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 [W.C. Sabine: Collected Papers on Acoustics. Harvard University Press, 1922.] oraz jego kolejne przekształcenia nie uwzględniały wielu zjawisk akustycznych. Zarówno wzory empiryczne, jak i metody statystyczne, wyznaczały jedynie niektóre parametry pola akustycznego i często wprowadzały duży błąd w odniesieniu do rzeczywistych wartości. Analityczne rozwiązanie równania falowego, lub wykorzystanie metody elementów skończonych było zbyt kosztowne, lub niemożliwe. Jednymi z prostszych do implementacji metod analizy pola akustycznego były metody geometryczne. W 19xx roku L. Cremer w publikacji „Geometrische Raumakustik” [2] przedstawia geometryczna 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 sie światła. W 1968 Arhtur Appel w swojej publikacji [2] zamodelował punktowe źródło światła jako nieskończony zbiór półprostych, odbijających sie 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 [...] przedstawiono wykorzystanie tej metody do obliczenia czasu pogłosu pomieszczenia. Metodę promieniową zauważono jako opłacalną i zaczęto ją modyfikować co doprowadziło do powstania kolejnych metod geometrycznych. Jedną z osób, które przyczyniły się do rozwoju metod geometrycznych był W. Straszewicz. W swojej pracy [4] wykorzystuje podstawy metody źródeł pozornych (ang. Image-Source). W 1979 J. Allen i D. Berkley implementują metodę źródeł pozornych przy użyciu technik cyfrowych [5]. Wysokie zapotrzebowanie na grafikę komputerową, rozwój gier komputerowych i niski koszt obliczeniowy doprowadził do rozwoju powyższych metod symulacji fali. Należy jednak pamiętać, że metody geometryczne często nie uwzględniają wielu zjawisk falowych, co wymusza uzupełnianie je o inne metody numeryczne.

1.2 Cel i zakres pracy

Metoda źródeł pozornych jest obecnie wykorzystywana w popularnych programach do symulacji akustycznych [6] [7]. 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 optymalizacja czasu obliczeń dla danej metody poprzez implementacje jej na platformy heterogeniczne.

Pomysł wykorzystania kart graficznych do implementacji metody źródeł pozornych pojawia się w publikacji [[8] Z.-h. Fu and J.-w. Li, “GPU-based image method for room impulse response calculation,” Multimedia Tools and Applications, pp. 1–17, 2016.] , która wykorzystuje środowisko CUDA wspierające karty graficzne NVIDIA. Kolejną implementacją metody źródeł pozornych w środowisku kart graficznych jest aplikacja Wayverb [ ... ] przeznaczona jedynie dla systemów typu macOS. Ze względu na ograniczenia systemowe powyższych prac, kolejnym założeniem pracy autora jest wykonanie kodu przystosowanego do popularnych środowisk. 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.

Implementacja 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.

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, prześledzić ścieżkę propagacji dźwięku. Ze względu na liczne uproszczenia, w metodzie jest uwzględniane 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 wraz 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 to współczynnika pochłaniania dźwięku dla danego materiału. Kolejnym założeniem jest, że powierzchnie odbijające są powierzchniami płaskimi, a w polu występują 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. [7] [8] [9] [10]

2.3 Wyznaczanie siatki źródeł pozornych

W celu wyznaczenia siatki źródeł pozornych N-tego rzędu zdefiniujmy punkt źródła dźwięku S, punkt odbioru oraz K-liczny zbiór powierzchni odbijających P. (rysunek z modelem pomieszczenia).

Rysowando

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. (znów rysunek kilku odbić).

rysowando

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 powierzchnie z wariacji, a 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 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 1)

Wzór 1

Definiując dla każdej powierzchni odbijającej współczynnik pochłaniania alpha\_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 dzieląc przez energię źródła otrzymujemy energię źródła pozornego odpowiadającego odbiciom od powyższych powierzchni (wzór 2)

Wzór 2

2.5 Przykładowe użycie metody

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

Rysunek x

Rysunek y

Położenia źródeł pozornych w siatce daje 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 ( echogram).

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 (rysunek milion).

Rysunek milion

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 w połączeniu [coś tam].

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 […] pozwoliła na uruchomienie pierwszych komputerów ogólnego przeznaczenia. Mimo, że Architektura Harwardzka […] 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. Był on bazą do rozwijania kolejnych urządzeń i tak w 1949 roku powstał EDSAC (akronim od ang. Electronic Delay Storage Automatic Calculator). Był to 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 akcelerację 2D i 3D korzystając z wbudowanych funkcji do generowania obrazu. W najnowszych procesorach grafiki umożliwiono użytkownikowi zaprogramowanie je 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.

Rysuneczek architektury GPGPU

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 [Rost, Randi J. OpenGL Shading Language. Addison-Wesley Professional. ISBN 0-321-19789-5.] czy ray-tracing [Appel A. (1968) Some techniques for shading machine renderings of solids. AFIPS Conference Proc. 32 pp.37-45]. 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 – kryptografii, fizyki kwantowej, ekonomii, medycynie. 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 dowolne układy złożone z różnych procesorów (m. in. CPU, GPU, DSP, FPGA), oraz zapewnia przenośność programów.

3.3 Środowisko OpenCL

OpenCL 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.

Aplikacja pisana w OpenCL opiera się na jednostkach zwanych platformami. Każda platforma odpowiada za przygotowanie danych do obliczeń i rozdzieleniem zadań. W nowszych standardach kod platformy może być pisany w wielu popularnych jezykach (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).

Rysunek z architekturą programu OpenCL

Elementy przetwarzania wykonują specjalnie przygotowane funkcje (tzw. kernele) definiowane według standardu OpenCL. W zależności od urządzenia (karta graficzna, procesor itp.), CU i PE reprezentują różne elementy procesora, ale odwołuje się do nich w ten sam sposób. Umożliwia to wykonywanie kernela o jednakowej składni na różnych typach procesorów. Kernele wykonywane są według kolejki zadań i rozdzielone pomiędzy jednostki obliczeniowe mogą być wykonywane równolegle na wielu elementach przetwarzania. Każda kolejka zadań definiuje przestrzeń indeksowania, w której wykonywane są poszczególne kernele. Instancje kernela wywołaną w obrębie jednego indeksu nazywamy wątkiem. Przestrzeń indeksów można podzielić na grupy (ang. work-group), do których należą poszczególne wątki.

OpenCL umożliwia dynamiczny dostęp do pamięci urządzeń. Zdefiniowane są cztery typy pamięci:  
- pamięć globalna – pamięć dostępną przez wszystkie wątki w obrębie jednego kontekstu  
- pamięć stałą – pamięć tylko do odczytu, dostępna globalnie  
- pamięć lokalną – pamięć dostępna w obrębie jednej grupy wątków  
- pamięć prywatną – pamięć dostępna w obrębie jednego wątku  
Przepływ danych między platformą, a urządzeniem może odbyć się poprzez operację kopiowania, lub odwzorowania z pamięci platformy do pamięci urządzenia.

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

4. Realizacja założeń projektowych

4.1 Wprowadzenie

Promowany w latach 80-tych ruch wolnego oprogramowania [Josh Lerner, Mark Schankerman: *The Comingled Code: Open Source and Economic Development*. Cambridge, MA: MIT Press, 2010. [ISBN 978-0-262-01463-2](https://pl.wikipedia.org/wiki/Specjalna:Ksi%C4%85%C5%BCki/9780262014632).] 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 być 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.

4.2 Implementacja algorytmu

Aplikacja do obliczeń używa jednej platformy (kot 1), która definiuje jeden kontekst obliczeniowy i jedno urządzenie.

cl\_platform\_id\* platforms = new cl\_platform\_id[numPlatforms];

status = clGetPlatformIDs(numPlatforms, platforms, NULL);

Dane wejściowe alokowane są w pamięci po stronie platformy. Następnie tworzony jest kontekst obliczeniowy (kot 2), który przy użyciu predyrektywy CL\_PRESENT\_DEVICE wybiera jaki typ urządzenia zostanie wybrany.

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 (kot 3)

clGetContextInfo(

context,

CL\_CONTEXT\_DEVICES,

deviceListSize,

devices,

NULL);

Kolejnym krokiem jest przygotowanie kolejki zadań (kot 4). Posłuży ona do stworzenia przestrzeni indeksowania dla wątków.

commandQueue = clCreateCommandQueue(

context,

devices[0],

0,

&status);

Dane wejściowe przekazywane są do kontekstu obliczeniowego poprzez bufory (kot 5). Oddzielne bufory dla każdej danej wejściowej transferują ów dane do zdefiniowanego przez kernel miejsca w pamięci.

plABuffer = clCreateBuffer(

context,

CL\_MEM\_READ\_WRITE | CL\_MEM\_USE\_HOST\_PTR,

sizeof(cl\_float) \* walls,

plA,

&status);

Kernel przechowywany jest w postaci pliku tekstowego (załącznik A). Plik ten jest wczytywany przez platformę, a następnie budowany do obiektu (kod 6), który później będzie wywoływany przez poszczególne wątki. Kod kernela zawiera implementację metody źródeł pozornych na podstawie danych z kontekstu obliczeniowego. 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.

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 ileś)

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 (kot kot)

clSetKernelArg(

kernel,

1,

sizeof(cl\_mem),

(void \*)&plABuffer);

Następuje uruchomienie kolejki zadań i czekanie na ich wykonanie (kod kot). Po wykonaniu zadań kolejka zostaje zwolniona.

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 (kotek)

clEnqueueReadBuffer(

commandQueue,

outputBuffer,

CL\_TRUE,

0,

width \* sizeof(cl\_uint),

output,

0,

NULL,

&events[1]);

4.3 Obsługa aplikacji

Przed kompilacją programu należy przygotować kod pod urządzenia na jakim 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 w kolejnych liniach zdefiniować rząd pozornych źródeł, współrzędne punktu źródła dźwięku i punktu obserwacji, wektory normalne powierzchni odbijających, granice powierzchni dla poszczególnych osi, oraz współczynniki pochłaniania dźwięku dla każdej powierzchni (rysunek pierdyliard).

Rysunek przykładowego pliku wejściowego

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 (rysunek ), na której znajdują się powierzchnie odbijające, punkt źródła dźwięku, punkt pozornego źródła, punkt obserwacji oraz ścieżka promienia dźwiękowego.

Rysunek ze ścieżką

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

Rysunek siatki źródeł

5. Obliczenia testowe

5.1 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 5.2 autor przedstawia przykładowe wykorzystanie aplikacji do analizy pola akustycznego. W tym celu przygotowane zostały geometryczne modele pomieszczeń o zróżnicowanych wymiarach i parametrach pochłaniania dźwięku i porównane ze sobą.

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

5.2 Użycie aplikacji

5.2.1 Obliczenia na pomieszczeniach zróżnicowanych geometrycznie

Do obliczeń dla zróżnicowanych pomieszczeń zdefiniowano 3 modele geometryczne (rysunki).

Rysunek pomieszczenie 1

Rysunek pomieszczenie 2

Rysunek pomieszczenie 3

Dla danych modeli zostały wykonane wyliczenia siatek źródeł pozornych do 12 rzędu (rysunki).

Siatka 1

Siatka 2

Siatka 3

Na podstawie siatek źródeł pozornych zostały wyznaczone echogramy oraz krzywe zaniku dźwięku ( rysunki)

Dużo rysunków

5.2.2 Obliczenia na pomieszczeniach o różnych parametrach pochłaniania

Do obliczeń dla zróżnicowanych współczynników pochłaniania zdefiniowano jeden model geometryczny.

Rysunek pomieszczenia

Dla danego modelu zdefiniowano 3 różne zestawy współczynników pochłaniania (tabela 1).

Tabela 1

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

Rysunki

Różne parametry pochłaniania mają wpływ na kształt i szybkość zanikania krzywej zaniku dźwięku (rysunki).

Rysowando

5.2.3 Wizualizacja poszczególnych promieni dźwiękowych

Podczas testów aplikacji użyteczna okazała się możliwość wizualizacji ścieżki wybranego promienia dźwiękowego. ASDIJAPIOEF

5.3 Testy wydajnościowe

Testy wydajnościowe przeprowadzono na 2 różnych architekturach procesorów – CPU i GPU. Do pomiarów na CPU posłużył procesor Intel Core i5-2520M (tabela). Pomiaru przy użyciu GPU przeprowadzono na kartach Radeon R7 250X (tabela) oraz Radeon XXX (tabela).

Tabela

Tabela

Tabela

Pomiary przeprowadzono na modelu 1 z rozdziału 5.2.1 (rysunek) dla rzędów źródeł pozornych od 2 do 12. Wyniki zostały zestawione na poniższym wykresie (rysunek).

Wykres wydajności.

6. Podsumowanie

W ramach pracy napisana została aplikacja, która implementuje metodę źródeł pozornych na heterogeniczne platformy obliczeniowe. Wyniki testów wydajności z rozdziału 5.3 pokazały, że obliczenia wykonane z użyciem kart graficznych są znacznie szybsze niż na zwykłych procesorach i możliwa jest optymalizacja tej metody wykorzystując bardziej złożone platformy.

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 innych parametrów nie rozpatrzonych w pracy.  
  
KONIEC