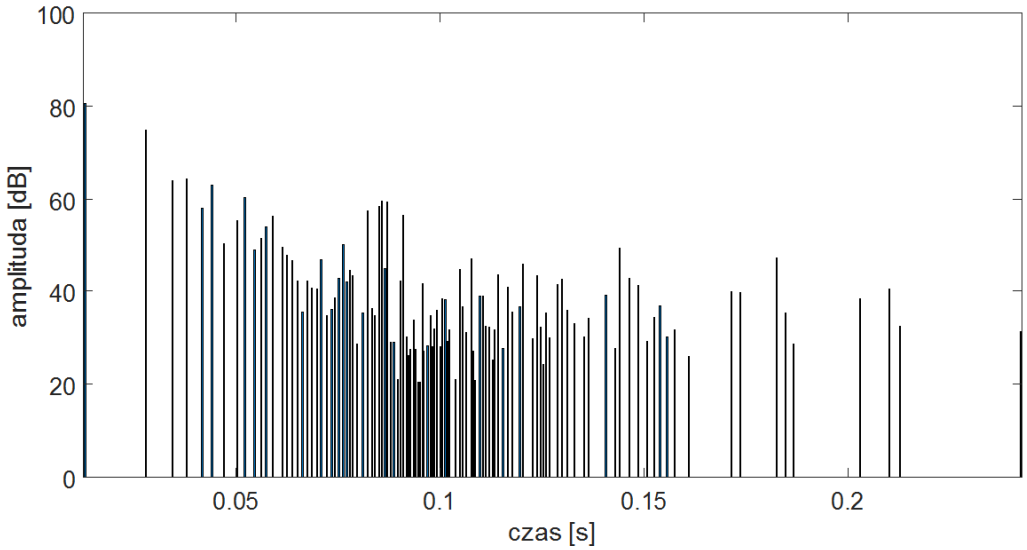
Dla danych zestawów siatki źródeł pozornych stanowią te same punkty. Zróżnicowane będą jedynie poziomy poszczególnych promieni dźwiękowych dochodzących do punktu obserwacji, co można zauważyć na echogramach (Rys. 6.10-6.12).



Rys. 6.10. Echogram modelu 1.

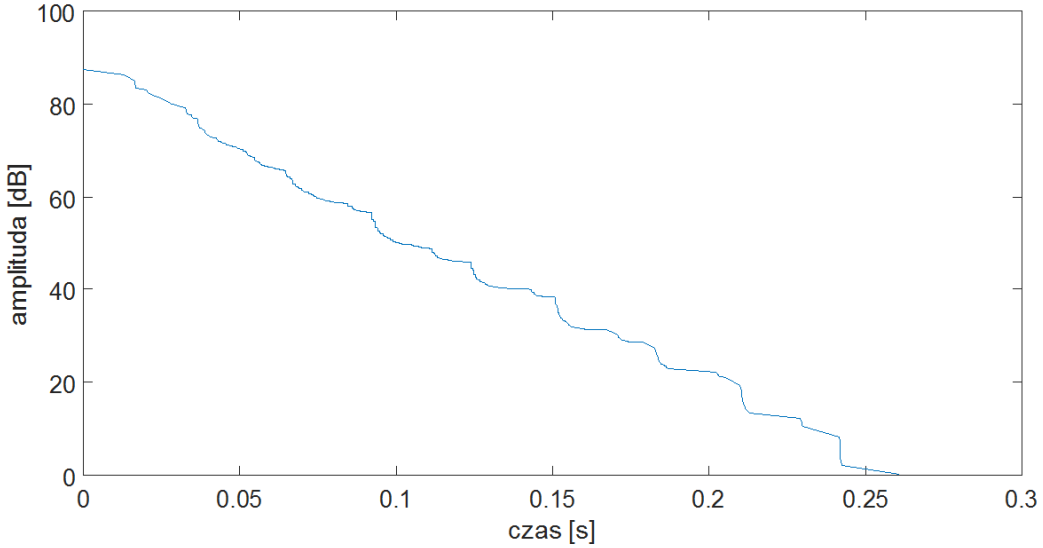


Rys. 6.11. Echogram modelu 2.

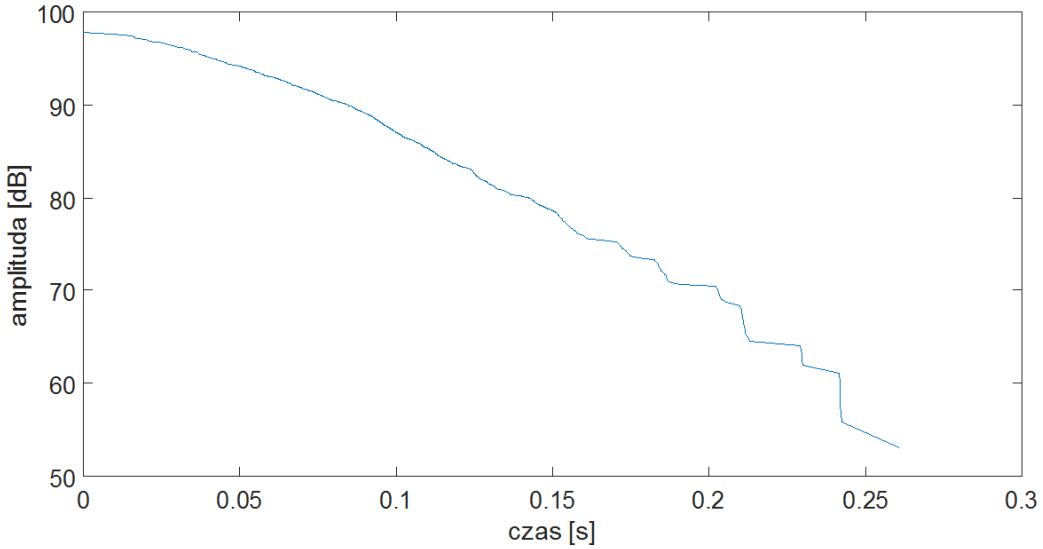


Rys. 6.12. Echogram modelu 3.

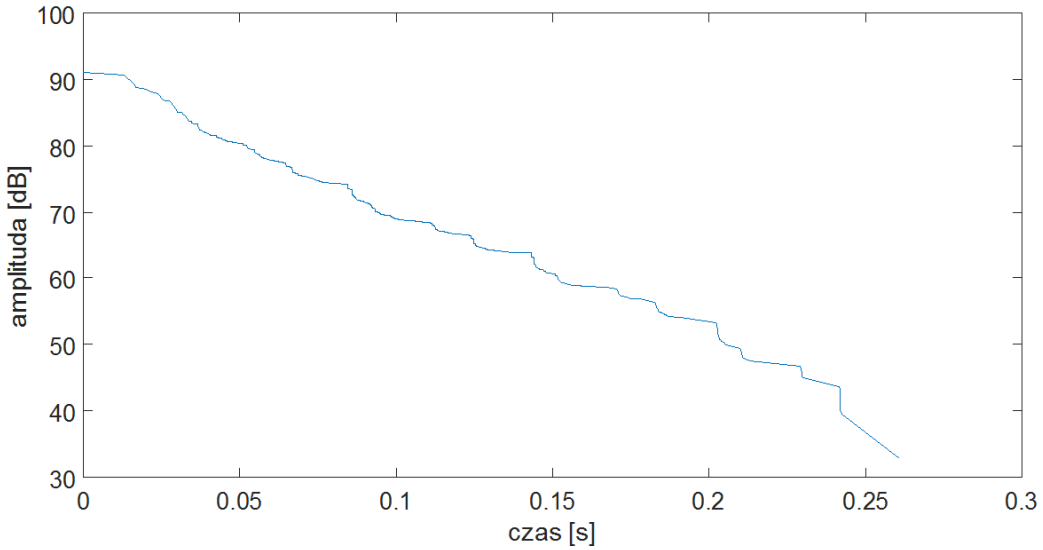
Różne parametry pochłaniania mają wpływ na kształt i szybkość zanikania krzywej zaniku dźwięku (Rys. 6.13-6.15).



Rys. 6.13. Krzywa zaniku dźwięku modelu 1.



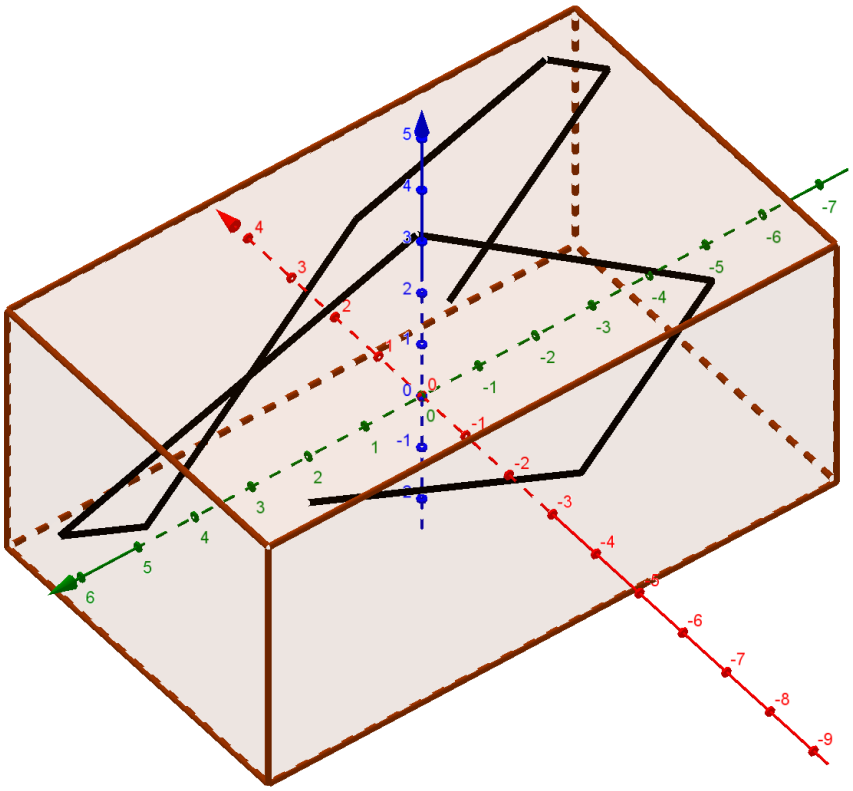
Rys. 6.14. Krzywa zaniku dźwięku modelu 2.



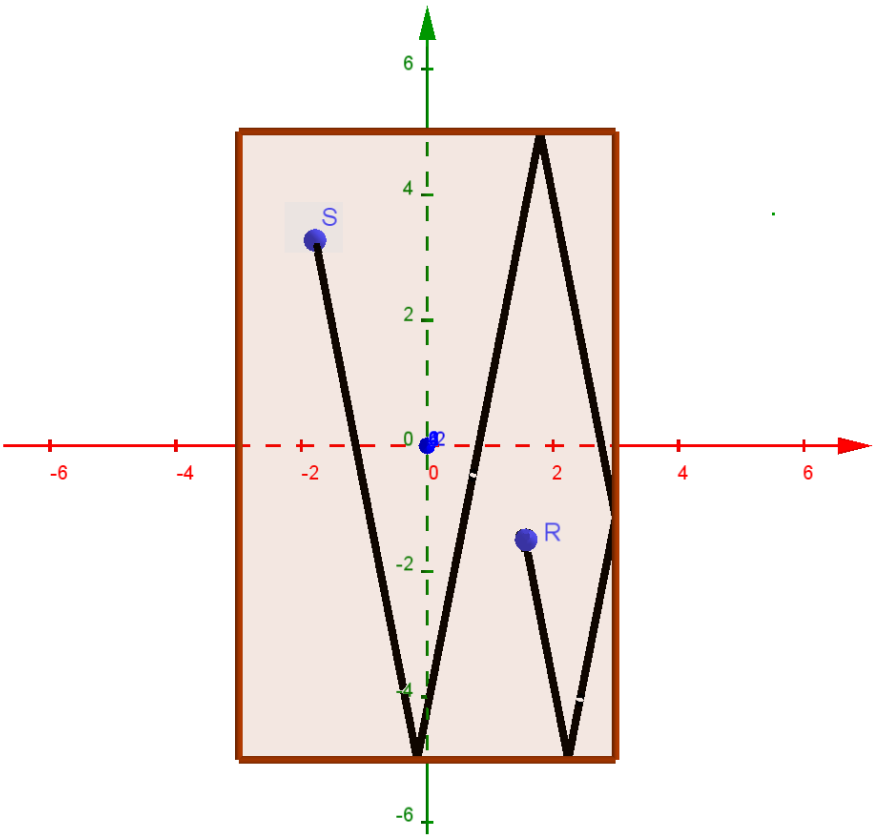
Rys. 6.15. Krzywa zaniku dźwięku modelu 3.

6.2.3. Wizualizacja poszczególnych promieni dźwiękowych

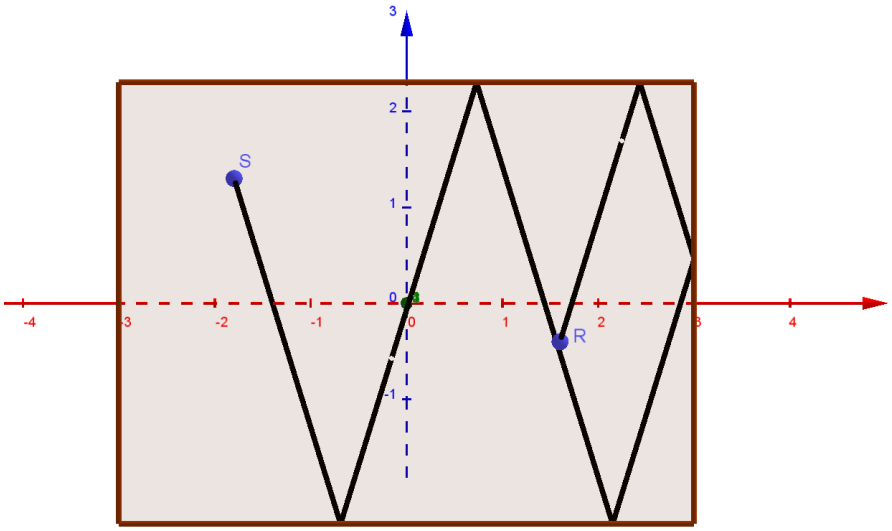
Podczas testów aplikacji użyteczna okazała się możliwość wizualizacji ścieżki wybranego promienia dźwiękowego. Aplikacja autora pracy umożliwia wygenerowanie skryptu jako plik wejściowy do programu GeoGebra, który umożliwia prezentację ścieżki promienia dźwiękowego dla wybranych wariacji powierzchni odbijających (Rysunek 6.16 - 6.19.).



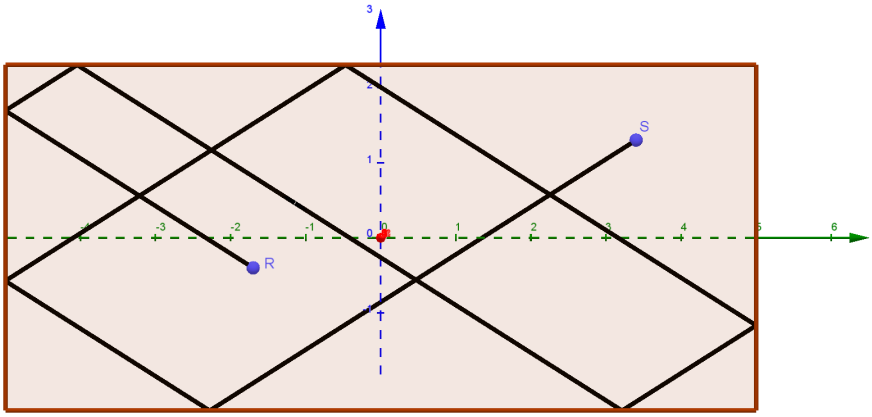
Rys. 6.16. Ścieżka wybranego promienia dźwiękowego dla 8 odbić.



Rys. 6.17. Ścieżka wybranego promienia dźwiękowego dla 8 odbić w rzucie na płaszczyznę XY (S - punkt źródła dźwięku, R - punkt obserwacji).

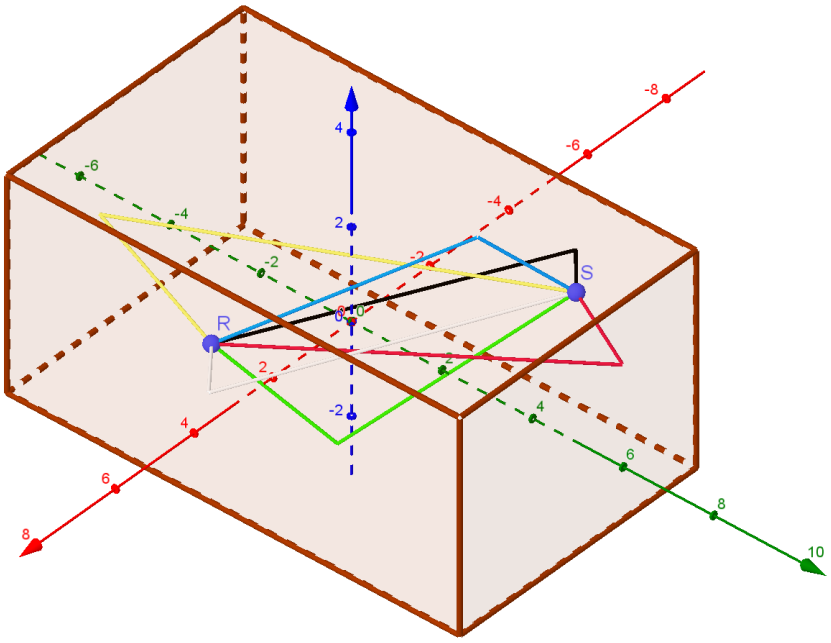


Rys. 6.18. Ścieżka wybranego promienia dźwiękowego dla 8 odbić w rzucie na płaszczyznę XZ (S - punkt źródła dźwięku, R - punkt obserwacji).

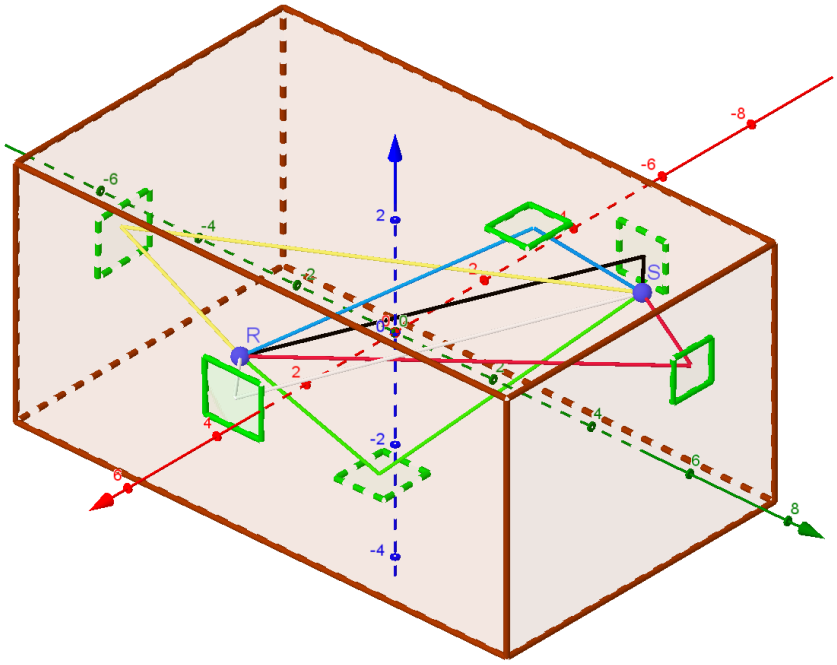


Rys. 6.19. Ścieżka wybranego promienia dźwiękowego dla 8 odbić w rzucie na płaszczyznę YZ (S - punkt źródła dźwięku, R - punkt obserwacji).

Wizualizacja ścieżek promieni dźwiękowych może być przydatna w akustyce architektonicznej przy projektowaniu rozmieszczenia ustrojów akustycznych. Rysując promienie dźwiękowe dla pierwszych odbić (Rysunek 6.20.) możemy wyznaczyć punkty, w których dane odbicia zachodzą. Umieszczając materiał o innych właściwościach pochłaniania w miejscach pierwszych odbić możemy wpłynąć na ilość energii jaka przychodzi do odbiornika w pierwszych milisekundach, co pozwala bezpośrednio wpłynąć na wskaźniki C50, C80, D50 oraz wskaźnik zrozumiałości mowy.

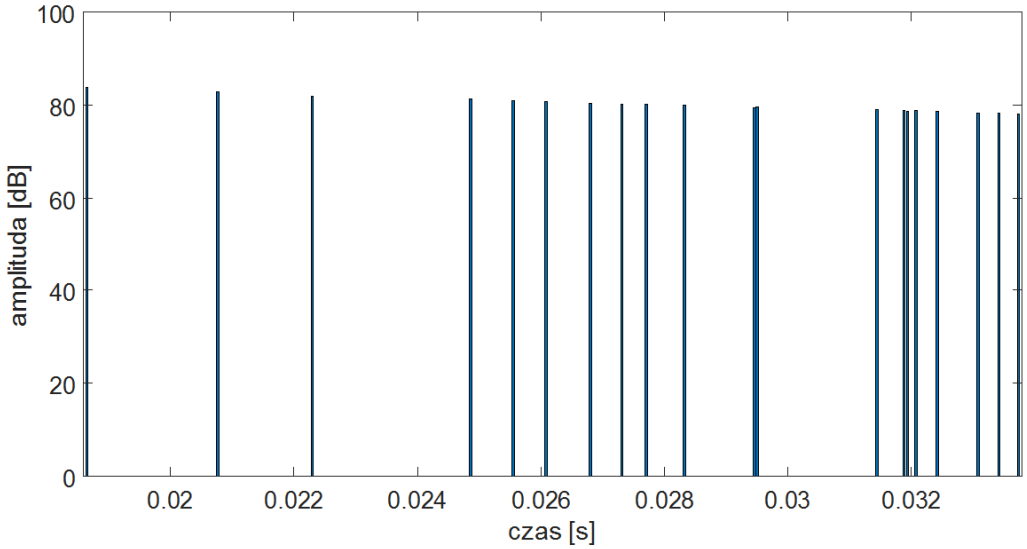


Rys. 6.20. Ścieżki promieni dźwiękowych dla pierwszych odbić (S - punkt źródła dźwięku, R - punkt obserwacji).

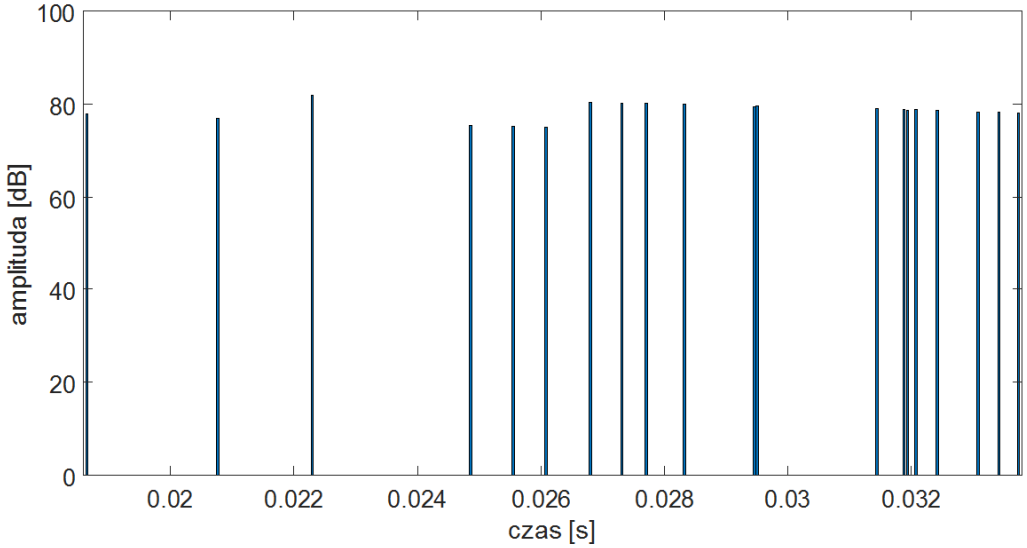


Rys. 6.21. Ścieżki promieni dźwiękowych dla pierwszych odbić wraz z rozmieszczeniem materiałów pochłaniających (S - punkt źródła dźwięku, R - punkt obserwacji).

Dla tak przygotowanych modeli możemy wyznaczyć echogramy (Rysunek 6.21 - 6.22.) i ilość energii, która dochodzi do punktu obserwacji. Pozwala to na dostrojenie modelu do wybranych założeń projektowych, jakimi mogą być wartości wcześniej wymienionych wskaźników.



Rys. 6.22. Echogram dla 20 pierwszych odbić.



Rys. 6.23. Echogram dla 20 pierwszych odbić dla modelu z ustrojami pochłaniającymi.

6.3. Testy wydajnościowe

Testy wydajnościowe przeprowadzono na 2 różnych architekturach procesorów – CPU i GPU. Do pomiarów na CPU posłużył procesor Intel Core i5-2520M (Tabela 6.5). Pomiaru przy użyciu GPU przeprowadzono na kartach Radeon R7 250X (Tabela 6.6) oraz Radeon R9 270X (Tabela 6.7). Obliczenia testowe zostały wykonane na zmiennych o pojedynczej precyzji.

Tab. 6.5. Dane techniczne procesora Intel Core i5-2520M

taktowanie	2,50 GHz
liczba rdzeni	2
liczba wątków	4

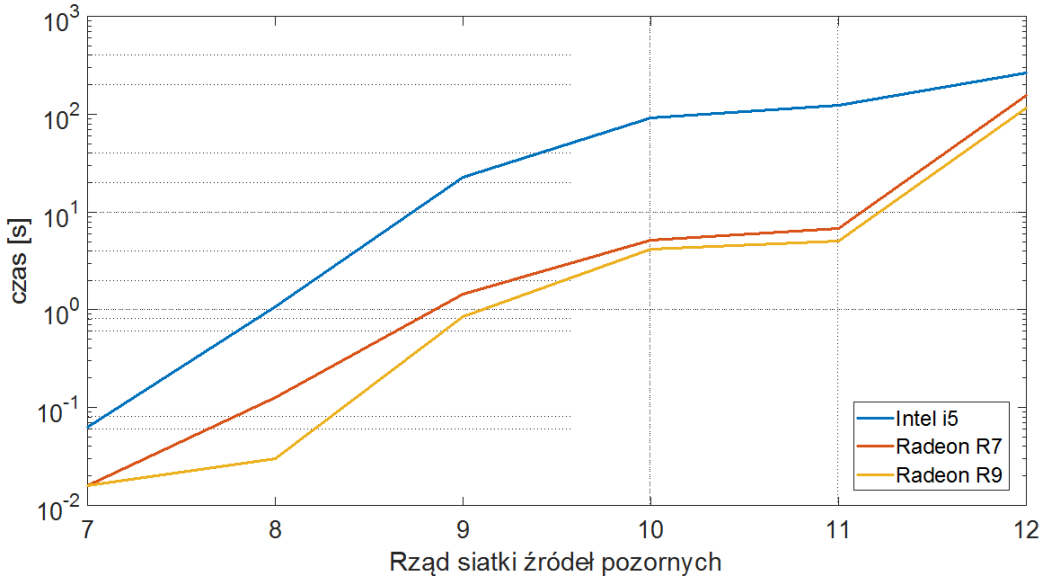
Tab. 6.6. Dane techniczne karty graficznej Radeon R7 250x

taktowanie	1000 MHz
liczba rdzeni GPU	512

Tab. 6.7. Dane techniczne karty graficznej Radeon R9 270x

taktowanie	1030 MHz
liczba rdzeni GPU	2560

Pomiary przeprowadzono na modelu 1 z rozdziału 6.2.1 (Rysunek 6.1.) dla rzędów źródeł pozornych od 5 do 12. Autor porównał ze sobą czasy obliczeń na różnych platformach (Rysunek 6.13., Tabela 6.8.).

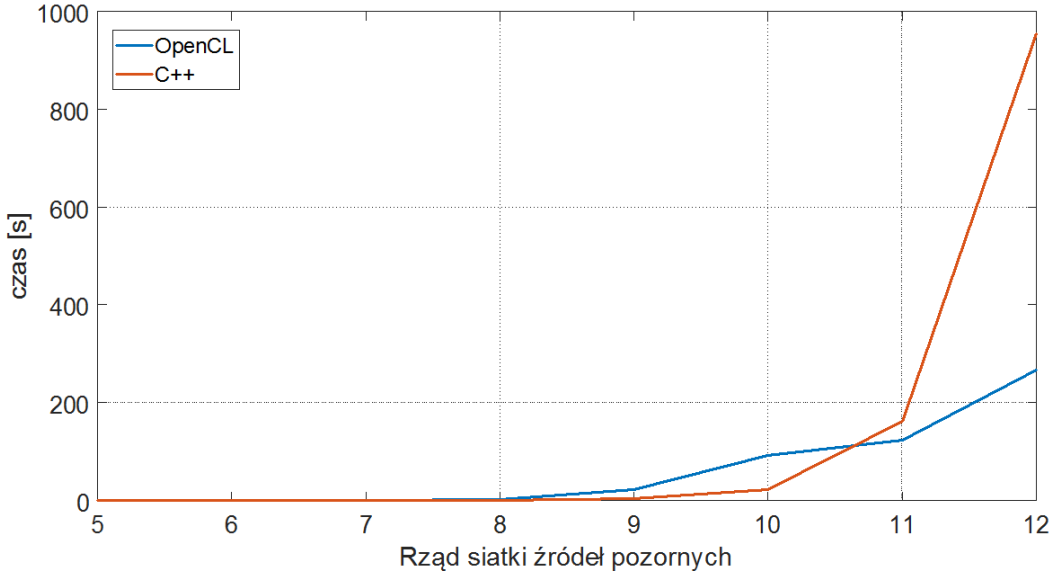


Rys. 6.24. Zależność czasu obliczeń od liczby wyliczanych rzędów siatki źródeł pozornych dla różnych urządzeń.

Tab. 6.8. Zależność czasu obliczeń od liczby wyliczanych rzędów siatki źródeł pozornych dla języka C++ i OpenCL. Kod został uruchomiony na architekturze Intel Core i5-2520M (Tabela 6.5).

rząd siatki	czas obliczeń [ms]		
	Intel i5	Radeon R7	Radeon R9
5	0	0	0
6	0	0	0
7	63	16	16
8	11E2	125	30
9	23E3	14E2	843
10	91E3	52E2	42E2
11	12E4	68E2	51E2
12	27E4	16E4	12E2

W celu sprawdzenia wydajności implementacji algorytmu w środowisku OpenCL autor porównał czas obliczeń algorytmu w tym środowisku z czasem obliczeń algorytmu zaimplementowanego przy użyciu czystego kodu C++, bez użycia wielowątkowości (Rysunek 6.14., Tabela 6.9.).



Rys. 6.25. Zależność czasu obliczeń od używanego środowiska.

Tab. 6.9. Zależność czasu obliczeń od liczby wyliczanych rzędów siatki źródeł pozornych dla różnych urządzeń.

rząd siatki	czas obliczeń [ms]	
	OpenCL	C++11
5	0	1
6	0	16
7	63	85
8	11E2	545
9	23E3	33E2
10	91E3	22E3
11	12E4	16E4
12	27E4	95E4

6.4. Błąd obliczeń

Większa część algorytmu metody źródeł pozornych odpowiada za sprawdzenie czy dany promień dźwiękowy istnieje w danym modelu geometrycznym. Sprawdzenie odpowiada jedynie za decyzję czy dany promień będzie uwzględniany w siatce źródeł pozornych i nie wpływa na błąd wyniku końcowego. Błąd obliczonych pozycji siatki źródeł pozornych powstaje w wyniku narastania błędu zaokrąglenia podczas operacji arytmetycznych związanych z wyznaczaniem kolejnych odbić lustrzanych punktu źródła. Do obliczenia współrzędnej odbicia należy wykonać 4 mnożenia co czterokrotnie zwiększa wartość błędu względnego ϵ. Dla współrzędnych zapisanych w zmiennych typu float daje to błąd względny równy $|\delta|$ = 2 E-21. Dla źródeł pozornych N-tego rzędu wartość tego błędu wzrasta o N razy. Nawet dla źródeł pozornych 16-tego rzędu błąd wynosi $|\delta|$ = 2 E-17. W akustyce architektonicznej błędy danych wejściowych są o

kilkanaście rzędów wielkości większe od tego błędu przez co może on zostać pominięty w obliczeniach.

7. Podsumowanie

W ramach pracy napisana została aplikacja, która implementuje metodę źródeł pozornych w aplikacji na heterogeniczne platformy obliczeniowe. Wyniki testów wydajności z rozdziału 5.3 wykazały, że obliczenia algorytmu zaimplementowanego w OpenCL wykonane z użyciem kart graficznych są znacznie szybsze niż na zwykłych procesorach i możliwe jest skrócenie czasu obliczeń tej metody wykorzystując bardziej złożone platformy. Algorytm napisany w OpenCL na procesorze CPU wykonuje się szybciej niż algorytm napisany w czystym języku C++ dla rzędu odbić powyżej 11. Przy małej ilości odbić na szybkość algorytmu wpływa głównie czas dostępu do pamięci, który jest znacznie krótszy przy jednowątkowych aplikacjach.

Wykorzystanie popularnej biblioteki OpenCL i prezentacja aplikacji w postaci otwartego kodu umożliwia wykorzystanie jej w innych aplikacjach lub do niezależnych badań. Otwarty kod kernela pozwala na implementacje programu w dowolnym języku platformy obsługiwanym przez standard OpenCL (m. in. Python, Matlab).

Obliczenia w rozdziale 5.2 ukazują użyteczność aplikacji w analizie pomieszczeń. Analiza wyznaczonych echogramów pozwala na wstępną ocenę warunków akustycznych pomieszczenia i daje informacje o wczesnych odbiciach. Użycie zewnętrznej nakładki graficznej w programie GeoGebra pozwoliło na analizę poszczególnych odbić, co może być użyteczne przy projektowaniu rozmieszczenia ustrojów akustycznych w pomieszczeniu. Wyznaczone siatki źródeł pozornych mogą posłużyć do dalszej analizy warunków akustycznych i wyznaczenia wskaźników C50, C80, D50, wskaźnika zrozumiałości mowy i innych parametrów nie rozpatrzonych w pracy.

Bibliografia

[1] W.C. Sabine: „Collected Papers on Acoustics", w: *Harvard University Press*, 1922

[2] L. Cremer: „Geometrische Raumakustik", w: *Die Wissenschaftlichen Grundlagen der Raumakusitk*, 1948

[3] A. Appel: „Some techniques for shading machine renderings of solids", w: *Spring Joint Computer Conference*, 1968

[4] S. Krokstad, S. Storm i Sorsdal S.: „Calculating the acoustical room response by the use of a ray tracing technique", w: *Journal of Sound and Vibration* 8, 1968, s. 118–125

[5] W. Straszewicz: *Analiza geometryczna właściwości pola akustycznego w obszarach ograniczonych*, Warszawa: WPW, 1974

[6] J. Allen i D. Berkley: „Image method for efficiently simulating small-room acoustics", w: *Journal of Society of America* 65, 1979, s. 943–950

[7] *Introduction*,
`http://catt.se`, CATT-Acoustic, 2017

[8] *ODEON Room Acoustics Software*,
`https://odeon.dk/download/Version15/OdeonManual.pdf`, Odeon, 2018

[9] Z. h. Fu i J. w. Li: „GPU-based image method for room impulse response calculation", w: *Multimedia Tools and Applications*, 2016, s. 1–17

[10] T. Reuben: *Wayverb manual*,
`https://reuk.github.io/wayverb/introduction.html`, Wayverb, 2016

[11] Naylor G. M.: „ODEON - Another Hybrid Room Acoustical Model", w: *Applied Acoustics* 38, 1993, s. 131–143

[12] D. van Maercke: „Simulation of sound fields in time and frequency domain using a geometrical model", w: *Proc 12th ICA* 38, 1993, s. 131–143

[13] B. I. Dalenback, P. Svensson i M. Kleiner: *Prediction and Auralization Based on a Combined Image Source/Ray-Tracing Model*, Proc. 14th ICA, 1992

[14] M. Vorlander: „Simulation of the transient and steady-state sound propagation in room using a new combined ray-tracing/image-source algorithm", w: *Journal of Society of America*, 1989

[15] J. von Neuman: *The First Draft Report on the EDVAC*, 1945

[16] Dr. Dobb's Journal: „386 vs. 030: the Crowded Fast Lane", w: *UBM*, 1988

[17] R. J. Randi: *OpenGL Shading Language*, Addison-Wesley Professional, 2004

[18] A. Munshi: *The OpenCL Specification*, https://www.khronos.org/registry/OpenCL/specs/opencl-1.1.pdf, Khronos OpenCL Working Group, 2011

[19] C. Michael: *OpenCL™ Runtimes for Intel® Processors*, https://software.intel.com/en-us/articles/opencl-drivers, Intel, 2018

[20] *Xcelerit SDK*, https://www.xcelerit.com/products/xcelerit-sdk, Xcelerit, 2018

[21] *Direct3D 11 Reference*, https://docs.microsoft.com/en-us/windows/win32/direct3d11/d3d11-graphics-r Microsoft, 2018

[22] *Introduction to Alea GPU*, http://www.aleagpu.com/release/3_0_2/doc, QuantAlea, 2018

[23] *ATI CTM Guide*, http://roland.pri.ee/doktor/papers/gpgpu/ATI_CTM_Guide.pdf, AMD, 2006

[24] A. Miller i K. Gregory: *C++ Amp: Accelerated Massive Parallelism With Microsoft Visual C++ - Microsoft*, Microsoft, 2012

[25] F. Jianbin, A. L. Varbanescu i H. Sips: „A comprehensive performance comparison of CUDA and OpenCL.", w: *International Conference on. IEEE*, 2011

[26] J. Lerner i M. Schankerman: *The Comingled Code: Open Source and Economic Development*, MIT Press, 2010

Załączniki

A. Kod kernela

Kod źr. 13. Plik Kernel.cl, część 1

```
__kernel void templateKernel(__global  uint * output,
                             __global  float * plA,
                             __global  float * plB,
                             __global  float * plC,
                             __global  float * plD,
                             __global  float * pt1x,
                             __global  float * pt1y,
                             __global  float * pt1z,
                             __global  float * pt2x,
                             __global  float * pt2y,
                             __global  float * pt2z,
                             __global  float * abso,
                             __local  uint * cache,
                             const     unsigned int level)
{
    uint L = get_global_id(0);
    uint LL = L;
    float Sx = 0.2; // harcoded values of source and receiver positions
    float Sy = 0.4;
    float Sz = 0.3;
    float Rx = 0.6;
    float Ry = -1.7;
    float Rz = 0.4;
    uint N = 6;
    bool czy_zly = false;
    bool czy_wektor = false;
```

Kod źr. 14. Plik Kernel.cl, część 2

```
1   for(int i = 0 ; i<level; i++)
2   {
3       cache[i] = L%N;
4       L=L/N;
5       float t = -(plA[cache[i]]*Sx + plB[cache[i]]*Sy +
            plC[cache[i]]*Sz +
            plD[cache[i]])/(plA[cache[i]]*plA[cache[i]] +
            plB[cache[i]]*plB[cache[i]] + plC[cache[i]]*plC[cache[i]]);
6       Sx = 2*plA[cache[i]]*t+Sx;
7       Sy = 2*plB[cache[i]]*t+Sy;
8       Sz = 2*plC[cache[i]]*t+Sz;
9
10  }
11
12  float vNx = Rx-Sx;
13  float vNy = Ry-Sy;
14  float vNz = Rz-Sz;
15
16  for( int i = level-1 ; i>=0 ; i--)
17  {
18      float t = -(plA[cache[i]]*Sx + plB[cache[i]]*Sy +
            plC[cache[i]]*Sz + plD[cache[i]])/(plA[cache[i]]*vNx +
            plB[cache[i]]*vNy + plC[cache[i]]*vNz);
19
20      float Cx = vNx*t + Sx;
21      float Cy = vNy*t + Sy;
22      float Cz = vNz*t + Sz;
```

Kod źr. 15. Plik Kernel.cl, część 3

```
1
2           if( vNx*plA[cache[i]] + vNy*plB[cache[i]] + vNz*plC[cache[i]]
                < 0)
3           {
4               czy_wektor = true;
5               break;
6           }
7           float eps = 0.00001;
8           if(Cx - pt1x[cache[i]] < -eps || Cx - pt2x[cache[i]] > eps ||
                Cy - pt1y[cache[i]] < -eps || Cy - pt2y[cache[i]] > eps ||
                Cz - pt1z[cache[i]] < -eps || Cz - pt2z[cache[i]] > eps)
9           {
10              czy_zly = true;
11              break;
12          }
13          float t2 = -(plA[cache[i]]*vNx + plB[cache[i]]*vNy +
                plC[cache[i]]*vNz)/(plA[cache[i]]*plA[cache[i]] +
                plB[cache[i]]*plB[cache[i]] + plC[cache[i]]*plC[cache[i]]);
14
15          vNx = 2*plA[cache[i]]*t2 + vNx;
16          vNy = 2*plB[cache[i]]*t2 + vNy;
17          vNz = 2*plC[cache[i]]*t2 + vNz;
18          Sx = Cx;
19          Sy = Cy;
20          Sz = Cz;
21      }
22      if(czy_zly == false && czy_wektor == false)
23      {
24          output[LL] = LL;
25      }
26      else
27      {
28          output[LL] = 0;
29      }
30  }
```