JCSI 2 (2016) 104-107
Journal of Computer Sciences Institute

Wysłane: 2016-11-14
Przyjęte: 2016-11-21

Optymalna alokacja zasobów dźwiękowych urządzenia mobilnego na potrzeby gier

Arkadiusz Wrzos*, Jakub Smołka
Politechnika Lubelska, Instytut Informatyki, Nadbystrzycka 36B, 20-618 Lublin, Polska

Streszczenie. Urządzenia mobilne dysponują ograniczonymi zasobami. Szczególnie gry potrzebują specjalnego podejścia od programisty. Większość wzorców projektowych pozwala osiągnąć statyczne zużycie pamięci i niskie użycie procesora, jednak ciągle jest wiele sytuacji gdzie wytworzenie płynnie działającej funkcjonalności jest wymagającym zadaniem, wymagającym przeprowadzenia badań. W tej pracy autor bada optymalną alokację zasobów dźwiękowych do odtwarzania ich z minimalnym opóźnieniem w języku Java na systemie Android.

Słowa kluczowe: android; audio; java

*Autor do korespondencji.

Adres e-mails: arkadiusz.wrzos@pollub.edu.pl

Optimal allocation of mobile device sound resources for game programming purposes

Arkadiusz Wrzos*, Jakub Smołka
Institute of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

Abstract. Android is a mobile device platform, whose resources are limited. Games, and high-performance applications need special approach. Most of programming patterns allow for static memory allocation or maintaining low processor usage. Regardless, there are still many situations, in which development is demanding task, and needs much research. In this work author studied and research for efficient allocation of audio resources to play them with minimal latency in non deterministic way.

Keywords: android; audio; java

*Corresponding author.

E-mail address/addresses: arkadiusz.wrzos@pollub.edu.pl

1. Wstęp

Aktualnie dostęp do urządzeń mobilnych wysokiej wydajności jest powszechny. Prace nad architekturą ARM i układami dla smartfonów i tabletów pozwalają na dostarczanie coraz wydajniejszych urządzeń, które już teraz są w stanie realizować większość zadań typowych dla komputerów osobistych, takich jak komunikacja, multimedia, rozrywka itp.. Popularność zyskują interfejsy służące do przyłączania ich do telewizorów oraz monitorów, dzięki czemu już teraz z powodzeniem można ich używać zamiast komputera stacjonarnego w codziennych zadaniach, takich jak przeglądanie internetu, obsługa poczty czy kina domowego [1]. Statystycznie mieszkaniec Polski spędza dziennie 6,3 godzin przy komputerze, z czego częściej używa urządzeń przenośnych niż komputerów osobistych [2]. Pozwalają na to coraz wydajniejsze ogniwa elektryczne oraz procesory heterogeniczne (np. big.LITTLE [3]) z coraz mniejszym zapotrzebowaniem na prąd. Intensywny rozwój tej branży komputerowej stawia wiele nowych wymagań przed programistami. Mimo wysokiej wydajności, trzeba mieć świadomość, że zasoby ciągle są ograniczone. Zmusza to twórców oprogramowania do odpowiedniego balansowania pomiędzy maksymalnym obciążeniem procesora a minimalnym zużyciem baterii. W niektórych dziedzinach, takich jak przetwarzanie dźwięku, system android jest ciągle intensywnie rozwijany, i wymaga specjalnych podejść by osiągnąć zadowalające rozwiązania.

W pracy wykonano badania optymalnego wykorzystania wybranych narzędzi dostarczanych przez system Android, dla osiągnięcia wydajnej prezentacji multimediów. Wykorzystano w tym celu autorską aplikację rozrywkową, w której zadaniem osadzonego w kosmicznej scenerii gracza jest unikanie pocisków przeciwnika, oraz stukania w część ekranu z przeciwnikami zgodnie z rytmika podkładu muzycznego.

Rys. 1. Zrzuty ekranu z badanej gry

W powyższej aplikacji użytkownik kieruje bohaterem przesuwając go ku przeciwległym brzegom ekranu, oraz stuka w niebieskich przeciwników, którzy wylatują z przeciwnej strony.

2. Badania alokacji zasobów

Dosyć podobnym problemem zajmują się producenci narzędzi dla sekwencerów i aplikacji muzycznych SuperPowered, jednak ich rozwiązania, podobnie jak Android NDK są niskopoziomowe [4], wymagają dużego nakładu pracy przy implementacji.

Na konferencji Google I/O 2016 przedstawiono narzędzia Android Professional Audio znacznie obniżające opóźnienia dźwiękowe na wybranych urządzeniach z systemem Android [5]. Teoretycznie umożliwi to osiągnięcie opóźnień sprzętowych w odtwarzaniu dźwięku na poziomie 3,7ms co jest bardzo dobrym wynikiem, biorąc pod uwagę, że komputerowe karty dźwiękowe sięgają 2ms. Technologia jest w trakcie intensywnego rozwoju i najpewniej w kolejnych wersjach systemu stanie się standardem.

3. Badanie alokacji zasobów dźwiękowych

Gry towarzyszą komputerom od bardzo dawna, dzięki czemu programiści zdążyli już opisać bazę optymalnych podejść do najpopularniejszych problemów [6]. W aplikacji przygotowanej na potrzeby artykułu wykorzystano wiele spośród tych rozwiązań [7]. Zgodnie z nimi Aktualizacja stanu gry realizowana jest w pętli gry, która manipulując czasem potrzebnym na wykonanie cyklu reguluje liczbę klatek tak, by osiągnąć zadowalające efekty. Detekcja kolizji jest wykonana przez analizę części wspólnych prostokątnych instancji obiektów ruchomych. Animacje wykorzystują atlasy tekstur i przesuwają się pośród klatek dla aktualizowania wyświetlanej klatki. Te i inne problemy są doskonale znane programistom gier. Problem, który wymagał specjalnej analizy w tym przypadku polegał na potrzebie uzyskania ciągłej linii melodycznej złożonej z krótkich wycinków, które odgrywane byłyby w różny sposób w zależności od powodzenia gracza. Dla przykładu: odtwarzanie zapętlonej linii perkusyjnej miałoby być zastąpione odtwarzaniem zapętlonymi liniami perkusyjną i gitarową. Analiza wystąpienia dotknięcia ekranu w odpowiednim momencie powinna zapisywać sukces, który mechanizm odtwarzania muzyki powinien zinterpretować i przedstawić odpowiednie dźwięki w tle w czasie możliwie szybkim, najlepiej niezauważalnym. Poszukiwania wzorców, metod i wskazówek nie przyniosły skutku, prawdopodobnie ze względu na bardzo specyficzne zastosowanie mechanizmów odtwarzania dźwięku.

3.1. Plan badań

Wykonano serię badań porównującą wydajność podstawowych klas systemowych dostarczanych z Android SDK pozwalających na odtwarzanie dźwięku. Każda z nich została zaimplementowana i następowało dwudziestosekundowe badanie sygnału wyjściowego audio, polegające na stawianiu markerów na początku i końcu każdej próbki i mierzeniu przerwy między nimi. Badano klasy SoundPool i MediaPlayer. Odtwarzanie dźwięków było wykonywane w cyklicznych odstępach czasu, a ich zmiany również były cykliczne. Program wykonywał zadanie w osobnym wątku, i nie posiadając spowalniającego sprzężenia z wątkiem głównym mógł być oddelegowany do osobnego rdzenia procesora [8]. Dla porównania wykonano sprawdzenie odtworzenia pojedynczej próbki w zapętleniu. Badanie przeprowadzono na urządzeniach OnePlus One i Sony E4g. Próby miały sprawdzić, czy istnieje możliwość wysokopoziomowego wykorzystania klas SoundPool i MediaPlayer do naprzemiennego odtwarzania czterech próbek dźwiękowych z niską latencją.

Badania przewidywały sześć scenariuszy, w których każdy był wykonywany w sześciu krokach:

1) Wygenerowaniu dźwięku na podstawie przebiegu sinusoidalnego
2) Implementacji wyzwalania kolejnych próbek dźwiękowych w wątku powtarzającym odtwarzanie w odstępie czasu równym długości próbki.
3) Instalacji i uruchomieniu programu na urządzeniu testowym
4) Przechwyceniu sygnału wyjściowego audio z urządzenia na którym uruchomiony był program
5) Oznaczenie markerami czasów początków i końców odtwarzania próbek
6) Wprowadzenie danych do arkusza kalkulacyjnego i pomiar długości opóźnień i ich sumy i średniej oraz liczby wystąpień próbek i liczby straconych próbek.

Zbadano następujące mechanizmy:

1) Użycie klasy SoundPool do odtwarzania próbek w cyklu czasowym
2) Użycie klasy SoundPool do odtwarzania próbek w cyklu czasowym, z optymalizacją. Optymalizacja polegała na ustawieniu odpowiednich flag w sterowniku urządzenia, informujących o priorytecie i sposobie wykorzystania sterownika. Dodatkowo zoptymalizowano próbki dźwiękowe tak, by miały zgodną z urządzeniem częstotliwość, tak, by nie występowała konieczność resamplingu strumienia do częstotliwości stosowanej wewnętrznie przez system i układ dźwiękowy.
3) Użycie klasy SoundPool do odtworzenia pojedynczej próbki w zapętleniu. Udostępniona jest jedynie metoda odtwarzania pojedynczej próbki w zapętleniu, nie można zmieniać ich w czasie trwania.
4) Użycie wielu instancji klasy SoundPool. Każda instancja była przeznaczona do odtwarzania pojedynczej próbki w zapętleniu. Wszystkie obiekty zostały wstępnie zatrzymane i uruchamiane naprzemiennie, tak żeby uniknąć wewnętrznego ładowania bufora dla każdej z nich występującego przy podmianie pliku-jak w punktach 1. i 2.
5) Użycie klasy MediaPlayer do odtwarzania próbek w cyklu czasowym.
6) Użycie klasy MediaPlayer do odtworzenia połączonych w jeden plik próbek, a następnie wykorzystywanie mechanizmu przesunięcia znacznika odtwarzania do czasu odpowiadającego kolejnym próbkom.

Wyniki badań przedstawiono w tabelach 1-6.

Tabela 1. Wyniki próby SoundPool - próbki wyzwalane w cyklu czasowym

Próba	1
Pożądana liczba próbek	66,667
Wystąpiło próbek	56,737
Straconych	9,93
Suma opóźnień w ms	2979
Średnie opóźnienie w ms	52,26
Długość próbki w ms	300
Długość próby w ms	20000

Tabela 2. Wyniki próby SoundPool z optymalizacją- próbki wyzwalane w cyklu czasowym

Próba	2
Pożądana liczba sampli	66,66666667
Wystąpiło sampli	65,81666667
Straconych	0,85
Suma latencji w ms	255
Średnia latencja w ms	18,21428571
Długość sampla w ms	300
Długość próby w ms	20000

Tabela 3. Wyniki próby SoundPool – pojedyncza próbka wyzwalana w zapętleniu

Próba	3
Pożądana liczba sampli	66,66666667
Wystąpiło sampli	66,66666667
Straconych	0
Suma latencji w ms	0
Średnia latencja w ms	0
Długość sampla w ms	300
Długość próby w ms	20000

Tabela 4. Wyniki próby SoundPool – kontrolowane przełączanie zapętlonych strumieni

Próba	4
Pożądana liczba sampli	66,66666667
Wystąpiło sampli	66,65
Straconych	0,01666666667
Suma latencji w ms	5
Średnia latencja w ms	0,07575757576
Długość sampla w ms	300
Długość próby w ms	20000

Tabela 5. Wyniki próby MediaPlayer – próbka wyzwalana w zapętleniu

Próba	5
Pożądana liczba sampli	66,66666667
Wystąpiło sampli	63,31333333
Straconych	3,353333333
Suma latencji w ms	1006
Średnia latencja w ms	20,95833333
Długość sampla w ms	300
Długość próby w ms	20000

Tabela 6. Wyniki próby MediaPlayer – przesuwanie znacznika odtwarzania

Próba	6
Pożądana liczba sampli	40
Wystąpiło sampli	38,696
Straconych	1,304
Suma latencji w ms	652
Średnia latencja w ms	40,75
Długość sampla w ms	500
Długość próby w ms	20000

4. Wnioski

W pracy przedstawiono zestawienia opóźnień i przerw w odgrywaniu dźwięku występujących przy dynamicznym zmienianiu jego źródła/pozycji. Do pomiarów wykorzystano różne implementacje klas systemowych.

Najgorsze wyniki napotkano przy klasie MediaPlayer, która mimo iż zgodnie z dokumentacją nie służy do odtwarzania dźwięku z niskimi opóźnieniami [9], dostarcza możliwość przesuwania markera aktualnie odtwarzanego dźwięku. Umożliwia to zmienianie dźwięków przetrzymując wszystkie w jednym pliku. Rozwiązanie okazało się bardzo niewydajne i powodowało stosunkowo duże opóźnienia zarówno przy wyzwalaniu próbki w zapętleniu (Tabela 5.), jak i podczas przesuwania markera znacznika czasowego (Tabela 6.).

Lepsze wyniki osiągnięto korzystając z klasy SoundPool, gdzie opóźnienia były niskie (Tabela 1.), a po dodatkowej konfiguracji dopasowującej parametry dźwięku do parametrów sprzętowych opóźnienia zostały praktycznie wyeliminowane (Tabela 2.). Odstępy między dźwiękami były znikome, a czasem nie występowały wcale. Kluczowym jest przygotowanie próbek spójnych z parametrami urządzenia. Interesujące jest też to, że odtwarzanie pojedynczej próbki nie tworzy opóźnień (Tabela 3.).

Metoda realizująca przełączanie zatrzymanych strumieni okazała się nieprzydatna, ponieważ nie dawało możliwości synchronizacji strumieni ze sobą. Każde opóźnienie powodowało utratę synchronizacji, a suma opóźnień spowodowała, że na koniec próby dźwięk był nie do zaakceptowania (Tabela 4.) .

Najlepszym i skutecznym sposobem było się podejście drugie, czyli „SoundPool z optymalizacją - próbki wyzwalane w cyklu czasowym„. Dostarczało ono najbardziej stabilne opóźnienia na dosyć niskim poziomie. Po wykonaniu prób zamieniono dźwięki wygenerowane na podstawie przebiegu sinusoidalnego na fragmenty linii melodycznej i według empirycznej oceny to rozwiązanie dostarczyło zadowalające wrażenia słuchowe.

Należy wziąć pod uwagę dużą fragmentację rynku urządzeń z systemem Android i pamiętać, że różnice w wydajności urządzeń poszczególnej klasy mogą dawać niejednakowe wyniki przy powyższych metodach, dlatego okazuje się, że są one wydajne na nowszych modelach. Testy odbywały się na wydajnych urządzeniach z systemem w wersji 5 (Lollipop) i wielordzeniowymi procesorami.

Ze względu na niezwykle dynamiczny rozwój systemu android i narzędzi z nim związanych większość wiedzy opiera się o internetową dokumentację techniczną, oraz nieliczne pozycje w literaturze.

Nie znaleziono publikacji naukowych powiązanych z pracą.

Literatura

[1] https://liliputing.com/2016/07/andromiums-99-superbook-turns-android-phone-laptop-crowdfunding.html Andromium's $99 Superbook turns your Android phone into a laptop (crowdfunding) [04.11.2016]

[2] http://qz.com/214307/mary-meeker-2014-internet-trends-report-all-the-slides/ - Mary Meeker's 2014 internet trends report: all the slides plus highlights [04.11.2016]

[3] https://www.arm.com/products/processors/technologies/biglittleprocessing.php - Presenting the Next Generation of Mobile Processing [04.11.2016]

[4] http://superpowered.com/superpowered-audio-sdk-for-ios-and-android About Superpowered Audio SDK for iOS, OSX and Android [04.11.2016]

[5] https://www.codechannels.com/video/Google/android/android-high-performance-audio-google-io-2016/ Android high-performance audio – Google I/O 2016 [04.11.2016]

[6] Ernest Adams: Projektowanie gier. Podstawy.Helion, Gliwice 2011

[7] Robert Nystom: Game Programming Patterns ISBN: 978-0-9905829-2-2

[8] Anders Göransson: Android. Aplikacje wielowątkowe. Techniki przetwarzania. Helion 2015

[9] https://developer.android.com/reference/android/media/MediaPlayer.html - MediaPlayer [04.11.2016]