УДК 681.3

*І.З. Ашур, А.Ю. Дорошенко*

# ВИСОКОПРОДУКТИВНИЙ ПАКЕТНИЙ ДОБУТОК МАТРИЦЬ АФІННИХ ПЕРЕТВОРЕНЬ ЗА ДОПОМОГОЮ ANDROID NDK ТА JNI

Робота містить опис високопродуктивного підходу для здійснення пакетного добутку матриць афінних перетворень за допомогою Android NDK та JNI. Розібрані основні використані техніки та особливості, проведена оцінка продуктивності підходу у порівнянні з його альтернативами і попередниками.

Ключові слова: Android NDK, Android SDK, JDK, JVM, JNI, OpenGL ES, Java, афінні перетворення.

## Вступ

Ефективний добуток матриць афінних перетворень – одна з актуальних проблем розробки додатків для мобільних платформ, що використовують високопродуктивну двовимірну комп'ютерну графіку.

Такі додатки розробляються для ігрової індустрії, програмного забезпечення візуалізації, моделювання в частині рендерингу візуальних моделей, візуальної інтерактивної взаємодії.

Одним з аспектів використання результатів даних обчислень є добуток матриць виду, моделі і проекції.

Матричне представлення використовується, зокрема, для запису афінних перетворень в комп'ютерній графіці.

Мета даної роботи – огляд результатів, отриманих за прогресом розробки високопродуктивних рішень для пакетного добутку матриць афінних перетворень у просторі Android SDK, NDK, JDK, JNI.

## 1. Аналіз предметної області

Афінні перетворення зазвичай використовуються у лінійній алгебрі для представлення лінійних перетворень, а векторна сума – для представлення паралельних перенесень. Використовуючи розширену матрицю та розширений вектор стає можливим представлення лінійного переносу та лінійних перетворень з використанням одного добутку матриць. Ця техніка потребує розширення всіх векторів додатковою «1» в кінці, а всіх матриць – додатковим рядком нулів знизу та додатковим стовпчиком – вектором переносу – справа, а також, «1» в правому нижньому кутку.

Звичайний матрично-векторний добуток завжди відображає початок координат на початок координат, а тому ніколи не може представляти лінійного переносу, в якому початок координат має бути відображеним до якоїсь іншої точки. Завдяки додаванню додаткової координати зі значенням «1» до кожного вектора, останній представляє відображення простору як підмножини простору з додатковим виміром. В цьому просторі початковий простір займає підмножину, де додаткова координата дорівнює 1. Виходячи з цього, початок координат початкового простору знаходиться в (0,0,…,0,1). Таким чином стає можливим лінійний перенос початкового простору засобами лінійного перетворення простору з більшим числом вимірів. Координати в просторі з більшим числом вимірів є прикладом однорідних координат [1, 2].

Завдяки використанню системи однорідних координат стає можливим застосування комбінації будь-якої кількості афінних перетворень одним добутком відповідних матриць. Ця властивість широко використовується в комп'ютерній графіці, засобах комп'ютерного зору та робототехніці.

Матриця виду описує представлення афінного перетворення як-то: стиснення, розтягнення, поворот, паралельний пе-

ренос, відображення та інші окремі та загальні випадки трансформацій.

Простір виду представляє результат перетворення світових координат у координати видимого простору. Видимий простір у даному випадку визначається сукупністю перетворень здвигів і поворотів сцени. Ці комбіновані перетворення представляються матрицею виду.

Сучасні системи рендерингу двовимірної графіки також оперують поняттям простору відсічення. Простір відсічення визначає видимі області сцени і, як наслідок, вершини [3].

Загальний процес використання перетворень показаний на рис. 1:

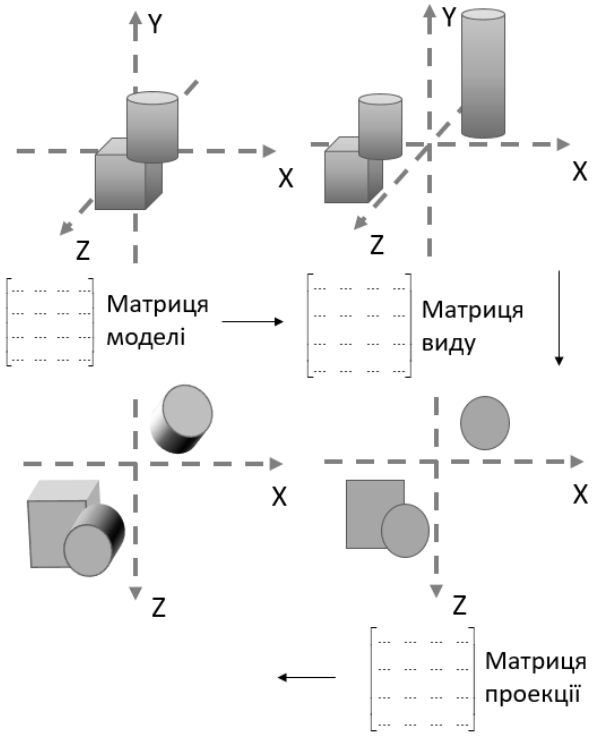


Рис. 1. Послідовність використання матричних перетворень

Деталі застосування даних трансформацій та їх математичний зміст не є предметом обговорення даної статті і приведені з ціллю надання загальної інформації про предметну область вирішуваної задачі.

Дане дослідження націлене на розробку, створення і аналіз продуктивності підходу добутку матриць афінних перетворень за допомогою Android NDK [4] та JNI [5].

## 2. Аналіз та розробка рішень

Відштовхуючись від введених обмежень предметної області у виді матриць афінних перетворень, надалі будемо оперувати матрицями розміру 4×4 [6].

Класична, наївна реалізація добутку матриць передбачає використання стандартних засобів JDK [7] з використанням трьох вкладених циклів.

```
int a[][] = {{}, {}, {}};
int b[][] = {{}, {}, {}, {}};
int al = a.length;
int bl = b[0].length;
int [][] result = new int[al][bl];

for (int i = 0; i < al; i++) {
   for (int j = 0; j < bl; j++) {
      for (int k = 0; k < al; k++) {
       result[i][j] += a[i][k] * b[k][j];
      }
   }
}
```

Для оцінки продуктивності підходу та його наївних аналогів були створені спеціальні програми-бенчмарки з урахуванням всіх «best practices». Існує багато оптимізації, що можуть бути використані віртуальною Java-машиною або апаратним забезпеченням до конкретного компонента під час його ізольованого тестування. Непродумані тести продуктивності можуть давати оптимістичні результати, що зовсім відрізняються від реальних. Наприклад, навіть порядок виконання тестів та кількість ітерацій тестування можуть створювати велику похибку в користь певного з підходів, шо тестуються.

Саме тестування конкретного компонента в складі робочого, практичного додатку з використанням таких технік, як JVM warm up (підігрів віртуальної Java-машини) [8] може дати достовірні результати.

Далі будуть приведені результати тестів продуктивності для ітерацій розроб-

леного підходу та його наївних альтернатив з різним стеком використаних технік та особливостей, їх порівняння. Під метрикою продуктивності мається на увазі час, затрачений процесором на виконання певної кількості матричних добутків.

Результати тестування затрат процесорного часу на добуток 1 мільйона матриць 4×4 у виді одновимірного масиву на 16 елементів типу з рухомою комою за використанням класичної, наївної реалізації засобами JDK на пристрої Google Pixel XL [9] в 10 ітерацій представлені в табл. 1. та рис 2.

Табл. 1. Результати тестування продуктивності наївного підходу

| Ітерація, № | Затрати процесорного часу, мс |
|---|---|
| 1 | 3121 |
| 2 | 3141 |
| 3 | 3132 |
| 4 | 3155 |
| 5 | 3137 |
| 6 | 3127 |
| 7 | 3133 |
| 8 | 3115 |
| 9 | 3111 |
| 10 | 3116 |
| Усереднений результат | 3129 |

Результати тестів для різної кількості матриць, різної кількості ітерацій, різних пристроїв та різної розмірності матриць опускаються за причини практично прямої залежності між приведеними метриками та продуктивністю обчислень за умови відсутності вузьких місць конкретної платформи для тестування.

Представлені результати отримані на пристрої з процесором Qualcomm MSM8996 Snapdragon 821 Quad-core (2x2.15 GHz Kryo & 2x1.6 GHz Kryo) [10].

Слід зазначити, що тести продуктивності проводились на збірках додатку з використанням агресивної оптимізації та обфускації сирців за використанням інструменту ProGuard [11] з сімома проходами оптимізації, що, зокрема, застосовують арифметичні оптимізації, фіналізацію класів, їх членів, вертикальне та горизонтальне склеювання класів, приватизацію полів, методів, витягнення статичних методів, оптимізацію кількості параметрів методів, вбудовування константних виразів, коротких та унікальних методів, набір методів «peephole optimization» [12] для арифметичних операцій, гілок коду, строкових даних, приведення типів, чистку надлишкових інструкцій, виключень, блоків коду.

Тема оптимізації продуктивності Java-коду на етапі компіляції достатньо вивчена на даний момент та глибоко розібрана, документована авторитетними та відомими розробниками та не потребує більш детального розгляду. Дані оптимізації застосовувалися для отримання «бойових» результатів тестування продуктивності. Маємо наступний типовий вивід процесу оптимізації:

*Removing unused program classes and class elements...*
*Original number of program classes: 8493*
*Final number of program classes: 1522*
*Optimizing...*
*# of finalized classes: 1790*
*# of unboxed enum classes: 23*
*# of privatized methods: 2100*
*# of staticized methods: 747*
*# of finalized methods: 10693*
*# of removed method parameters: 713*
*# of inlined constant parameters: 277*
*# of inlined constant return values: 103*
*# of inlined short method calls: 3242*
*# of inlined unique method calls: 4711*
*# of inlined tail recursion calls: 43*
*# of merged code blocks: 155*
*# of variable peephole optimizations: 15645*
*# of field peephole optimizations: 166*
*# of branch peephole optimizations: 5619*

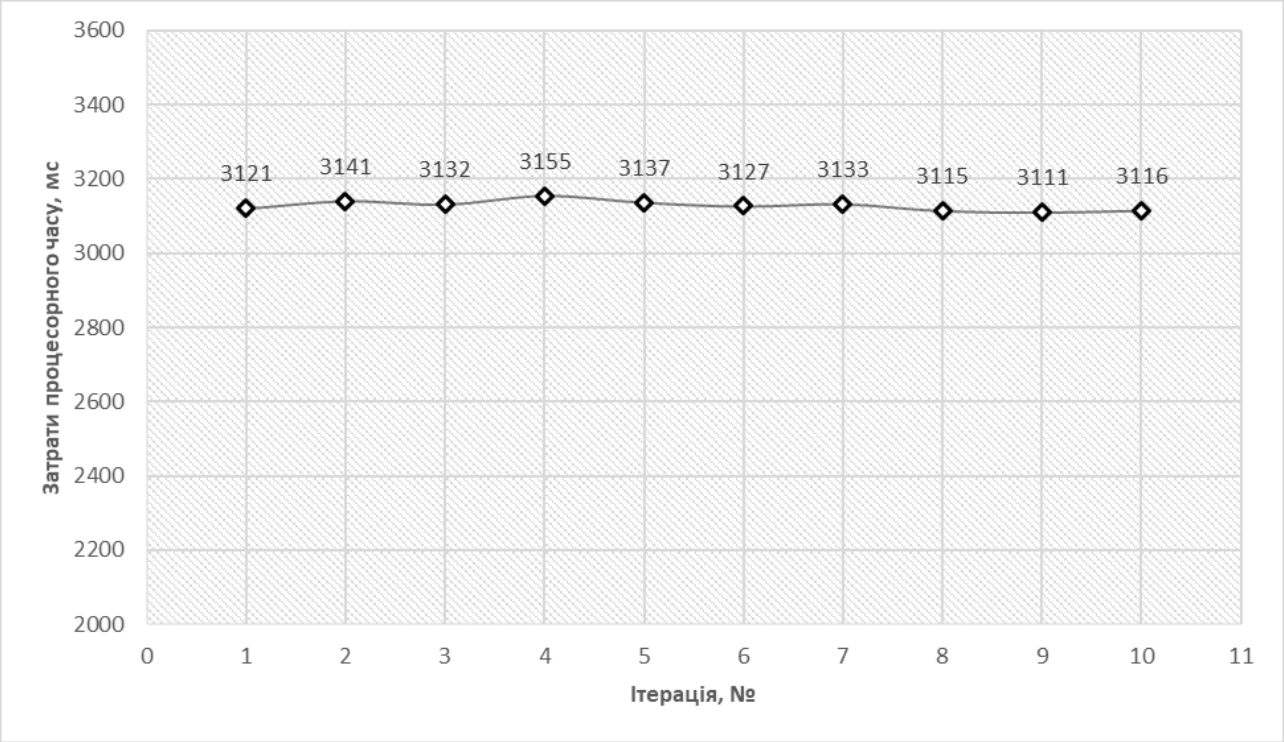


Рис. 2. Графік залежності затрат процесорного часу від ітерації тестування для наївного підходу

```
# of string peephole optimizations: 2011
# of simplified instructions: 1521
# of removed instructions: 9233
# of removed local variables: 599
# of removed exception blocks: 264
# of optimized local variable frames: 5395
Shrinking...
```

В складі Android SDK [13] присутній набір класів-застосувань для роботи з матрицями. Їх реалізація, подібно й надалі запропонованому підходу, також імплементована на рівні NDK. В цілому, вона повністю дублює зазначений вище підхід за виключенням наявності цілого ряду додаткового захисту і перевірок, інструкцій для очистки пам'яті.

Java Native Interface (JNI) є стандартним механізмом запуску C/C++ коду під керуванням віртуальної машини Java (JVM) [14].

Android NDK (Android Native Development Kit) – набір інструментів для розробки компонентів Android, базований на C/C++.

Як приклад буде приведений фрагмент зазначеної реалізації з ядра Android: /base/core/jni/android/opengl/util.cpp за станом на версію Oreo 8.0.0_r4 [15]:

```
#define I(_i, _j) ((_j)+ 4*(_i))

static
void multiplyMM(float* r,
                const float* lhs,
                const float* rhs) {
  for (int i=0 ; i<4 ; i++) {
    const float rhs_i0 = rhs[ I(i,0) ];
    float ri0 = lhs[ I(0,0) ] * rhs_i0;
    float ri1 = lhs[ I(0,1) ] * rhs_i0;
    float ri2 = lhs[ I(0,2) ] * rhs_i0;
    float ri3 = lhs[ I(0,3) ] * rhs_i0;
    for (int j=1 ; j<4 ; j++) {
      const float rhs_ij = rhs[ I(i,j) ];
      ri0 += lhs[ I(j,0) ] * rhs_ij;
      ri1 += lhs[ I(j,1) ] * rhs_ij;
      ri2 += lhs[ I(j,2) ] * rhs_ij;
      ri3 += lhs[ I(j,3) ] * rhs_ij;
    }
```

```
            r[ I(i,0) ] = ri0;
            r[ I(i,1) ] = ri1;
            r[ I(i,2) ] = ri2;
            r[ I(i,3) ] = ri3;
        }
    }
```

Результати тестів продуктивності вбудованого рішення по визначеному алгоритму тестування на ідентичному наборі вихідних даних та за ідентичного апаратного та програмного середовища (табл. 2, рис. 3):

Таблиця 2. Результати тестування продуктивності нативного вбудованого підходу

| Ітерація, № | Затрати процесорного часу, мс |
|---|---|
| 1 | 6075 |
| 2 | 5997 |
| 3 | 5987 |
| 4 | 6008 |
| 5 | 6037 |
| 6 | 6023 |
| 7 | 6005 |
| 8 | 5994 |
| 9 | 5973 |
| 10 | 6053 |
| Усереднений результат | 6015 |

Виходячи з приведених результатів можливо зробити висновок, що в даному випадку нативна реалізація даної задачі, що поставляється в пакеті Android, демонструє результати в середньому у 1,9 рази повільніші ніж наївна реалізація того ж алгоритму в просторі JDK.

Можливо виділити дві основні причини такої низької продуктивності даної реалізації. Перша – наявність накладних витрат виклику нативних функцій за використанням JNI, так званий «overhead» [16]. Друга – наявність цілого ряду додаткових перевірок вхідних даних та надлишкового копіювання даних у тимчасові буфери для проведення обчислень.

Виходячи з проведених тестів, націлених на оцінку витрат виклику JNI за використанням Google Caliper можна зробити висновок, що на цільовому апаратно-програмному забезпеченні виклик JNI функцій може займати в перспективі час еквівалентний 5-30 Java операціям над примітивними типами (табл. 3) [17, 18]:

*Scenario JniCall}*
*10.26 ns; σ=0.02 ns @ 10 trials*
*Scenario{AddIntOperation}*
*0.48 ns; σ=0.02 ns @ 10 trials*
*Scenario{AddLongOperation}*
*0.87 ns; σ=0.02 ns @ 10 trials*

Таблиця 3. Результати тестування накладних витрат виклику нативних функцій та Java-операцій над примітивними типами

| Тест | Витрати процесорного часу, нс |
|---|---|
| JniCall | 10.265 |
| AddIntOperation | 0.481 |
| AddLongOperation | 0.873 |

Таким чином, можливо запропонувати вирішення даної задачі прибираючи максимальну кількість надлишкових перевірок і вбудованих методів захисту і перевірки нативної імплементації, що допустимо на такому низькому рівні виконання, оскільки предметна область даного рішення передбачає використання у складі бойових, відлагоджених і стабільних систем, де в принципі не допустима передача не дійсних вхідних параметрів для обчислень, оскільки відповідальність за такі перевірки та препроцесінг у таких системах несуть верхні рівні, абстракції та реалізації системи.

Також можливо оптимізувати представлений наївний алгоритм добутку матриць введенням константних індексів, агресивних рівнів оптимізації компілятора нативного рішення.

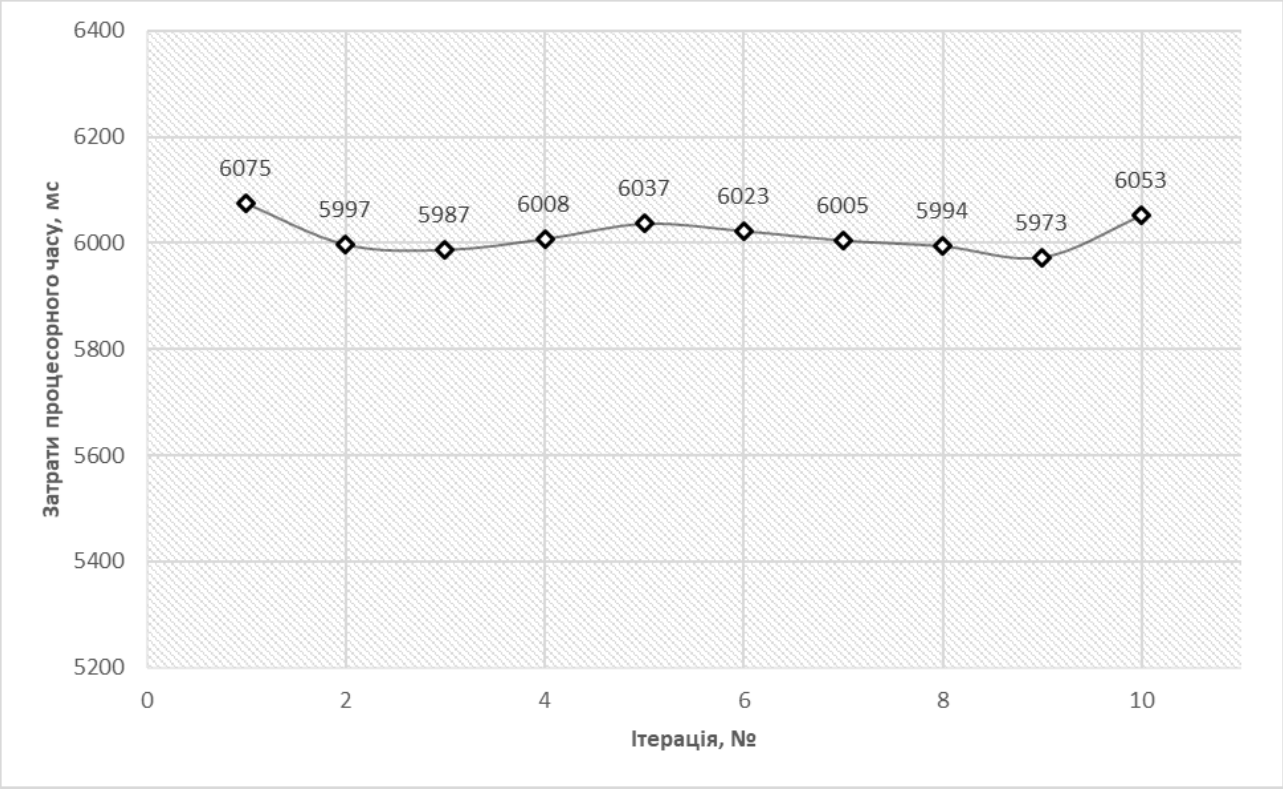


Рис. 3. Графік залежності затрат процесорного часу від ітерації тестування для нативного вбудованого підходу

Окрім цього, в контексті цих самих систем у більшості випадків існує потреба саме пакетної обробки даних. Маючи можливість отримувати результат добутку десятків, сотень матриць одночасно можливо, відповідно, багатократно скоротити накладні затрати на виклик високопродуктивних JNI функцій. Цей підхід, безперечно, потребує важкої та високопродуктивної імплементації на рівні клієнтського коду JVM, проте цілком заслуговує його.

Для імплементації запропонованого рішення використовується відносно нова підтримка роботи з NDK в складі середи розробки Android Studio. За допомогою цього інструментарію буде згенеровано скрипт ndk-build для генерації бінарних файлів, визначення правил компіляції.

Файл Application.mk [19]:

```
APP_ABI := all
APP_PLATFORM := android-16
APP_CPPFLAGS += -std=c++11
```

У даному файлі конфігурації було визначено:

підтримку генерації машинного коду для всіх підтримуваних процесорних архітектур, а саме: Hardware FPU instructions on ARMv7 based devices, ARMv8 AArch64, IA-32, Intel64, MIPS32, MIPS64 (r6), мінімальну підтримувану версію Android-платформи, підтримку стандарту C++11.

Header-файл:

```
/*
 * Method:    mul
 * Signature: (LBuffer;I[F[F)V
 */
JNIEXPORT void JNICALL
Java_com_util_math_MathUtils_mul_
_Ljava_nio_Buffer_2I_3F_3F (JNIEnv *,
jclass, jobject, jint, jfloatArray, jfloatArray);
/*
```

```
* Method:    mul
* Signature: ([FI[F[F)V
*/
JNIEXPORT void JNICALL
Java_com_util_math_MathUtils_mul_
__3FI_3F_3F (JNIEnv *, jclass, jfloatArray,
jint, jfloatArray, jfloatArray);
```

Запропонований підхід має можливість записувати результат добутку не тільки у масив, а й у сімейство класів FloatBuffer [20], що спеціально призначені для впорядкованої організації масивів примітивів у пам'яті для подальшого високопродуктивного доступу інших компонентів та модулів рендерингу до даних в буферах. В результаті імплементації запропонованих рішень вдалося отримати наступні показники продуктивності для пакетної обробки матриць пакетами по 16 [21] у перерахунку на мільйон матриць (табл. 4, рис. 4):

Таблиця 4. Результати тестування продуктивності пропонованого пакетного підходу

| Ітерація, № | Затрати процесорного часу, мс |
|---|---|
| 1 | 1056 |
| 2 | 1050 |
| 3 | 1059 |
| 4 | 1053 |
| 5 | 1050 |
| 6 | 1059 |
| 7 | 1065 |
| 8 | 1101 |
| 9 | 1143 |
| 10 | 1086 |
| Усереднений результат | 1072 |

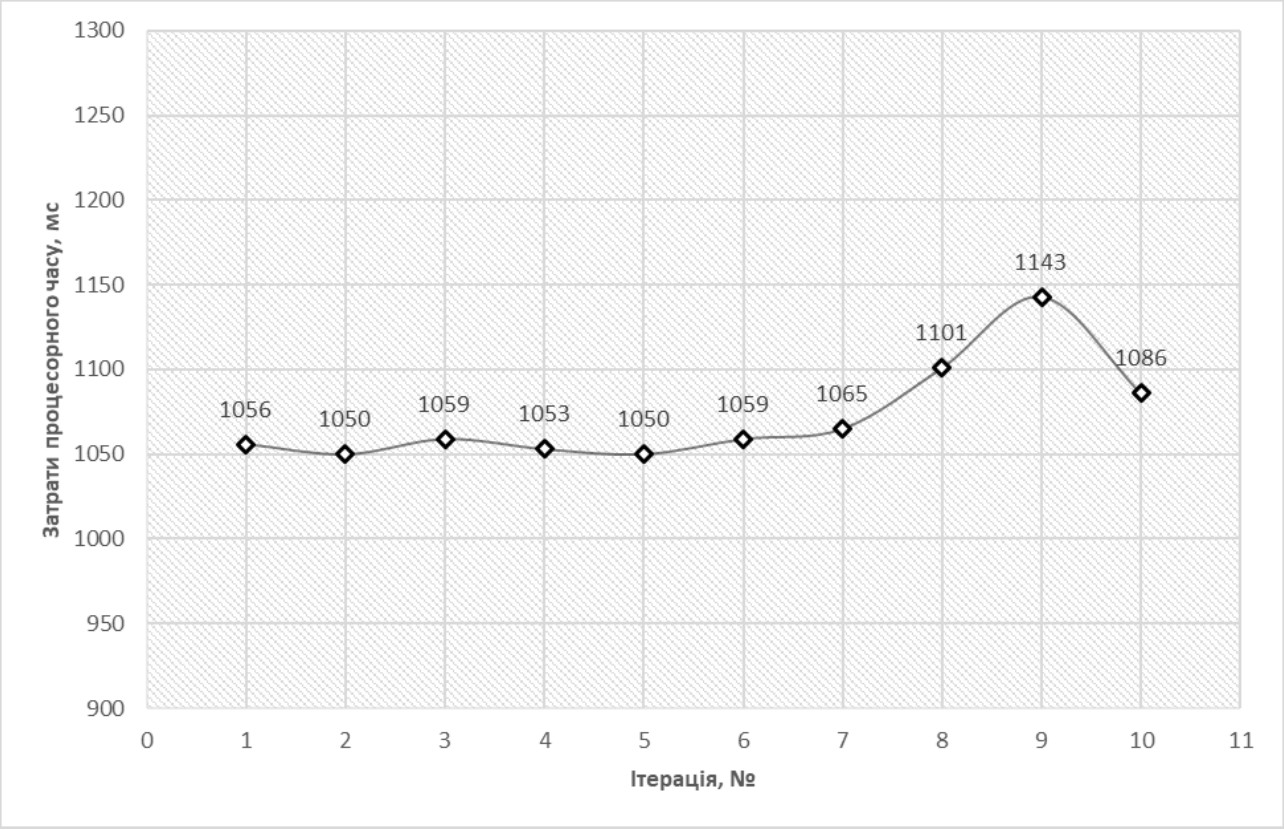


Рис. 4. Графік залежності затрат процесорного часу від ітерації тестування для пропонованого пакетного підходу

## Висновки

Таким чином, запропоноване рішення демонструє результати в середньому у 5,6 разів продуктивніші у порівнянні з нативним рішенням із складу SDK та в середньому у 2,9 разів продуктивніші у порівнянні з наївним Java-рішенням.

Таких показників, у першу чергу, вдалося досягнути за рахунок максимального скорочення накладних витрат на багатократний виклик та взаємодію з нативними функціями. Порівняльна візуалізація використаних підходів показана на рис. 5.

В перспективі пропонується вдосконалити запропоноване рішення за допомогою інтеграції більшої кількості особливостей систем предметної області, що потребують високопродуктивних обчислень на цільовій платформі, у простір NDK з попередньо вказаними спробами скоротити накладні витрати процесорного часу та провадженням пакетних обробок.

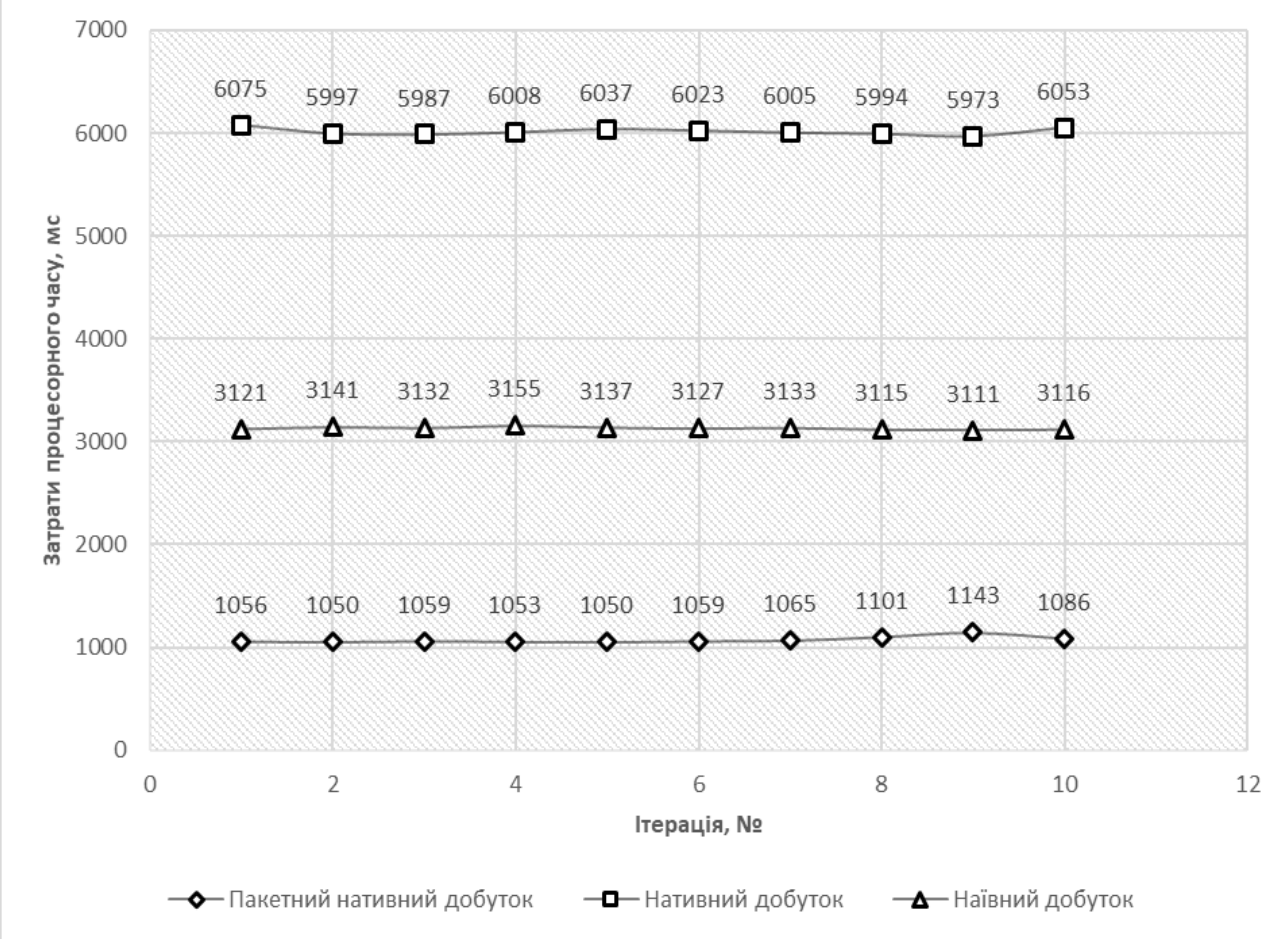


Рис. 5. Графік залежності затрат процесорного часу від ітерації тестування для всіх розібраних та запропонованих підходів

## Література


1. Marcel B. Geometry I / Berger Marcel. – Berlin: Springer, 1987.
2. Nomizu K. Affine Differential Geometry (New ed.) / K. Nomizu, S. S., 1994.
3. Shreiner D. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th Edition) / Dave Shreiner., 2016.
4. Android NDK [Електронний ресурс]. Режим доступу до ресурсу: https://developer.android.com/ndk/index.html.
5. Java Native Interface Specification [Електронний ресурс]. Режим доступу до ресурсу: https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html.
6. Article – World, View and Projection Transformation Matrices [Електронный ресурс]. Режим доступу до ресурсу: http://www.codinglabs.net/article_world_view_projection_matrix.aspx.
7. The Java® Language Specification Java SE 9 Edition [Електронний ресурс] / [J. Gosling, B. Joy, G. Steele та ін.]. Режим доступу до ресурсу: https://docs.oracle.com/javase/specs/jls/se9/html/index.html.
8. Don't Get Caught In the Cold, Warm-up Your JVM [Електронный ресурс] / [D. Lion, A. Chiu, H. Sun та ін.]. 2016. Режим доступу до ресурсу: http://www.eecg.toronto.edu/~yuan/papers/osdi16-hottub.pdf.
9. Pixel, Phone by Google. 1st generation. [Електронний ресурс] – Режим доступу до ресурсу: https://store.google.com/gb/product/pixel_phone.
10. Snapdragon 821 Mobile Platform [Електронний ресурс] – Режим доступу до ресурсу: https://www.qualcomm.com/products/snapdragon/processors/821.
11. ProGuard [Електронний ресурс] – Режим доступу до ресурсу: https://www.guardsquare.com/en/proguard.
12. McKeeman W.M. Peephole optimization. Stanford Univ., Stanford, CA. 1965. P. 443–444.
13. Android Studio The Official IDE for Android [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/index.html.
14. Lindholm T., Yellin F., Bracha G. The Java® Virtual Machine Specification Java SE 9 Edition [Електронний ресурс]. Режим доступу до ресурсу: https://docs.oracle.com/javase/specs/jvms/se9/html/index.html.
15. Cross Reference: util.cpp [Електронний ресурс]. Режим доступу до ресурсу: http://androidxref.com/8.0.0_r4/xref/frameworks/base/core/jni/android/opengl/util.cpp.
16. Dawson M., Johnson G., Low A. Best practices for using the Java Native Interface [Електронний ресурс]. 2009. Режим доступу до ресурсу: https://www.ibm.com/developerworks/library/j-jni/index.html.
17. Kurzyniec D., Sunderam V. Efficient Cooperation between Java and Native Codes – JNI Performance Benchmark [Електронний ресурс]. Режим доступу до ресурсу: http://janet-project.sourceforge.net/papers/jnibench.pdf.
18. Caliper [Електронний ресурс]. Режим доступу до ресурсу: https://github.com/google/caliper.
19. Application.mk [Електронний ресурс]. Режим доступу до ресурсу: https://developer.android.com/ndk/guides/application_mk.html.
20. FloatBuffer [Електронний ресурс]. Режим доступу до ресурсу: https://developer.android.com/reference/java/nio/FloatBuffer.html.
21. Wloka M. "Batch, Batch, Batch:" What Does It Really Mean? [Електронний ресурс]. Режим доступу до ресурсу: http://www.ce.u-sys.org/Veranstaltungen/Interaktive%20Computergraphik%20(Stamminger)/papers/BatchBatchBatch.pdf.


## References


1. Marcel B. Geometry I. Berlin: Springer, 1987.
2. Nomizu K. Affine Differential Geometry (New ed.), 1994.
3. Shreiner D. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th Edition) / Dave Shreiner., 2016.
4. Android NDK – Retrieved from https://developer.android.com/ndk/index.html.
5. Java Native Interface Specification – Retrieved from https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html.
6. Article - World, View and Projection Transformation Matrices – Retrieved from http://www.codinglabs.net/article_world_view_projection_matrix.aspx.
7. The Java® Language Specification Java SE 9 Edition / [J. Gosling, B. Joy, G. Steele та ін.]

– Retrieved from https://docs.oracle.com/javase/specs/jls/se9/html/index.html.
8. Don't Get Caught In the Cold, Warm-up Your JVM / [D. Lion, A. Chiu, H. Sun та ін.]. – 2016. – Retrieved from http://www.eecg.toronto.edu/~yuan/papers/osdi16-hottub.pdf.
9. Pixel, Phone by Google. 1st generation. – Retrieved from https://store.google.com/gb/product/pixel_phone.
10. Snapdragon 821 Mobile Platform – Retrieved from https://www.qualcomm.com/products/snapdragon/processors/821.
11. ProGuard – Retrieved from https://www.guardsquare.com/en/proguard.
12. W. M. McKeeman. Peephole optimization / W. M. McKeeman. // Stanford Univ., Stanford, CA. – 1965. – С. 443–444.
13. Android Studio The Official IDE for Android – Retrieved from https://developer.android.com/studio/index.html.
14. Lindholm T. The Java® Virtual Machine Specification Java SE 9 Edition / T. Lindholm, F. Yellin, G. Bracha – Retrieved from https://docs.oracle.com/javase/specs/jvms/se9/html/index.html.
15. Cross Reference: util.cpp – Retrieved from http://androidxref.com/8.0.0_r4/xref/frameworks/base/core/jni/android/opengl/util.cpp.
16. Dawson M. Best practices for using the Java Native Interface / M. Dawson, G. Johnson, A. Low. – 2009. – Retrieved from https://www.ibm.com/developerworks/library/j-jni/index.html.
17. Kurzyniec D. Efficient Cooperation between Java and Native Codes – JNI Performance Benchmark / D. Kurzyniec, V. Sunderam – Retrieved from http://janet-project.sourceforge.net/papers/jnibench.pdf.
18. Caliper – Retrieved from https://github.com/google/caliper.
19. Application.mk – Retrieved from https://developer.android.com/ndk/guides/application_mk.html.
20. FloatBuffer – Retrieved from https://developer.android.com/reference/java/nio/FloatBuffer.html.
21. Wloka M. "Batch, Batch, Batch:" What Does It Really Mean? / Matthias Wloka – Retrieved from http://www.ce.u-sys.org/Veranstaltungen/Interaktive%20Computergraphik%20(Stamminger)/papers/BatchBatchBatch.pdf.

Одержано 05.12.2017

***Про авторів*:**

*Ашур Ілля Зін-Еддінович,*
магістрант, Національний технічний університет України
«КПІ імені Ігоря Сікорського»
https://orcid.org/0000-0003-2348-8777

*Дорошенко Анатолій Юхимович,*
доктор фізико-математичних наук, професор, завідувач відділу теорії комп'ютерних обчислень Інституту програмних систем НАН України, професор кафедри автоматики і управління в технічних системах НТУУ "КПІ".
Кількість наукових публікацій в українських виданнях – понад 150.
Кількість наукових публікацій в іноземних виданнях – понад 50.
Індекс Гірша – 5.
http://orcid.org/0000-0002-8435-1451.

***Місце роботи авторів:***

Національний технічний університет України «КПІ імені Ігоря Сікорського»
03056, Київ, пр. Перемоги, 37

E-mail: ilyaachour@gmail.com,
doroshenkoanatoliy2@gmail.com