https://doi.org/10.30857/2786-5371.2024.4.6

УДК 621.3

МОСИСА А. А., ЛАТИШЕВ Я.-В. Г.
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна

ПОРІВНЯЛЬНИЙ АНАЛІЗ ВИКОРИСТАННЯ АЛГОРИТМІВ ГЕОМЕТРИЧНОЇ ТРАНСФОРМАЦІЇ ДЛЯ ПОКРАЩЕННЯ ПРОДУКТИВНОСТІ В МОБІЛЬНИХ ГРАФІЧНИХ ДОДАТКАХ

Мета. *Наукові дослідження в області мобільних технологій показують, що з кожним роком зростає потреба в оптимізації алгоритмів геометричної трансформації для підвищення продуктивності графічних додатків. Це дослідження присвячене аналізу різних методів трансформації зображень, їх впливу на продуктивність та якість відображення в мобільних додатках.*

Методика. *Виконано порівняльний аналіз різних алгоритмів геометричної трансформації для визначення найбільш ефективних для застосування у мобільних графічних додатках. Білінійна інтерполяція розглядається як основний метод, а також оцінюється використання методів на основі нейронних мереж.*

Результати. *Встановлено, що використання оптимізаційних алгоритмів геометричної трансформації зменшує час відгуку, ефективно використовує ресурси, покращує якість відображення та підвищує стабільність і надійність додатків.*

Наукова новизна. *Запропоновано новий підхід до застосування методів на основі нейронних мереж для геометричних трансформацій, що дозволяє автоматизувати процес визначення оптимальних параметрів трансформації.*

Практична значимість*. Використання запропонованих методів сприяє підвищенню продуктивності мобільних графічних додатків шляхом оптимізації часу відгуку та покращення якості зображень.*

Ключові слова*: геометрична трансформація; оптимізація продуктивності; мобільні додатки; алгоритми; математичні формули.*

Вступ. Наукові дослідження в області мобільних технологій показують, що з кожним роком зростає потреба в оптимізації алгоритмів геометричної трансформації для підвищення продуктивності графічних додатків. Це дослідження присвячене аналізу різних методів трансформації зображень, їх впливу на продуктивність та якість відображення в мобільних додатках.

В сучасному світі мобільні графічні додатки стають все більш поширеними та важливими для користувачів на різних пристроях. Їхнє успішне функціонування та ефективна робота залежать від продуктивності та якості графіки, яку вони відтворюють. Одним із ключових аспектів у покращенні продуктивності мобільних графічних додатків є оптимізація алгоритмів геометричної трансформації.

Геометричні трансформації використовуються для зміни розміру, форми, орієнтації та положення об'єктів на площині зображення. Проте, вибір оптимального алгоритму геометричної трансформації може бути складним завданням, оскільки він повинен забезпечити баланс між продуктивністю та якістю відтворення графіки на мобільному пристрої.

У цій статті ми проведемо порівняльний аналіз різних алгоритмів геометричної трансформації з метою визначення найбільш ефективних для застосування у мобільних графічних додатках. Буде розглянуто різні алгоритми геометричної трансформації, які використовуються в мобільних графічних додатках, та проаналізуємо їх вплив на продуктивність та якість зображення.

Постановка завдання. Мета цієї статті полягає в проведенні порівняльного аналізу різних алгоритмів геометричної трансформації для визначення найбільш ефективних методів,

що забезпечують оптимальну продуктивність та якість зображень у мобільних графічних додатках. Дослідження спрямоване на ліквідацію "білих плям" у загальній проблемі, зокрема, у питаннях вибору найкращих алгоритмів для використання в умовах обмежених ресурсів мобільних пристроїв.

Результати дослідження. В рамках дослідження було проведено порівняльний аналіз основних алгоритмів геометричної трансформації, що використовуються у мобільних графічних додатках, включаючи білінійну інтерполяцію та методи на основі нейронних мереж. Нижче наведено виклад основних результатів дослідження.

1. **Білінійна інтерполяція.** Цей метод використовується для згладжування зображень під час масштабування. Його основною перевагою є простота реалізації та низькі обчислювальні витрати. Формула білінійної інтерполяції дозволяє отримати інтенсивність пікселя на основі значень сусідніх пікселів, що забезпечує плавні переходи між пікселями та зменшує кількість артефактів на зображенні.

Формула білінійної інтерполяції виглядає наступним чином: $I(x,y) = (1-a)(1-b)I_{00} + a(1-b)I_{10} + (1-a)bI_{01} + abI_{11}$ де:

1) $I(x,y)$ – інтенсивність пікселя у координатах (x,y);

2) I_{00},I_{10},I_{01},I_{11} – інтенсивності сусідніх пікселів;

3) a та b – дробові частини координат (x,y), що визначають відстань до сусідніх пікселів.

```
import 'dart:io';
import 'dart:typed_data';
import 'package:image/image.dart' as img;

void main() async {
 // Завантажуємо зображення
 final inputImage = File('path_to_your_image.png').readAsBytesSync();
 final originalImage = img.decodeImage(inputImage)!;

 // Встановлюємо масштабний коефіцієнт
 final scaleFactor = 2.0;

 // Виконуємо білінійну інтерполяцію
 final scaledImage = scaleImageBilinear(originalImage, scaleFactor);

 // Зберігаємо результат
 final outputImage = File('scaled_image.png');
 outputImage.writeAsBytesSync(img.encodePng(scaledImage));
 print('Зображення успішно масштабовано та збережено як scaled_image.png');
}

img.Image scaleImageBilinear(img.Image src, double scaleFactor) {
 final width = (src.width * scaleFactor).toInt();
 final height = (src.height * scaleFactor).toInt();
 final dst = img.Image(width, height);

 for (int y = 0; y < height; y++) {
 for (int x = 0; x < width; x++) {
```

```
 final gx = x / scaleFactor;
 final gy = y / scaleFactor;

 final gxi = gx.floor();
 final gyi = gy.floor();
 final gxf = gx - gxi;
 final gyf = gy - gyi;

 final a00 = getPixel(src, gxi, gyi);
 final a10 = getPixel(src, gxi + 1, gyi);
 final a01 = getPixel(src, gxi, gyi + 1);
 final a11 = getPixel(src, gxi + 1, gyi + 1);

 final result = bilinearInterpolation(a00, a10, a01, a11, gxf, gyf);
 dst.setPixel(x, y, img.getColor(result[0], result[1], result[2], result[3]));
 }
 }
 return dst;
}
List<int> getPixel(img.Image src, int x, int y) {
 if (x < 0 || x >= src.width || y < 0 || y >= src.height) {
 return [0, 0, 0, 0];
 }
 final pixel = src.getPixel(x, y);
 return [img.getRed(pixel), img.getGreen(pixel), img.getBlue(pixel), img.getAlpha(pixel)];
}
List<int> bilinearInterpolation(List<int> a00, List<int> a10, List<int> a01, List<int> a11, double x,
double y) {
 final result = List<int>.filled(4, 0);
 for (int i = 0; i < 4; i++) {
 final r0 = a00[i] * (1 - x) + a10[i] * x;
 final r1 = a01[i] * (1 - x) + a11[i] * x;
 result[i] = (r0 * (1 - y) + r1 * y).toInt();
 }
 return result;
}
```

Опис коду:

Завантаження зображення: Зображення завантажується з файлу за допомогою File.readAsBytesSync(). Білінійна інтерполяція: Виконується білінійна інтерполяція для зміни масштабу зображення. Збереження результату: Масштабоване зображення зберігається у файл scaled_image.png.

2. **Методи на основі нейронних мереж.** Нейронні мережі є потужним інструментом для обробки зображень та виконання геометричних трансформацій. Вони дозволяють автоматизувати процес визначення оптимальних параметрів трансформації та можуть покращити якість зображення, забезпечуючи швидкий час обробки. Одним із популярних методів є використання згорткових нейронних мереж (CNN) для завдань з обробки зображень, таких як масштабування, обертання та перекручування зображень.

Даний код застосовує бібліотеку tflite для геометричних трансформацій на мові програмування Dart, який використовується фреймворком Fluter для написання кросплатформенних додатків:

```
import 'dart:io';
import 'package:tflite/tflite.dart';

void main() async {
 await loadModel();
 await processImage('path_to_your_image.png');
}

Future<void> loadModel() async {
 String? res = await Tflite.loadModel(
 model: "assets/scale_model.tflite",
 );
 print("Model loaded: $res");
}

Future<void> processImage(String imagePath) async {
 var output = await Tflite.runModelOnImage(
 path: imagePath,
 imageMean: 0.0,
 imageStd: 255.0,
 numResults: 1,
 asynch: true,
 );

 print("Processing result: $output");
}
```

Опис коду:

1. Завантаження моделі: Метод loadModel завантажує попередньо натреновану модель з папки assets.
2. Обробка зображення: Метод processImage застосовує модель до вибраного зображення та виводить результати в консоль.

Запуск коду:

1. Додайте файл моделі: Завантажте попередньо натреновану модель scale_model.tflite та розмістіть її у папці assets вашого проєкту.
2. Змініть шлях до зображення: У методі processImage змініть 'path_to_your_image.png' на шлях до вашого зображення.
3. Запустіть програму: Запустіть програму у вашому середовищі розробки.

Висновки. У рамках цього дослідження було проведено порівняльний аналіз різних алгоритмів геометричної трансформації для покращення продуктивності мобільних графічних додатків. Основними методами, які були розглянуті, стали білінійна інтерполяція та методи на основі нейронних мереж. На основі отриманих результатів можна зробити такі висновки:

1. **Білінійна інтерполяція**:
   - Відзначається простота реалізації та низькі обчислювальні витрати.

- Забезпечує плавні переходи між пікселями та зменшує кількість артефактів на зображенні.
- Проте, цей метод може мати обмежену точність при значному збільшенні зображення.

2. **Методи на основі нейронних мереж**:

- Дозволяють автоматизувати процес визначення оптимальних параметрів трансформації.
- Забезпечують високу якість зображення та швидкий час обробки.
- Використання нейронних мереж може бути ефективним для складних задач обробки зображень, таких як зміна масштабу, обертання та перекручування.

References

1. Chiou, J.-S. (2023). Investigating Effective Geometric Transformation for Image Augmentation to Improve Static Hand Gestures with a Pre-Trained Convolutional Neural Network. *Mathematics*, No. № 11 (23), Art. 4783.

2. Duchenne, O., Bach, F., Kweon, I. S., Ponce, J. (2011). A Tensor-Based Algorithm for High-Order Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, No. 33 (12), P. 2383–2395.

3. Simonyan, K., Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *Proceedings of the International Conference on Learning Representations* (ICLR).

4. Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems* (NIPS), P. 1097–1105.

5. Tuor, A., Kaplan, S., Hutchinson, B., Nichols, N., Robinson, S. (2017). Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams. In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

6. LeCun, Y., Bengio, Y., Hinton, G. (2015). Deep Learning. *Nature*, No. 521 (7553), P. 436–444.

7. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., Torralba, A. (2016). Learning Deep Features for Discriminative Localization. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), P. 2921–2929.

8. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), P. 2818–2826.

Література

1. Chiou J.-S. Investigating Effective Geometric Transformation for Image Augmentation to Improve Static Hand Gestures with a Pre-Trained Convolutional Neural Network. *Mathematics*. 2023. № 11 (23). Art. 4783.

2. Duchenne O., Bach F., Kweon I. S., Ponce J. A Tensor-Based Algorithm for High-Order Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2011. № 33 (12). P. 2383–2395.

3. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *Proceedings of the International Conference on Learning Representations* (ICLR). 2015.

4. Krizhevsky A., Sutskever I., Hinton G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems* (NIPS). 2012. P. 1097–1105.

5. Tuor A., Kaplan S., Hutchinson B., Nichols N., Robinson S. Deep Learning for Unsupervised Insider Threat Detection in Structured Cybersecurity Data Streams. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2017.

6. LeCun Y., Bengio Y., Hinton G. Deep Learning. *Nature*. 2015. No. 521 (7553). P. 436–444.

7. Zhou B., Khosla A., Lapedriza A., Oliva A., Torralba A. Learning Deep Features for Discriminative Localization. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR). 2016. P. 2921–2929).

8. Szegedy C., Vanhoucke V., Ioffe S., Shlens J., Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In: *Proceedings of the IEEE Conference on*

9. He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), P. 770–778.

10. Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv: 1704.04861.

Computer Vision and Pattern Recognition (CVPR). 2016. P. 2818–2826.

9. He K., Zhang X., Ren S., Sun J. Deep Residual Learning for Image Recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR). 2016. P. 770–778.

10. Howard A. G., Zhu M., Chen B., Kalenichenko D., Wang W., Weyand T., ... Adam H. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv: 1704.04861. 2017.

MOSISA ALEX ALEMAIKHU
Postgraduate student,
Faculty of Electronics, Department of AMES,
National Technical University of Ukraine
"Ihor Sikorsky Kyiv Polytechnic Institute",
Kyiv, Ukraine
E-mail: mosisa.alex@gmail.com

LATYSHEV YAN-VICTOR
Postgraduate student,
Faculty of Electronics, Department of AMES,
National Technical University of Ukraine
"Ihor Sikorsky Kyiv Polytechnic Institute",
Kyiv, Ukraine
E-mail: yan.latyshev@gmail.com

MOSISA A. A., LATYSHEV Ya.-V. H.
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine

COMPARATIVE ANALYSIS OF USING GEOMETRIC TRANSFORMATION ALGORITHMS TO IMPROVE PERFORMANCE IN MOBILE APPLICATIONS

Purpose. *Scientific research in the field of mobile technologies shows that every year there is an increasing need to optimize geometric transformation algorithms to improve the performance of graphic applications. This study is dedicated to the analysis of various image transformation methods, their impact on performance, and the quality of display in mobile applications.*

Methodology. *A comparative analysis of various geometric transformation algorithms was performed to determine the most effective ones for use in mobile graphic applications. Bilinear interpolation and affine transformations are considered as the primary methods, and the use of neural network-based methods is also evaluated.*

Findings. *It was found that the use of optimization algorithms for geometric transformation reduces response time, effectively uses resources, improves display quality, and increases the stability and reliability of applications.*

Originality. *A new approach to the application of neural network-based methods for geometric transformations is proposed, which allows automating the process of determining optimal transformation parameters.*

Practical value. *The use of the proposed methods contributes to the improvement of the performance of mobile graphic applications by optimizing response time and improving image quality.*

Keywords: *geometric transformation; performance optimization; mobile applications; algorithms; mathematical formulas.*