МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

КОМП'ЮТЕРНА ГРАФІКА ТА ОБРОБКА ЗОБРАЖЕНЬ

Методичні рекомендації до лабораторних робіт
для здобувачів вищої освіти спеціальності 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня

Харків
ХНЕУ ім. С. Кузнеця
2024

УДК 004.92(072.034)
К63

Укладачі: С. Г. Удовенко
О. В. Гороховатський
О. О. Передрій

Затверджено на засіданні кафедри інформатики та комп'ютерної техніки.

Протокол № 1 від 29.08.2023 р.

Самостійне електронне текстове мережеве видання

К63 **Комп'ютерна** графіка та обробка зображень [Електронний ресурс] : методичні рекомендації до лабораторних робіт для здобувачів вищої освіти спеціальності 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня / уклад. С. Г. Удовенко, О. В. Гороховатський, О. О. Передрій. – Харків : ХНЕУ ім. С. Кузнеця, 2024. – 46 с.

Подано методичні рекомендації до виконання лабораторних робіт із навчальної дисципліни, метою яких є закріплення знань і набуття практичних навичок із розроблення та застосування методів і способів оброблення зображень та комп'ютерної графіки. Для кожної лабораторної роботи визначено мету, завдання, засоби й порядок виконання, зміст звіту та контрольні запитання.

Рекомендовано для здобувачів вищої освіти спеціальності 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня.

УДК 004.92(072.034)

Вступ

Візуальний спосіб сприйняття відіграє надважливу роль у житті людини, оскільки близько вісімдесяти відсотків інформації про навколишній світ людина отримує завдяки йому. Кожного дня завдяки поширенню сучасних технологій та гаджетів користувачі інтернету створюють і завантажують на персональні комп'ютери та в мережу сотні тисяч нових фотографій і годин відео. Сучасний стан розвитку комп'ютерних технологій та інформаційних систем уже не дозволяє не тільки швидко обробляти такий обсяг даних, а й інколи навіть зберігати їх. Саме тому формування компетентностей у майбутніх професіоналів з інформаційних систем та технологій щодо принципів застосування комп'ютерної графіки, а також сучасних методів та способів ефективної обробки зображень та їх зберігання є надзвичайно важливим.

Навчальна дисципліна "Комп'ютерна графіка та обробка зображень" є обов'язковою навчальною дисципліною, яку вивчають відповідно до навчального плану підготовки здобувачів вищої освіти за спеціальністю 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня всіх форм навчання.

Мета навчальної дисципліни – формування в майбутніх фахівців з інформаційних систем і технологій системи компетентностей щодо роботи з базовими елементами комп'ютерної графіки, ефективного зберігання зображень, знання сучасних методів автоматичної обробки зображень для вирішення профільних завдань професійної діяльності, що пов'язані з обробкою візуальних образів.

Предметом навчальної дисципліни є методи й алгоритми роботи з комп'ютерною графікою та зображеннями.

Об'єктом навчальної дисципліни є процес створення, обробки та зберігання зображень.

Навчальна дисципліна "Комп'ютерна графіка та обробка зображень" ознайомлює здобувачів вищої освіти з основними поняттями комп'ютерної графіки, сферами її застосування, технічним забезпеченням, яке використовують для обробки графічних образів. Увагу також приділено ефективному зберіганню зображень у різних графічних форматах, методам і способам аналізу та обробки зображень і реалізації відповідних методів.

Методичні рекомендації містять п'ять лабораторних робіт, які ознайомлюють здобувачів вищої освіти з форматами графічних файлів, фрактальною графікою, геометричними трансформаціями, способами фільтрації та поліпшення зображень, а також з OpenGL.

Результати навчання та компетентності, які формує навчальна дисципліна, визначено в табл. 1.

Таблиця 1

Результати навчання та компетентності, які формує навчальна дисципліна

Результати навчання	Компетентності, якими повинен оволодіти здобувач вищої освіти
ПР1	К31, К32, К36

Примітка.

ПР1. Знати лінійну та векторну алгебру, диференціальне та інтегральне числення, теорію функцій багатьох змінних, теорію рядів, диференціальні рівняння для функції однієї та багатьох змінних, операційне числення, теорію ймовірностей та математичну статистику в обсязі, необхідному для розробки та використання інформаційних систем, технологій та інфокомунікацій, сервісів та інфраструктури організації.

К31. Здатність до абстрактного мислення, аналізу та синтезу.

К32. Здатність застосовувати знання у практичних ситуаціях.

К36. Здатність до пошуку, оброблення та узагальнення інформації з різних джерел.

Лабораторна робота 1
Формати графічних файлів

Мета роботи: вивчити найпоширеніші графічні формати для зберігання зображень і навчитися вибирати формат для найбільш вдалого застосування в практичних умовах.

Теоретична частина

Стиснення інформації чи даних є найбільш ефективним з погляду обсягу збереження чи подання інформації. Виокремлюють два різновиди методів стиснення даних – із втратами та без втрат. Якщо використовують стиснення без втрат, початкові дані (зображення) можуть бути отримані зі стиснутих у повному обсязі. У разі використанням стиснення з втратами досягають кращого коефіцієнта стиснення, однак вихідні дані не можна отримати зі стиснутих абсолютно точно.

Найпростішими методами стиснення без втрат є кодування довжин серій (RLE, Run-Length Encoding), LZW (Lempel-Ziv-Welch) і алгоритм (коди) Гаффмана (Huffman). Ці методи працюють на різних принципах, але їхня загальна мета полягає в тому, щоб зменшити розмір даних, видаляючи повторювані елементи або кодуючи їх більш компактно.

RLE (Run-Length Encoding) – це найпростіший метод стиснення даних. Він працює, замінюючи послідовність однакових елементів одним символом, за яким позначено кількість повторень. Наприклад, замість послідовності abbbbbbbccdddd одержуємо 1a7b2c4d. Цей підхід добре працює з довгими серіями повторюваних величин. Його використовують у багатьох системах як один з етапів кодування.

Метод стиснення LZW (Lempel-Ziv-Welch) стискає дані шляхом пошуку однакових послідовностей (фраз) у всьому файлі та створення відповідного словника з ключами для повторюваних фраз. Метод LZW так само, як і RLE, краще діє на ділянках однорідних, вільних від шуму, кольорів. Він є набагато ефективнішим порівняно з RLE, але потребує більше часу на кодування (стиснення) та розкодування. Алгоритм LZW використовують в багатьох графічних форматах, таких як PNG, GIF і TIFF, і даних, наприклад, у форматах PDF і ZIP.

Алгоритм Гаффмана заміняє повторювані дані більш ефективними кодами. Основна ідея полягає в тому, що кожній унікальній величині

присвоюють двійковий код, довжина якого пропорційна частоті появи цих величин.

Підсумовуючи, можна зазначити, що RLE має просту реалізацію і добре підходить для даних, які містять багато повторюваних елементів. Однак його ефективність може бути низькою для даних з малою кількістю повторюваних елементів. LZW має більш складну реалізацію, але він може забезпечити більш високу ефективність для даних з різноманітними послідовностями. Алгоритм Гаффмана має найбільш складну реалізацію (із цих трьох), але він може забезпечити найвищу ефективність для будь-якого набору даних.

GIF (CompuServe Graphics Interchange Format)

GIF (Graphics Interchange Format) – це формат растрового зображення, який використовує до 256 чітких кольорів із 24-бітного діапазону RGB. Формат GIF також дає змогу об'єднувати зображення або кадри, створюючи просту анімацію, завдяки чому залишається одним із найбільш поширених в інтернеті. Його розроблено компанією CompuServe у 1987 р. Формат GIF застосовує алгоритм стиснення LZW для зменшення розміру файлу та підтримує прозорість (піксель може бути або повністю прозорим, або повністю непрозорим). Це дає змогу створювати зображення з прозорими ділянками, які дозволяють бачити фоновий малюнок.

Формат GIF використовують для створення простих анімацій. Анімація GIF складається з послідовності зображень, які відтворюються один за одним із певною частотою.

Формат GIF часто застосовують для передачі зображень в інтернеті, створення логотипів, рекламних банерів та інших графічних елементів.

Сфера застосування формату GIF – створення зображень в інтернеті, створення логотипів, анімованих зображень, рекламних банерів та інших графічних елементів.

JPEG (Joint Photographic Experts Group)

JPEG (Joint Photographic Experts Group) – це формат растрового зображення, який використовує метод стиснення з втратами для зменшення розміру файлу. Формат JPEG є одним із найпоширеніших форматів зображень у світі, який використовують для зберігання та передачі цифрових фотографій.

Формат JPEG застосовує алгоритм стиснення з втратами, який працює, розділяючи зображення на блоки 8×8 пікселів. Потім кожний блок обробляється дискретним косинусним перетворенням (DCT), яке розкладає зображення на складові частоти. Ці складові частоти потім кодуються за допомогою алгоритму RLE (Run-Length Encoding), або Гаффмана, який видаляє повторювані елементи.

JPEG краще використовувати для стиснення растрових зображень фотографічної якості, ніж логотипів або схем, адже в них більше напівтонових переходів, серед однотонних заливань з'являються небажані завади. JPEG підтримує кольорову палітру 24 біти на піксель, що цілком достатньо для збереження фотографічної якості зображення. Установлюючи потрібну якість, можна регулювати розмір стисненого файлу. Формат JPEG не підтримує прозорість.

Формат JPEG широко застосовують для зберігання та передачі цифрових фотографій, а також для зберігання інших типів зображень, таких як ілюстрації та скановані зображення.

PNG (Portable Network Graphics)

PNG (Portable Network Graphics) – це формат растрового зображення, який використовує метод стиснення без втрат для зменшення розміру файлу. Формат PNG є відкритим форматом, який не вимагає ліцензії для використання.

Формат PNG підтримує кілька алгоритмів стиснення, зокрема:

Deflate – алгоритм стиснення без втрат, який використовують для стиснення основних даних зображення.

LZMA – алгоритм стиснення без втрат, який використовують для стиснення альфа-каналів.

ZLIB – алгоритм стиснення без втрат, який використовують для стиснення додаткових даних, таких як метадані.

Глибина кольору може бути до 48 біт включно з 8-бітним альфа-каналом.

Формат PNG широко застосовують для зберігання та передачі зображень, які вимагають високої якості, таких як логотипи, ілюстрації та скановані зображення. Його також використовують для створення вебграфіки, оскільки він підтримує прозорість і альфа-канали.

TIFF (Tagged Image File Format)

TIFF (Tagged Image File Format) – це формат растрового зображення, який підтримує різні методи стиснення, зокрема стиснення без втрат

і з втратами. Формат TIFF є відкритим форматом, який не вимагає ліцензії для використання.

Формат TIFF підтримує кілька методів стиснення:

- RLE;
- LZW;
- JPEG;

CCITT Group 3/4 – метод стиснення з втратами, який використовують для сканованих зображень;

PackBits – простий метод стиснення без втрат, який працює, видаляючи повторювані елементи.

Формат TIFF підтримується практично всіма сучасними графічними програмами та браузерами. Він дозволяє зберігати кілька зображень або шарів в одному файлі (але не у вигляді анімації).

Формат TIFF широко використовують для зберігання та передачі зображень високої якості, таких як фотографії, ілюстрації та скановані зображення. Його також застосовують для створення вебграфіки, оскільки він підтримує прозорість і альфа-канали.

BMP (Windows Device Independent Bitmap)

BMP (Bitmap) – це формат растрового зображення, який використовують для зберігання даних зображення в нестисненому вигляді. Формат BMP не застосовує методів стиснення. Зображення зберігаються в нестисненому вигляді, це означає, що кожен піксель зображення кодується окремо, але стиснення RLE може бути використано. Це може призвести до великих розмірів файлів, особливо для зображень з високою роздільною здатністю. Формат BMP не підтримує прозорість, тому зображення не можуть мати прозорі ділянки; може зберігати 24 біти кольору на піксель.

Формат BMP застосовують для зберігання та передачі зображень високої якості, наприклад, фотографій. Його також використовують для створення вебграфіки, оскільки він підтримується більшістю веббраузерів.

Практична частина

Розглянемо на прикладах використання різних форматів. Спочатку зведемо в одну таблицю найпоширеніші формати разом із їхніми технічними характеристиками. Найбільше нас будуть цікавити алгоритми компресії кожного з них.

Розглянемо зображення розміру 800×600 пікселів із глибиною кольору 24 біти на піксель (рис. 1). Будемо порівнювати такі формати, як BMP, JPEG (кілька різновидів) та GIF. Стиснення такого зображення дало результати, наведені в табл. 2. Результати можуть варіюватися залежно від програмного забезпечення, яке використовували. У цьому випадку використано Adobe Photoshop.

Рис. 1. **Тестове зображення**

Таблиця 2

Розмір зображення в різних форматах

Формат і його особливості	Розмір файлу, байт
BMP	1 440 056
JPEG quality=1 standart	25 483
JPEG quality=10 standart	115 163
JPEG quality=10 optimized	98 439
GIF	241 000

Як бачимо, за однакової візуальної якості зображення формат JPEG має очевидні переваги. Це можна пояснити тим, що вихідне зображення багате на кількість плавних переходів, які добре стискаються алгоритмом JPEG.

Розглянемо інше зображення розміром 800×600, подане на рис. 2. Результати порівняння наведено в табл. 3. Як можна помітити, компресія за допомогою алгоритму JPEG погіршується порівняно з попереднім прикладом. Це пояснюється наявністю на вихідному зображенні різких переходів кольорів.

Рис. 2. **Тестове зображення**

Таблиця 3

Розмір зображення в різних форматах

Формат і його особливості	Розмір файлу, байт
BMP	1 370 000
JPEG quality=1 standart	45 000
GIF	235 000

Розглянемо останній приклад. На рис. 3 подано зображення розміром 191×138. Відповідні дані з розмірами наведено в табл. 4.

Рис. 3. **Тестове зображення**

Таблиця 4

Розмір зображення в різних форматах

Формат і його особливості	Розмір файлу, байт
BMP	77 600
JPEG quality=6 standart	32 500
JPEG quality=0 progressive	25 500
GIF	9 070

Як бачимо, для цього зображення стиснення за методом LZW набагато ефективніше, ніж JPEG. Це спричинено невеликою кількістю кольорів вихідного зображення, невеликим розміром та різкими змінами кольорів.

Одним із суттєвих недоліків формату JPEG є викривлення дрібних деталей зображення. Приклад, наведений на рис. 4, показує якість порівняння стиснення за алгоритмами JPEG quality=0 standart і JPEG quality=12 standart.

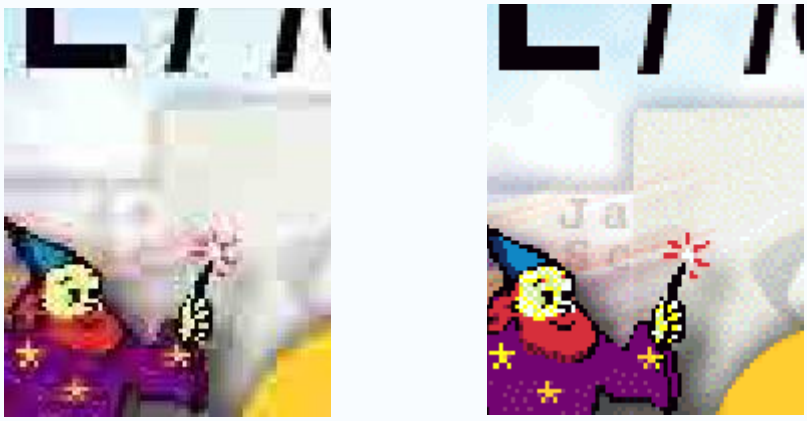


Рис. 4. **Викривлення зображення під час стиснення за допомогою JPEG**

Індивідуальні завдання

1. Вивчіть теоретичний матеріал щодо відповідних алгоритмів стиснення, використовуваних у запропонованих форматах.

Для роботи можна застосовувати будь-які програмні засоби обробки зображень, зокрема онлайн-редактори.

2. Виконайте порівняння запропонованих форматів таким чином:

- виберіть тестове зображення у форматі BMP;
- проведіть стиснення запропонованими форматами;
- поясніть переваги того чи іншого формату, за необхідності підтвердіть пояснення експериментально;
- порівняйте якість зображення після стиснення (для цього потрібно збільшити та навести фрагмент зображення в запропонованих форматах).

Початкове зображення у форматі BMP можна створити або знайти в датасетах в мережі "Інтернет".

3. Наведіть приклади зображень та їхній розмір для кожного з форматів, які будуть стиснуті найбільш ефективно відповідним форматом (табл. 5).

Таблиця 5

Варіанти завдань

Номер	Формат вихідного зображення	Формати для порівняння		
		1	2	3
1	BMP	GIF	JPEG	PNG
2	BMP	TIFF	JPEG quality=medium	GIF
3	BMP	JPEG	PNG	GIF
4	BMP	PNG	TIFF	JPEG quality=high
5	BMP	TIFF	JPEG quality=bad	GIF
6	BMP	GIF	PNG	JPEG quality=medium
7	BMP	PNG	GIF	TIFF
8	BMP	GIF	JPEG quality=bad	PNG
9	BMP	JPEG	PNG	GIF
10	BMP	GIF	TIFF	PNG
11	BMP	PNG	JPEG quality=medium	TIFF
12	BMP	GIF	PNG	JPEG quality=bad
13	BMP	TIFF	JPEG quality=medium	GIF
14	BMP	TIFF	JPEG quality=high	PNG
15	BMP	TIFF	JPEG	PNG

До завдання після погодження з викладачем (після обґрунтування причини) можуть бути внесені зміни:

- зміна вихідного формату файлу;
- зміна одного з форматів вихідних файлів (PCX, PSD, WebP та ін.) або його параметрів.

4. Зробіть висновки про доцільність використання кожного з форматів залежно від вихідного зображення.

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями; кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні копіювати мету лабораторної роботи.

Підготуйтеся до захисту роботи за контрольними запитаннями.

Контрольні запитання до лабораторної роботи 1

1. Що таке стиснення зображення? Для чого його використовують?
2. У чому полягає сутність кодування за методом RLE?
3. Як працює алгоритм стиснення Гаффмана?
4. Які критерії можуть допомогти під час вибору формату?
5. Як досягається GIF-анімація?
6. Для чого найчастіше використовують формат JPEG?
7. Що таке стиснення з втратами та без?
8. Для чого було створено GIF?
9. Для чого можна використовувати формат BMP?

Лабораторна робота 2
Фрактальна графіка

Мета роботи: вивчити визначення фракталів, їхню класифікацію і способи побудови.

Теоретична частина

Значним кроком у розробленні нових методів формування зображень є поява процедурних методів у комп'ютерній графіці, суть яких

полягає в реалізації деяких алгоритмів (процедур), у результаті чого з'являється зображення тих чи інших об'єктів. До цих методів передусім зараховують методи побудови фракталів. Фрактал визначають як об'єкт складної форми, що будується в результаті ітераційного циклу. Окремі частини фракталу схожі за формою на весь фрактал. Основи теорії фракталів створив французький математик Б. Мандельброт у 1975 р. За визначенням Б. Мандельброта, фрактал – це "груба або фрагментована геометрична форма, яку можна розділити на частини, кожна з яких (принаймні приблизно) є зменшеною копією цілого" [7].

Роль фракталів у комп'ютерній графіці досить велика. З їх допомогою, наприклад, можна задавати лінії та поверхні дуже складної форми. Фрактальна геометрія незамінна під час генерації штучних хмар, гір, поверхні моря, а також складних неевклідових об'єктів, образи яких дуже схожі на природні.

Відомі кілька різних способів класифікації фракталів. Один із найпоширеніших – за типом самоподібності.

Точні фрактали – це фрактали, які мають точну самоподібність. Це означає, що будь-яка частина фракталу є точною копією всього фракталу. Прикладами точних фракталів є сніжинка Коха, дерево Піфагора та криві Ляпунова.

Неповні фрактали – це фрактали, які мають неповну самоподібність. Це означає, що будь-яка частина фракталу є лише приблизною копією всього фракталу. Прикладами неповних фракталів є фрактали Мандельброта і Жюліа та фрактальні хмари.

Стохастичні фрактали – це фрактали, які мають стохастичну самоподібність. Це означає, що будь-яка частина фракталу є випадковою копією всього фракталу. Прикладами стохастичних фракталів є фрактал Пеано, фрактал Серпінського.

Інший спосіб класифікації фракталів – за типом функції, яку використовують для їх створення.

Алгебраїчні фрактали – це фрактали, які створюють за допомогою алгебраїчних функцій. Прикладами алгебраїчних фракталів є фрактал Мандельброта, фрактал Жюліа і фрактал Пеано.

Геометричні фрактали – це фрактали, які створюють за допомогою геометричних операцій, таких як відбиття, поворот і масштабування. Прикладами геометричних фракталів є сніжинка Коха та дерево Піфагора.

Фрактали випадкових процесів – це фрактали, які створюють за допомогою випадкових процесів, таких як дробове відсікання і шум. Прикладами фракталів випадкових процесів є фрактали Серпінського, Бернуллі та фрактальний шум.

Класифікація фракталів є корисним способом зрозуміти різноманітність цих об'єктів та їхні різні властивості.

Є й інші класифікації фракталів, наприклад, розподіл фракталів на детерміновані (алгебраїчні та геометричні) і недетерміновані (стохастичні).

Фрактали поділяють також на регулярні та мультифрактали. Мультифрактали – це неоднорідні фрактали. Для їхньої повної характеристики потрібні не одна, а кілька фрактальних розмірностей, число яких у загальному випадку нескінченно.

Фракталом Мандельброта називають фігуру, для кожної точки якої слід виконати цикл ітерацій за формулою:

$$z_{k+1} = z_k^2 + z_0,$$

де k = 0,1, …, n.

Величини z_k – це комплексні числа, $z_k = x_k + iy_k$, причому стартові значення x_0, y_0 – координати точки зображення. Для кожної точки зображення ітерації виконують обмежену кількість разів (n) або до тих пір, доки модуль комплексного числа z_k не перевищуватиме фіксоване значення 2. Цикл ітерацій для фракталу Мандельброта зазвичай виконують у діапазоні x = (–2,2; 1,0), y = (–1,2; 1,2). Для отримання точок зображення в растрі необхідно перерахувати координати цього діапазону в піксельні координати. Результат моделювання фракталу Мандельброта зображено на рис. 5.

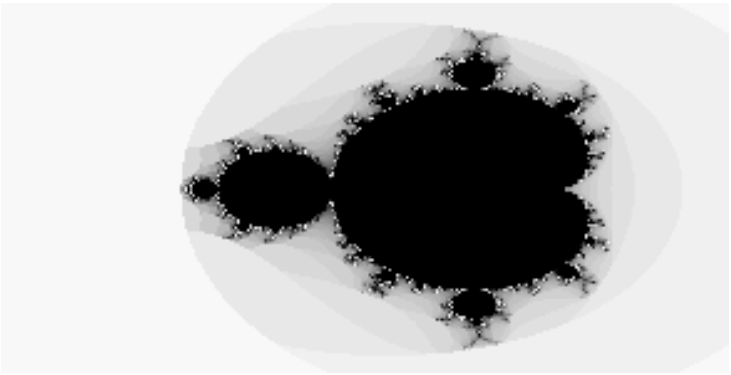


Рис. 5. **Фрактал Мандельброта**

Фрактал Жюліа (рис. 6) візуально зовсім не схожий на фрактал Мандельброта, але генерується майже аналогічним циклом:

$$z_{k+1} = z_k^2 + c,$$

де c – задана наперед комплексна константа. Умова завершення та діапазон параметрів – аналогічні фракталу Мандельброта.

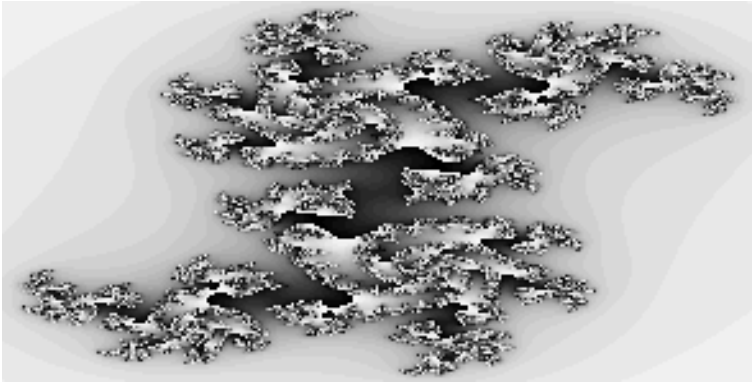


Рис. 6. **Фрактал Жюліа**

Найважливішою властивістю фракталів є подібність їхніх частин до форми всього фракталу. Крім того, форма отриманого зображення суттєво залежить від заданих параметрів фракталу.

Ще один поширений приклад – фрактал Ньютона, для якого ітераційна формула має такий вигляд:

$$z_{k+1} = (3z_k^4+1) / (4z_k^3),$$

причому початкова точка z_0 відповідає координатам точки зображення, де формується значення фракталу, а умовою зупинки ітерацій є наближення значення $|z^4-1|$ до нуля.

Значна група фракталів отримала назву геометричних фракталів, тому що їхню форму описують як послідовність деяких геометричних операцій. Наприклад, крива Коха стає фракталом в результаті ітерацій, у ході яких виконується ділення відрізка прямої на три частини. Далі кожний отриманий фрагмент знову ділиться на три частини і т. д. Поступово лінія стає схожою на сніжинку (рис. 7). Ламану лінію з чотирьох відрізків називають генератором, на кожному наступному кроці всі відрізки поточної лінії замінюють на генератор.

У загальному вигляді таку процедуру перетворення задають як окрему таблицю модельних точок, а самі перетворення застосовують спочатку до заданої базової фігури, а потім у процесі побудови – до її чергової частини. Таблиця модельних точок – це значення координат (x, y), де змінюються параметри прямої або кривої. Такі фрактали візуально схожі на острів з береговою лінією, яка здається плавною здалека, але стає нерегулярною (зрізаною) у міру наближення.

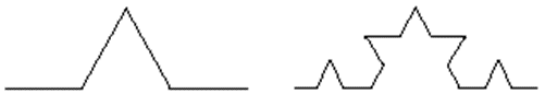


Рис. 7. **Фрактал Коха**

Зважаючи на те, що параметри фракталу можна вибирати за допомогою випадкових процесів, а також використовувати у сформованому зображенні кольори та напівтони, різноманітність форм, які задають за допомогою фракталів, можна вважати нескінченною.

Наступною важливою групою є фрактали, що генерують відповідно до методу "систем ітеративних функцій" (Iterated Function System, IFS). Цей метод описують як послідовне ітеративне обчислення координат нових точок простору відповідно до заданих виразів:

$$x_{k+1} = F_x(x_k, y_k),$$
$$y_{k+1} = F_y(x_k, y_k),$$

де F_x, F_y – функції перетворення координат, наприклад, можуть представляти афінні трансформації. Ці функції безпосередньо і визначають результуючу форму фракталу. Наприклад, зображення рослин можна сформувати послідовними геометричними перетвореннями зміщення, повороту та масштабу відрізків прямих з відповідними параметрами.

Прикладом фракталу, який можна створити із використанням IFS, є папороть Барнслі. Для його побудови використовують чотири афінні перетворення:

$$f(x,y) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

у вигляді матриці значень, де в останньому стовпці вказано ймовірність використання функції f.

w	a	b	c	d	e	f	p
f1	0	0	0	0.16	0	0	0.01
f2	0.85	0.04	–0.04	0.85	0	1.6	0.85
f3	0.2	–0.26	0.23	0.22	0	1.6	0.07
f4	–0.15	0.28	0.26	0.24	0	0.44	0.07

Приклад коду для генерації папороті та результат наведено на рис. 8.

```
const int w = 600;
const int h = 600;
var bm = new Bitmap(w, h);
var r = new Random();
double x = 0;
double y = 0;
for (int count = 0; count < 500000; count++)
{
    bm.SetPixel((int)(300 + 58 * x), (int)(58 * y), Color.ForestGreen);
    int roll = r.Next(100);
    double xp = x;
    if (roll < 1)
    {
        x = 0;
        y = 0.16 * y;
    }
    else if (roll < 86)
    {
        x = 0.85 * x + 0.04 * y;
        y = -0.04 * xp + 0.85 * y + 1.6;
    }
    else if (roll < 93)
    {
        x = 0.2 * x - 0.26 * y;
        y = 0.23 * xp + 0.22 * y + 1.6;
    }
    else
    {
        x = -0.15 * x + 0.28 * y;
        y = 0.26 * xp + 0.24 * y + 0.44;
    }
}
pictureBox1.Image = bm;
```

Рис. 8. **Папороть Барнслі**

Генерувати фрактальні зображення також можна із використанням L-systems, запропонованих А. Ліденмаєром (Aristid Lindenmayer) у 1968 р. L-системи містять три компоненти:

- алфавіт;
- аксіоми;
- правила.

Наприклад, із L-системою з алфавітом {A, B}, аксіомою {A} та правилами (A → AB), (B → A) генерують таку послідовність:

генерація 0: A;

генерація 1: AB;

генерація 2: AB A;

генерація 3: AB A AB;

генерація 4: AB A AB AB A.

Практична частина

Індивідуальні завдання

1. Реалізуйте генерацію фракталу у вигляді випадкового дерева (рис. 9). Опишіть використані параметри фракталу. Можна використати такий приклад (https://www.programmersought.com/article/76724202459/) як базовий:

```
private void Tree(PointF O, double angle, double length, float width, Graphics g)
{
    if (width < 1)
        width = 1;
    if (length < 10)//Stop drawing at almost the end of the branch
    {
        return;
    }
    PointF p = new PointF(O.X + (float)(length * Math.Cos(angle)), O.Y - (float)(length * Math.Sin(angle)));
    Pen pen = new Pen(Brushes.Red, width);
    g.DrawLine(pen, O, p);
    Tree(p, angle + Math.PI / 10, length * 0.8, width * 0.8f, g);
    Tree(p, angle - Math.PI / 10, length * 0.8, width * 0.8f, g);
}
```

```
Graphics g = e.Graphics;
PointF O = new PointF(this.panel1.Width / 2, this.panel1.Height * 0.95f);
Tree(O, Math.PI / 2, 70, 5, g);
```

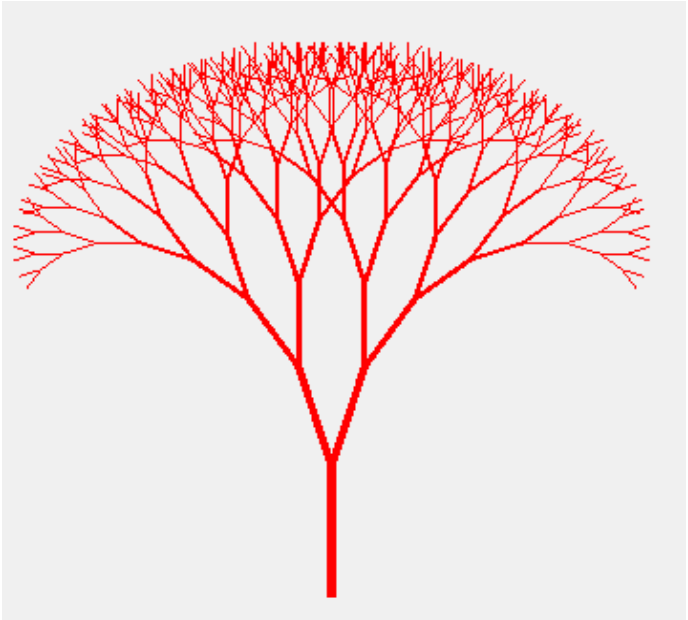


Рис. 9. **Фрактал "дерево"**

2. Реалізуйте генерацію будь-якого іншого фракталу (окрім дерева).

3. Напишіть програму для генерації трикутника Серпінського.

4*. Напишіть програму для генерації блискавок (lightning bolt effect).

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні копіювати мету лабораторної роботи.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають самостійне вивчення додаткової літератури.

Контрольні запитання до лабораторної роботи 2

1. Дайте визначення терміна "фрактал".

2. Назвіть застосування фрактальної графіки.

3. Назвіть основні види фракталів. У чому полягають особливості кожного з них?

4. Що таке геометричний фрактал і які представники цього класу вам відомі?

5. Що таке алгебраїчний фрактал і які представники цього класу вам відомі?

6*. Що таке стохастичний фрактал і які представники цього класу вам відомі?

7. Що таке генератор? У яких видах фракталів його застосовують?

8. У чому полягають принципи побудови IFS-фракталів?

9*. У чому полягає метод використання L-систем?

10*. Що таке мультифрактали? У чому їхня відмінність від регулярних фракталів?

Лабораторна робота 3
Геометричні перетворення зображень

Мета роботи: дослідити геометричні перетворення зображень, що існують; вивчити особливості застосування комбінацій перетворень; навчитися відновлювати афінну матрицю перетворень із трьома відповідними точками на початковому та перетвореному зображенні.

Теоретична частина

Розглянемо основні класи для роботи з інтерфейсом GDI+, який надає базові методи роботи з 2D-графікою для .NET. Під час виконання лабораторної роботи можна використовувати будь-яку мову програмування, зберігаючи при цьому еквівалентність виконання наведеним далі прикладам з погляду складності.

Основним простором імен, який застосовують для доступу до інструментів GDI+, є System.Drawing. Для виконання нижченаведеного прикладу достатньо створити новий застосунок типу Windows.Forms і розташувати на формі одну кнопку, яка буде все запускати.

Клас Drawings надає потужності для малювання на полотні, клас Bitmap являє собою піксельне зображення. Обидва ці класи реалізують інтерфейс IDisposable, що означає, що в кінці використання цих об'єктів необхідно викликати для них метод Dispose, який буде звільняти ресурси. Альтернативою є використання конструкції using, яка викликає цей метод автоматично.

Ось такий фрагмент коду дозволяє розмістити заздалегідь підготовлене зображення на полотні основної формі (рис. 10, 11).

```
using (Graphics g = this.CreateGraphics())
{
    g.Clear(this.BackColor);

    using (Bitmap bitmap = new Bitmap("4.2.03_resized.png"))
    {
        g.DrawImage(bitmap, 0, 0, bitmap.Width, bitmap.Height);
    }
}
```

Рис. 10. **Приклад початкового коду розміщення зображення**

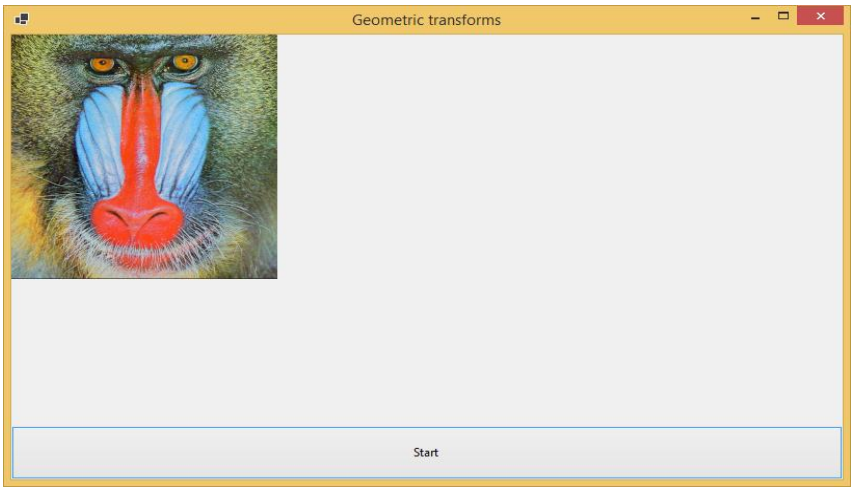


Рис. 11. **Приклад виконання початкового коду розміщення зображення**

Клас Matrix дає змогу застосовувати геометричні перетворення до об'єкту Graphics через присвоювання властивості Transform відповідного значення.

Приклад коду далі (рис. 12, 13) дозволяє перемістити зображення на 100 пікселів по горизонталі та 50 по вертикалі. Значення MatrixOrder вказує на порядок виконання операцій. Append вказує, що нове перетворення буде відбуватися після попереднього, Prepend – перед ним. Зверніть увагу, що початок координат знаходиться в лівому верхньому кутку форми і координата Y збільшується вниз.

```
using (Graphics g = this.CreateGraphics())
{
    g.Clear(this.BackColor);

    using (Bitmap bitmap = new Bitmap("4.2.03_resized.png"))
    {
        Matrix m = new Matrix();
        m.Translate(100, 50, MatrixOrder.Append);
        g.Transform = m;

       g.DrawImage(bitmap, 0, 0, bitmap.Width, bitmap.Height);
    }
}
```

Рис. 12. **Приклад коду для зміщення зображення**

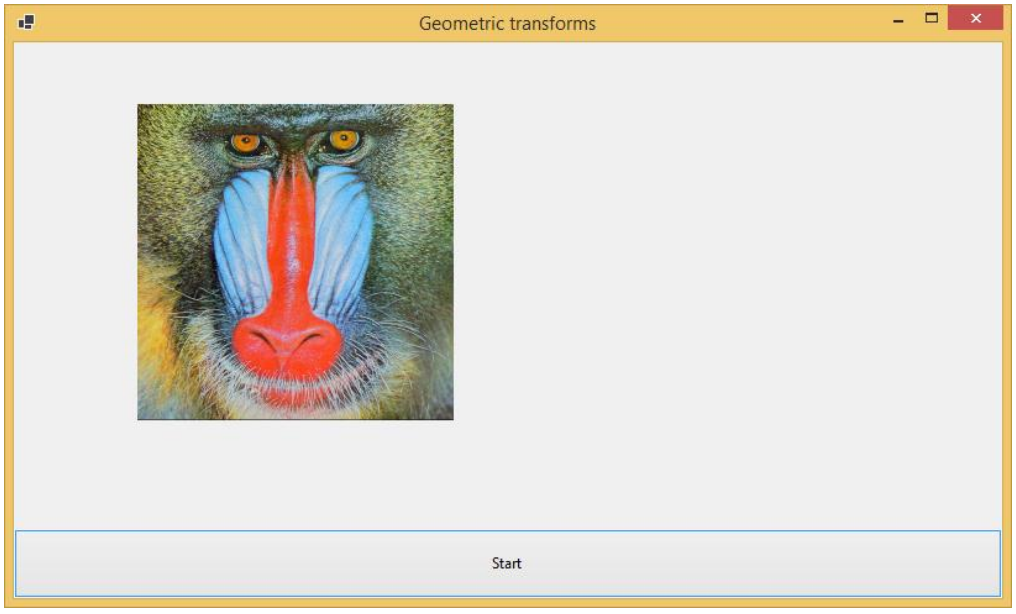


Рис. 13. **Зміщене зображення**

Наступний фрагмент коду додає масштабування після зміщення (рис. 14).

```
using (Graphics g = this.CreateGraphics())
{
    g.Clear(this.BackColor);

    using (Bitmap bitmap = new Bitmap("4.2.03_resized.png"))
    {
        Matrix m = new Matrix();

        m.Translate(100, 50, MatrixOrder.Append);
        m.Scale(0.5f, 0.5f, MatrixOrder.Append);

        g.Transform = m;

        g.DrawImage(bitmap, 0, 0, bitmap.Width, bitmap.Height);
    }
}
```

Рис. 14. **Приклад коду для зміщення та масштабування зображення**

Як можна помітити, зображення змістилося, після чого було виконано масштабування (рис. 15). Фінальна матриця перетворень матиме коефіцієнт однорідного масштабування, що дорівнює 0.5, та зміщення 50 і 25 пікселів по горизонталі та вертикалі відповідно. Значення зміщень зменшилися через вплив масштабу, який було застосовано після переміщення.

Якщо ми змінимо порядок застосування перетворень, установивши для масштабування MatrixOrder.Prepend, то спочатку буде виконано масштабування, потім – зміщення. Сформується матриця з масштабом 0.5 та зміщеннями 100 і 50 пікселів, відповідно, результуюче зображення буде відрізнятися від попереднього (рис. 16).

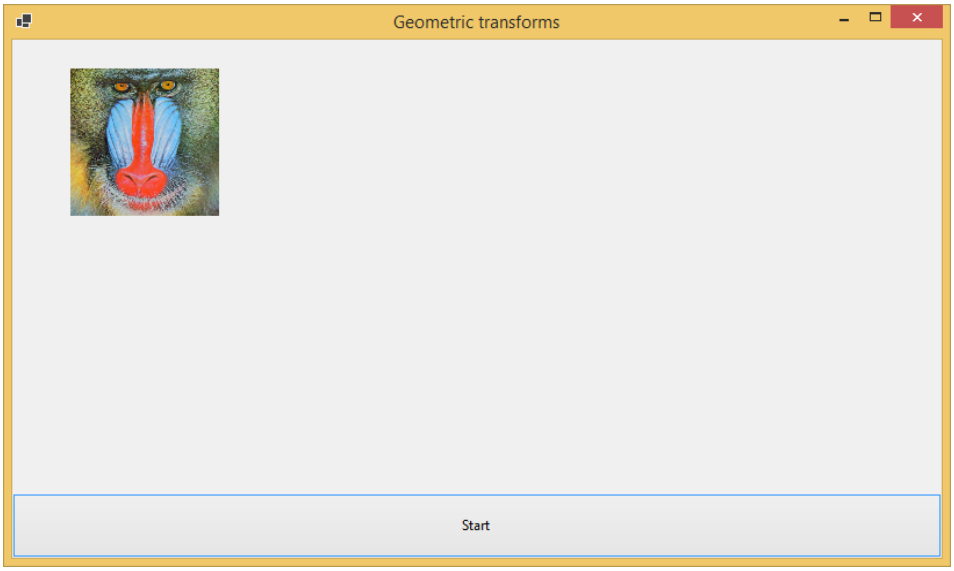


Рис. 15. **Результат застосування послідовного зміщення та масштабування**

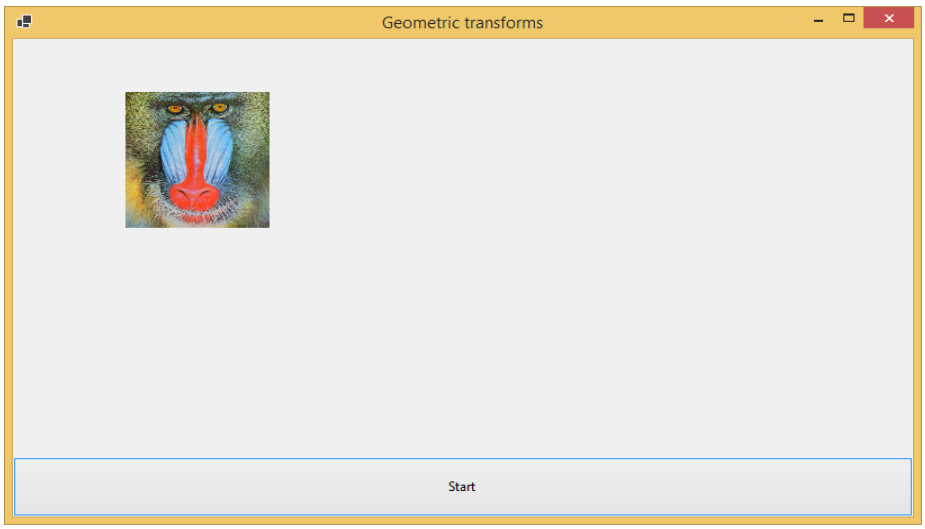


Рис. 16. **Результат застосування послідовного масштабування та зміщення**

Розглянемо наступний приклад (рис. 17, 18), який використовує комбінацію масштабування та повороту.

```
using (Graphics g = this.CreateGraphics())
{
    g.Clear(this.BackColor);

    using (Bitmap bitmap = new Bitmap("4.2.03_resized.png"))
    {
        Matrix m = new Matrix();

        m.Scale(1.5f, 1.5f, MatrixOrder.Append);
        m.Rotate(-45, MatrixOrder.Append);

        g.Transform = m;

        g.DrawImage(bitmap, 0, 0, bitmap.Width, bitmap.Height);
    }
}
```

Рис. 17. **Приклад коду для застосування масштабування та повороту**

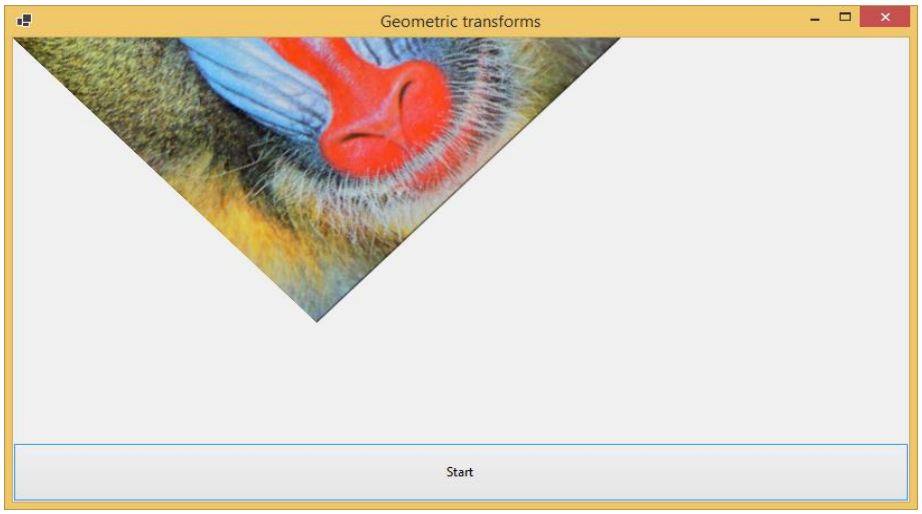


Рис. 18. **Результат виконання масштабування та повороту**

Зміна порядку перетворень із використанням Prepend у конкретному випадку не впливає на результат (рис. 19, 20).

```
using (Graphics g = this.CreateGraphics())
{
    g.Clear(this.BackColor);

    using (Bitmap bitmap = new Bitmap("4.2.03_resized.png"))
    {
        Matrix m = new Matrix();

        m.Scale(1.5f, 1.5f, MatrixOrder.Append);
        m.Rotate(-45, MatrixOrder.Prepend);

        g.Transform = m;

        g.DrawImage(bitmap, 0, 0, bitmap.Width, bitmap.Height);
    }
}
```

Рис. 19. **Приклад коду для застосування повороту та масштабування**

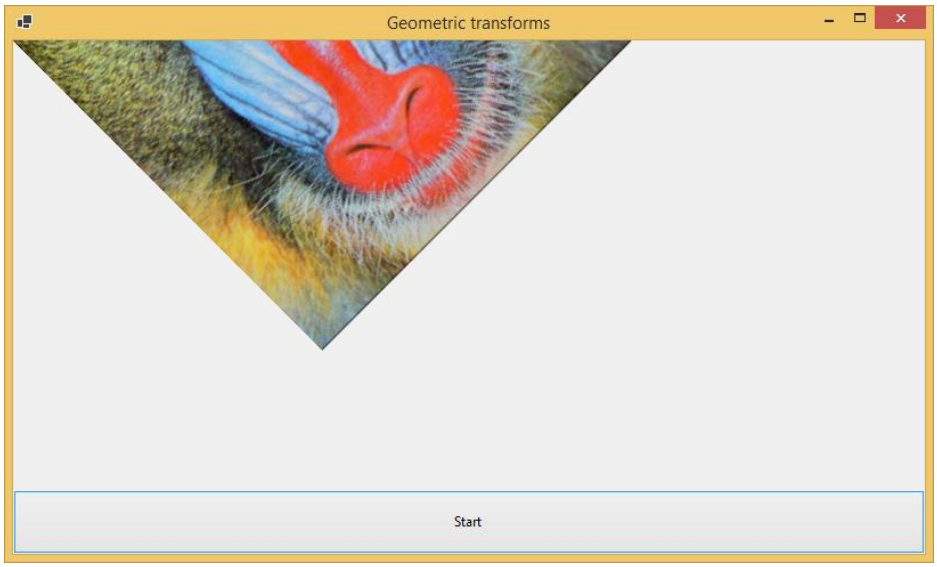


Рис. 20. **Результат виконання повороту та масштабування**

Окрім методу Rotate, існує також метод обертання навколо довільної точки RotateAt, приклад використання якого наведено на рис. 21. Зміна порядку перетворень у цьому випадку буде впливати на результат, у чому нескладно переконатися (рис. 22).

```
using (Graphics g = this.CreateGraphics())
{
    g.Clear(this.BackColor);

    using (Bitmap bitmap = new Bitmap("4.2.03_resized.png"))
    {
        Matrix m = new Matrix();

        m.Scale(1.5f, 1.5f, MatrixOrder.Append);
        m.RotateAt(-45, new PointF(10,10), MatrixOrder.Append);

        g.Transform = m;

        g.DrawImage(bitmap, 0, 0, bitmap.Width, bitmap.Height);
    }
}
```

Рис. 21. **Приклад коду із масштабуванням та поворотом навколо точки**

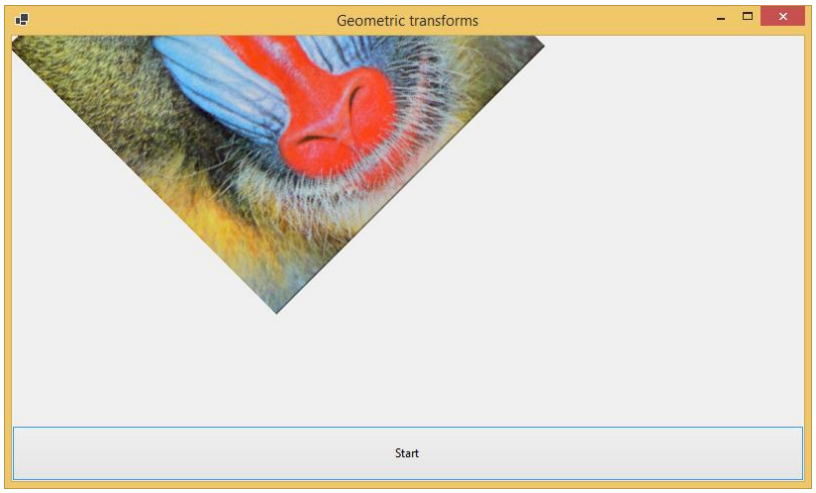


Рис. 22. **Результат виконання масштабування та повороту навколо точки**

Типовою послідовністю перетворень у комп'ютерній графіці є **масштабування, поворот і зміщення**.

Загалом виділяють декілька класів перетворень:

- ізометричні перетворення. Включають зміщення (два параметри), віддзеркалення (негативний знак перед значенням масштабування) та поворот (один параметр). Цей клас перетворень зберігає всі вихідні відстані на зображенні;
- перетворення подібності. Включають однорідне масштабування (один параметр), зміщення (два параметри) та поворот (один параметр), загалом чотири ступені свободи. Цей клас перетворень зберігає відношення подібності (similarity ratio, відношення між двома відстанями);

- афінні перетворення. Включають масштабування (два параметри масштабування, один для напрямку масштабування), зміщення (два параметри) та поворот (один параметр), загалом шість ступенів свободи. Цей клас перетворень зберігає паралелизм, відношення довжин паралельних ліній;
- проєктивні перетворення. Включають викривлення перспективи (два параметри), масштабування (три параметри), зміщення (два параметри) та поворот (один параметр), загалом вісім ступенів свободи. Цей клас перетворень зберігає подвійне відношення (cross ratio).

Практична частина

Завдання в цій лабораторній роботі виконують відповідно до індивідуальних варіантів завдань, що розташовані далі.

1. Побудуйте визначену послідовність трансформацій (за вказаними параметрами) над довільним зображенням:
   - масштабування з параметрами s_x, s_y;
   - поворот відносно точки (r_x, r_y) на кут α градусів;
   - зміщення з параметрами m_x, m_y.
2. Отримайте праве зображення з початкового, використовуючи комбінацію масштабування, повороту та зміщення в довільному порядку. Початкове розташування зображення відповідає прикладам, наведеним вище, де нульова точка розташована в лівому верхньому куті.
3. Знайдіть матрицю афінних перетворень зображення за трьома відповідними точками на початковому та перетвореному зображенні. Три початкові точки для всіх зображень: (0, 256), (0,0), (256, 0). У таблиці з варіантами вказано три відповідні точки на перетвореному зображенні.

Алгоритм знаходження точок можна взяти тут: https://staff.fnwi.uva.nl/r.vandenboomgaard/IPCV20172018/LectureNotes/MATH/homogenous.html (у розділі Estimating Parameters).

Було застосовано таку модель перетворень:

```
m.Scale(scaleX, scaleY, MatrixOrder.Append);
m.RotateAt(angle, new PointF(originX, originY), MatrixOrder.Append);
m.Translate(shiftX, shiftY).
```

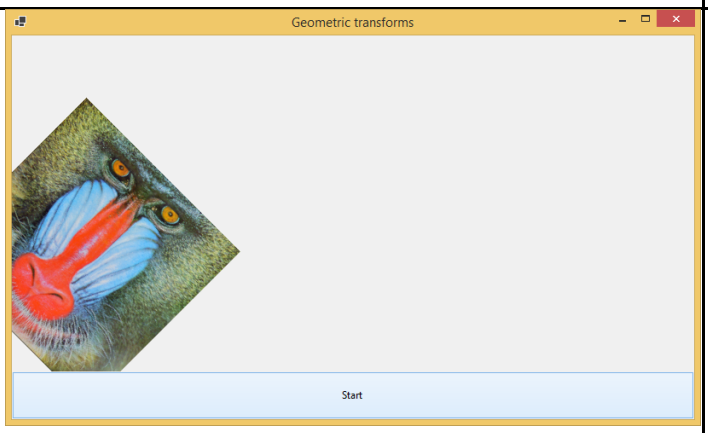
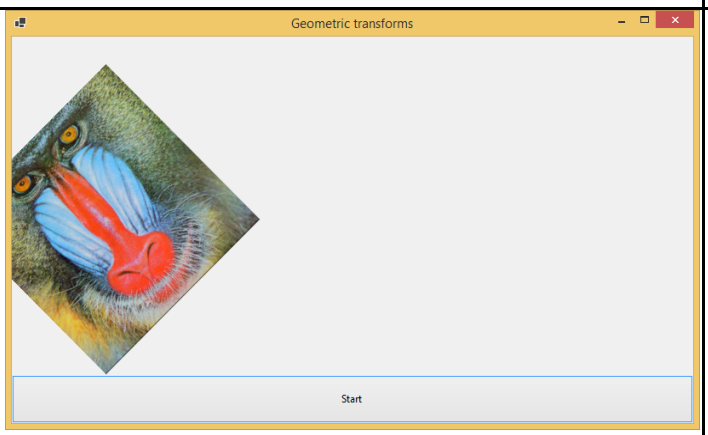
Перевірте, наскільки цей метод є чутливим до точності знаходження координат відповідних точок. Змініть несуттєво значення якоїсь точки в умові задачі, обчисліть матрицю, порівняйте з попередньою.

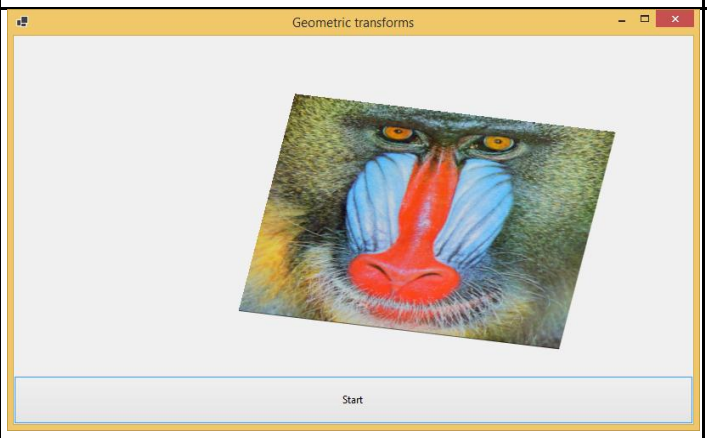
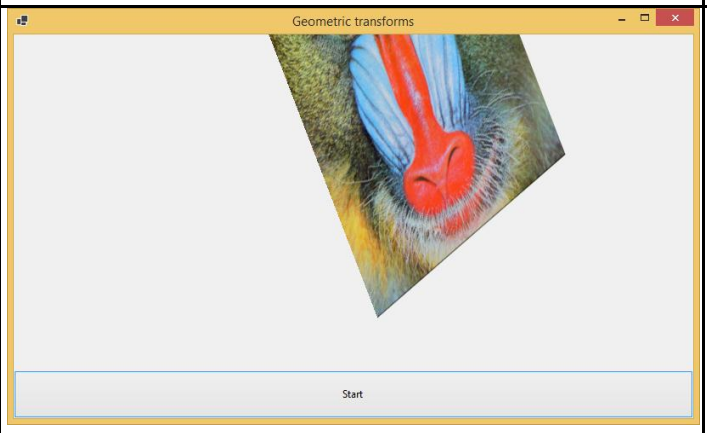
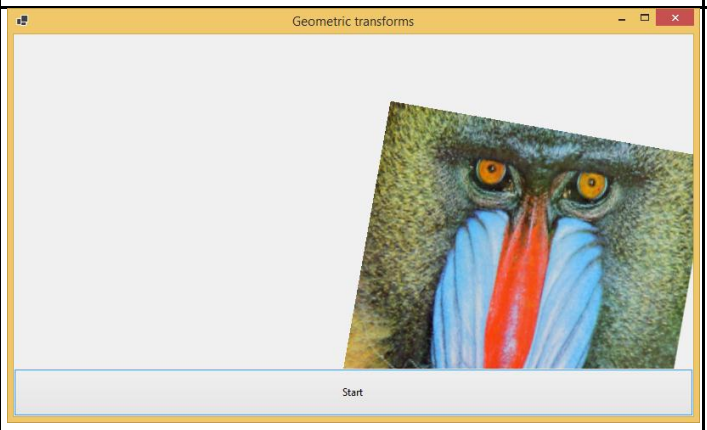
Підготуйте звіт за результатами виконаної роботи. Звіт повинен містити титульний аркуш; опис виконаних завдань (варіанти завдань подано у табл. 6) із коментарями, кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні копіювати мету лабораторної роботи.

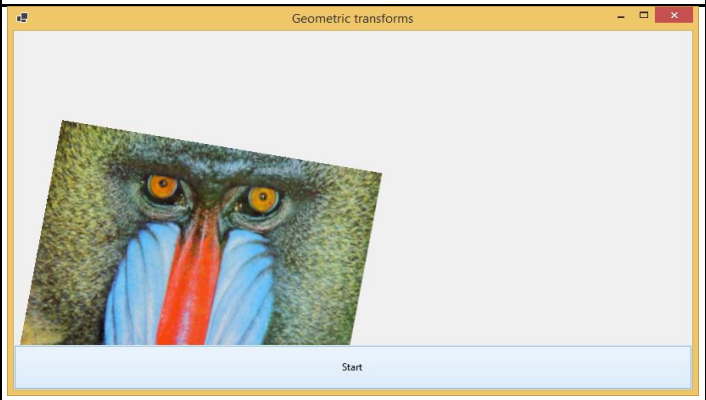
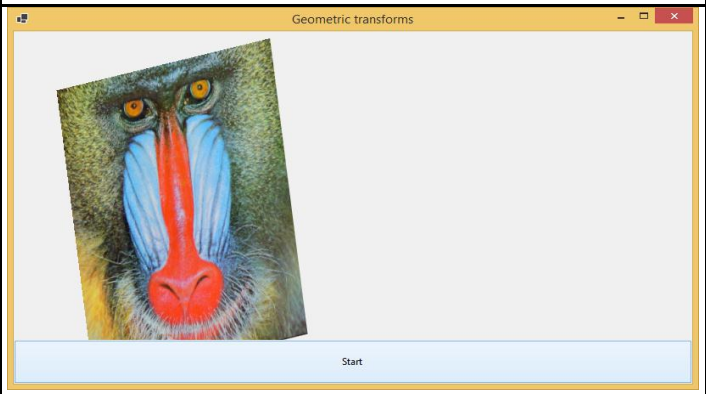
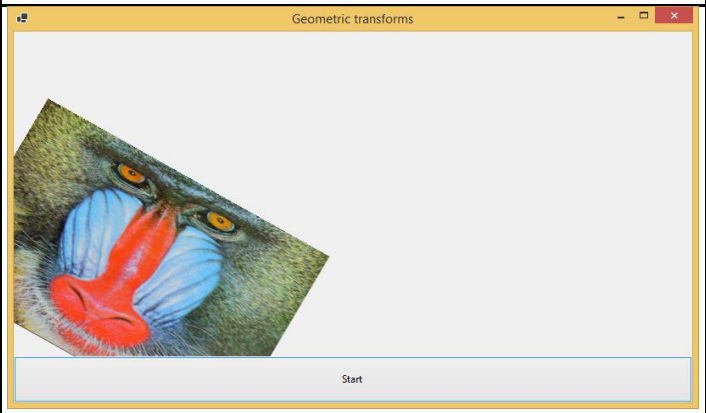
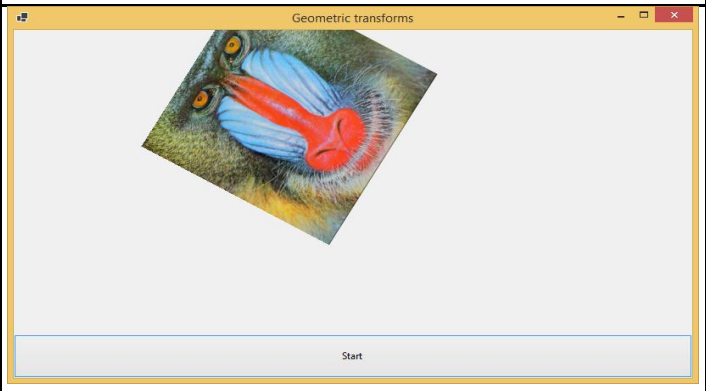
Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають самостійне вивчення додаткової літератури.

Таблиця 6

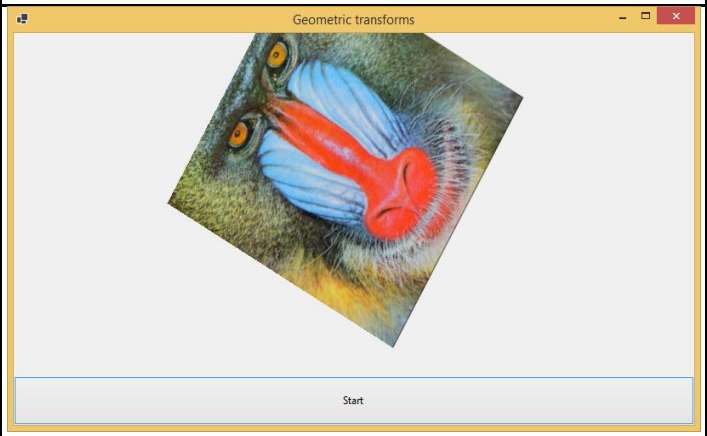
Варіанти завдань

Варіант	Завдання 1	Завдання 2	Завдання 3
1	2	3	4
1	$\mathbf{s_x} = 1$, $\mathbf{s_y} = 2$ $(\mathbf{r_x}, \mathbf{r_y}) =$ $= (20, 30)$ $\boldsymbol{\alpha} = 30$ $\mathbf{m_x} = 40$ $\mathbf{m_y} = 50$		{X = -156.05983 Y = -347.10025} {X = -255.446289 Y = 23.8153076} {X = -626.3619 Y = -75.57116}
2	$\mathbf{s_x} = 2$, $\mathbf{s_y} = 1$ $(\mathbf{r_x}, \mathbf{r_y}) =$ $= (50, 0)$ $\boldsymbol{\alpha} = -45$ $\mathbf{m_x} = 120$ $\mathbf{m_y} = 30$		{X = 248.593155 Y = -546.4561} {X = 176.343216 Y = 140.95755} {X = -129.173965 Y = 108.846466}

1	2	3	4
3	s_x= 1, s_y= 2 (r_x,r_y) = = (20, 30) **α** = 30 m_x = 40 m_y = 50		{X = 426.8151 Y = 144.2379} {X = -178.25087 Y = 250.927368} {X = -262.713379 Y = -228.08316}
4	s_x= 2, s_y= 2 (r_x,r_y) = = (0, 0) **α** = 45 m_x= 100 m_y= 50		{X = 1.94708252 Y = 694.0736} {X = 128.960754 Y = 251.124283} {X = 375.043732 Y = 321.687439}
5	s_x= 1.2, s_y= 1 (r_x,r_y) = = (50, 50) **α** = 60 m_x= -30 m_y= 50		{X = 201.448853 Y = 231.096008} {X = -278.962769 Y = 155.006348} {X = -210.882538 Y = -274.835663}

1	2	3	4
6	s_x= 1.5, s_y= 1.5 (r_x,r_y) = = (70, 80) α = -20 m_x= 0 m_y= 100		{X = -318.273163 Y = -234.358887} {X = -353.988464 Y = 276.393921} {X = -864.7413 Y = 240.6786}
7	s_x= 1.2, s_y= 1.6 (r_x,r_y) = = (20, 50) α = 20 m_x= 70 m_y= 50		{X = 394.3857 Y = 189.158234} {X = 210.9013 Y = -288.83493} {X = 784.4931 Y = -509.016235}
8	s_x= 1, s_y= 1.7 (r_x,r_y) = = (10, 10) α = 90 m_x= 0 m_y= 0		{X = 89.34436 Y = -826.412537} {X = -147.682571 Y = -461.423035} {X = -770.3117 Y = -865.7631}
9	s_x= 1.5, s_y= 1 (r_x,r_y) = = (0, 0) α = -78 m_x= 0 m_y= 50		{X = -427.335968 Y = -261.03775} {X = -187.519547 Y = 5.305382} {X = -548.9852 Y = 330.770477}

Закінчення табл. 6

1	2	3	4
10	$\mathbf{s_x}$= 2, $\mathbf{s_y}$= 1 ($\mathbf{r_x}$,$\mathbf{r_y}$) = = (-10, 0) **α** = -45 $\mathbf{m_x}$= 30 $\mathbf{m_y}$= 50		{X = 203.62677 Y = -292.4544} {X = 310.316284 Y = 312.611572} {X = -143.483154 Y = 392.6287}

Контрольні запитання до лабораторної роботи 3

1. Які геометричні перетворення існують?
2. Для чого існує параметр застосування матриці перетворень Append / Prepend?
3. Чи завжди важливий порядок геометричних перетворень?
4. Що включає група афінних перетворень?
5. Скільки параметрів характеризує групу афінних перетворень?
6. Чи можна відновити матрицю афінних перетворень? Як?
7. Чи можна відновити матрицю проєктивних перетворень? Як?
8. Чи впливає точність відповідних точок на пошук афінної матриці перетворень?
9. Які особливості зображення зберігаються незмінними після афінних перетворень?
10*. Які перетворення належать до групи перетворень rigid body?
11*. Які перетворення належать і не належать до лінійних? Чому?
12*. Що таке однорідні координати? Навіщо їх використовують?
13*. Що таке інтерполяція? Навіщо її використовують?

Лабораторна робота 4
Фільтрація та поліпшення зображень

Мета роботи: дослідити і реалізувати методи поліпшення та фільтрації зображень, що існують.

Теоретична частина

Розглянемо основні методи для обробки та фільтрації зображень. Приклади застосування цих методів наведено в матеріалах лекції.

Конвертація в градації сірого

Зображення в градаціях сірого характеризується однаковим значенням від 0 до 255 в усіх трьох каналах R, G та B. Є декілька стандартів (наборів коефіцієнтів) конвертації трьох значень окремих каналів в одне.

Бінаризація

Бінаризацією називають процес конвертації зображення в градаціях сірого в таке, яке містить лише два кольори, найчастіше, це чорний та білий. Алгоритм такої конвертації полягає в порівнянні значення яскравості кожного пікселя із заздалегідь визначеним значенням порогу. Якщо яскравість пікселя більша за порогове значення, піксель трансформується в білий (отримує значення яскравості 255), інакше – у чорний.

Згортка

Математично процес згортки зображення I(x,y) з ядром w можна записати в такому вигляді:

$$w*I(x,y) = \sum_{dx=-a}^{dx=a} \sum_{dy=-a}^{dy=a} w(dx,dy)I(x+dx, y+dy),$$

де a – половина розміру матриці з ядром.

Згортка зображення технічно полягає в ітеративному скануванні пікселів зображення матрицею з ядром, поелементному множенні значень ядра на яскравості пікселів, знаходженні суми та заміщенні центрального елемента зображення, навколо якого розташовано поточну матрицю з ядром на обчислене значення.

Матрицю з ядром неможливо розмістити повністю навколо пікселів, які знаходяться на краях зображення, тому вони обробляються окремо або не обробляються взагалі, внаслідок чого зображення зменшується ("згортається"). Для збереження пікселів на краях можна використати багато стратегій за домовленістю або залежно від вирішуваного завдання: їх можна копіювати з початкового зображення, може бути виконана якась інтерполяція, вони можуть бути залиті якимось фіксованим кольором і т. д.

Згортка зображення є універсальним методом вирішення багатьох завдань обробки зображень. Залежно від того, яке ядро використовують, згортка може знаходити контури зображення (фільтри Собела, Робертс, Превітта, компас), згладжувати (розмивати) його, додавати чіткості (sharp) або інші ефекти.

Медіанний фільтр

Медіанний фільтр становить ще один різновид алгоритму сканування зображення попіксельно матрицями різного розміру, але без використання значень ядра.

Алгоритм застосування медіанного фільтру такий:

- вибрати розмір матриці фільтрування 3×3, 5×5, 7×7 та ін.;
- для кожного пікселя вихідного зображення, навколо якого можна поставити матрицю фільтрації вибраного розміру:

виписати всі сусідні пікселі;

відсортувати їх за зростанням чи спаданням;

знайти медіану цього ряду;

замінити значення яскравості пікселя на отримане медіанне.

Згортку зображення та медіанну фільтрацію технічно слід виконувати повністю над початковим зображенням: значення яскравостей пікселів, які вже були змінені в процесі згортки/фільтрації, не повинні впливати на обробку наступних пікселів.

Практична частина

Індивідуальні завдання

Для виконання завдань можна використати шаблон програми, який уже містить методи ефективного завантаження та збереження зображень (надається додатково, мовою C#).

1. Реалізуйте конвертацію кольорового RGB-зображення в сірі тони. Поясніть вибір конкретних коефіцієнтів конвертації.

2. Реалізуйте бінаризацію зображення. Спробуйте використати різні порогові значення для одного й того самого зображення.

3. Реалізуйте згортку зображення, дослідивши її роботу з використанням ядер різного типу (три-чотири ядра для різних завдань).

4. Реалізуйте медіанний фільтр і порівняйте його роботу (щодо якості та швидкодії) для різних розмірів матриці фільтрації.

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні копіювати мету лабораторної роботи.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають самостійне вивчення додаткової літератури.

Контрольні запитання до лабораторної роботи 4

1. Чому під час конвертації в сірі кольори не використовують просто середнє значення окремих каналів R, G та B?

2. Чи можна відновити кольорову інформацію про кожен піксель після конвертації в напівтони? Чому?

3*. Чому завдання бінаризації досі є достатньо проблемним з наукового погляду?

4. Для вирішення яких завдань використовують згортку зображення?

5. Чому для згортки та медіанного фільтру зазвичай використовують квадратні матриці непарного розміру?

6. Що таке медіана? Подумайте, чи можна в медіанному фільтрі використовувати не медіану, а моду чи середнє арифметичне?

7. Якого розміру завади може прибрати медіанний фільтр залежно від розміру матриці?

8*. Для вирішення яких практичних завдань використовують бінаризацію зображення?

9*. Які загальні властивості мають матриці детектування границь? Яким чином виконують детектування границь під різними кутами?

10*. Що таке гістограма зображення? Для чого її використовують в практичних завданнях?

Лабораторна робота 5
Вступ до OpenGL

Мета роботи: дослідити приклад програмного застосунку з використанням OpenGL, вивчити особливості використання OpenGL.

Теоретична частина

Для створення прикладу будемо використовувати Visual Studio, мову C# та бібліотеку-обгортку над OpenGL, яку називають OpenTK (https://opentk.net/learn/index.html).

Відкрийте проєкт-шаблон, наданий вам із цими методичними рекомендаціями. Якщо його запустити, повинно з'явитися вікно з квадратом, наведено на рис. 23.

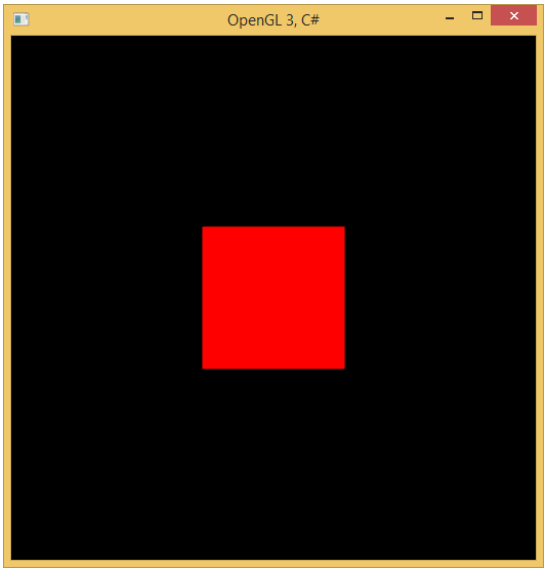


Рис. 23. **Початковий результат виконання проєкту-шаблону**

Розглянемо фрагменти коду, які вже існують.

Основний клас створює та запускає вікно із заданою частотою оновлення у фреймах на секунду й назвою "OpenGL 3, C#" (рис. 24).

```
0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        using (var window = new Window())
        {
            window.Title = "OpenGL 3, C#";
            window.Run(60);
        }
    }
}
```

Рис. 24. **Код для запуску основного вікна**

Завантаження програми починається з методу OnLoad, який створює вершинний шейдер (vShaderSource) та фрагментний шейдер (fShaderSource) (рис. 25).

```
var vShaderSource =
    @"
                #version 130

                in vec3 aPosition;
                uniform mat4 uMvpMatrix;

                void main()
                {
                    gl_Position = uMvpMatrix * vec4(aPosition, 1.0);
                }
            ";


var fShaderSource =
    @"
                #version 130
                precision mediump float;

                out vec4 fragColor;

                void main()
                {
                    fragColor = vec4(1.0,0.0,0.0,1.0);
                }
";
```

Рис. 25. **Код шейдерів**

Вершинний шейдер приймає як вхідні дані значення позиції точки aPosition і матрицю трансформацій uMvpMatrix, а заповнює системну змінну gl_Position трансформованою точкою.

Фрагментний шейдер установлює для результуючої змінної fragColor червоний колір.

Далі ці шейдери компілюються, "прикріплюються" до об'єкта OpenGL, який називають "програмою", після чого ця програма запускається (рис. 26).

```
var vShader = GL.CreateShader(ShaderType.VertexShader);
GL.ShaderSource(vShader, vShaderSource);
GL.CompileShader(vShader);

var fShader = GL.CreateShader(ShaderType.FragmentShader);
GL.ShaderSource(fShader, fShaderSource);
GL.CompileShader(fShader);

var program = GL.CreateProgram();
GL.AttachShader(program, vShader);
GL.AttachShader(program, fShader);
GL.LinkProgram(program);
GL.UseProgram(program);
```

Рис. 26. **Код компіляції та налаштування шейдерів**

У наступному рядку створюються об'єкти сцени та підраховується їхня кількість:

_amountOfVertices = InitVertexBuffers(program).

Розглянемо метод InitVertexBuffers. На початку створюються вершини, нормалі та грані об'єкта, який є кубом (тож насправді той плаский квадрат, який ми бачимо під час запуску є кубом).

Масив vertices містить координати вершин (по три значення підряд) для кожної з шести граней.

Масив normals зберігає вектори нормалей до кожної з вершин аналогічним чином.

Масив indices містить номери вершин для кожної з шести квадратних граней, кожна з яких складається з двох трикутників (рис. 27).

```
// Create a cube
//    v6----- v5
//   /|      /|
//  v1------v0|
//  | |     | |
//  | |v7---|-|v4
//  |/      |/
//  v2------v3

float[] vertices = new float[]
{
   1f, 1f, 1f, -1f, 1f, 1f, -1f, -1f, 1f, 1f, -1f, 1f,      // v0-v1-v2-v3 front
   1f, 1f, 1f, 1f, -1f, 1f, 1f, -1f, -1f, 1f, 1f, -1f,      // v0-v3-v4-v5 right
   1f, 1f, 1f, 1f, 1f, -1f, -1f, 1f, -1f, -1f, 1f, 1f,      // v0-v5-v6-v1 up
  -1f, 1f, 1f, -1f, 1f, -1f, -1f, -1f, -1f, -1f, -1f, 1f,   // v1-v6-v7-v2 left
  -1f, -1f, -1f, 1f, -1f, -1f, 1f, -1f, 1f, -1f, -1f, 1f,   // v7-v4-v3-v2 down
   1f, -1f, -1f, -1f, -1f, -1f, -1f, 1f, -1f, 1f, 1f, -1f   // v4-v7-v6-v5 back
};

float[] normals = new float[]
{
    0f, 0f, 1f, 0f, 0f, 1f, 0f, 0f, 1f, 0f, 0f, 1f,     // v0-v1-v2-v3 front
    1f, 0f, 0f, 1f, 0f, 0f, 1f, 0f, 0f, 1f, 0f, 0f,     // v0-v3-v4-v5 right
    0f, 1f, 0f, 0f, 1f, 0f, 0f, 1f, 0f, 0f, 1f, 0f,     // v0-v5-v6-v1 up
    -1f, 0f, 0f, -1f, 0f, 0f, -1f, 0f, 0f, -1f, 0f, 0f, // v1-v6-v7-v2 left
    0f, -1f, 0f, 0f, -1f, 0f, 0f, -1f, 0f, 0f, -1f, 0f, // v7-v4-v3-v2 down
    0f, 0f, -1f, 0f, 0f, -1f, 0f, 0f, -1f, 0f, 0f, -1f  // v4-v7-v6-v5 back
};

int[] indices = new int[]
{
    0, 1, 2, 0, 2, 3,       // front
    4, 5, 6, 4, 6, 7,       // right
    8, 9, 10, 8, 10, 11,    // up
    12, 13, 14, 12, 14, 15, // left
    16, 17, 18, 16, 18, 19, // down
    20, 21, 22, 20, 22, 23  // back
};
```

Рис. 27. **Опис кубічного об'єкта**

Після опису об'єкту треба створити буфери для обробки та збереження вершин і нормалей за допомогою методу InitArrayBuffer.

InitArrayBuffer(program, vertices, "aPosition");
InitArrayBuffer(program, normals, "aNormal").

Об'єкти OpenGL зберігаються на графічній карті, тому їхній напрям змінювати безпосередньо не можна. Робота з об'єктами відбувається за їхнім дескриптором – номером, за яким цей об'єкт розташовано на графічній карті.

Створення нового буфера виконують функцією GenBuffer, яка повертає номер новоствореного буфера (VBO – Vertex Buffer Object). Метод BindBuffer пов'язує буфер типу Array із цим номером. Потім метод BufferData копіює дані в буфер. Останній параметр цього методу вказує, як часто можуть змінюватися дані буфера.

Наступні рядки вказують вершинному шейдеру, як саме інтерпретувати дані в буфері, а також асоціюють із об'єктом у буфері символьну назву (рис. 28). Функція GetAttribLocation знаходить номер вказаного символьного атрибута в поточній програмі (він потрапив туди після попередньої компіляції шейдера). Наступна функція VertexAttribPointer пов'язує атрибут із створеним буфером, указує, що один його об'єкт містить три значення типу float, дані ненормалізовані. Наступний нуль (stride) указує, скільки байт міститься між останнім значенням об'єкта і першим значенням наступного об'єкта. У нашому випадку трійки координат розташовані одна за одною, тож там нема порожнього місця між ними. Останній нуль (offset) указує, скільки байтів необхідно пропустити на початку масиву, щоб знайти перший елемент. У нашому випадку він починається з першого байта.

Функція EnableVertexAttribArray вмикає створений атрибут.

```
2 references
private void InitArrayBuffer(int program, float[] data, string attributeName)
{
    int vbo = GL.GenBuffer();

    GL.BindBuffer(BufferTarget.ArrayBuffer, vbo);
    GL.BufferData(BufferTarget.ArrayBuffer, sizeof(float) * data.Length, data, BufferUsageHint.StaticDraw);

    int attributeLocation = GL.GetAttribLocation(program, attributeName);
    GL.VertexAttribPointer(attributeLocation, 3, VertexAttribPointerType.Float, false, 0, 0);
    GL.EnableVertexAttribArray(attributeLocation);
}
```

Рис. 28. **Код ініціалізації буферу**

Після створення буферів для координат вершин і нормалей потрібно також створити буфер елементів (рис. 29), який містить номери вершин, з яких будуть створюватися трикутники на гранях нашого куба.

```
int indexBuffer = GL.GenBuffer();
GL.BindBuffer(BufferTarget.ElementArrayBuffer, indexBuffer);
GL.BufferData(BufferTarget.ElementArrayBuffer, sizeof(int) * indices.Length, indices, BufferUsageHint.StaticDraw);
```

Рис. 29. **Код створення буферу елементів**

Створення буферів завершено, і ми повертаємося в метод OnLoad. У наступних рядках послідовно:

- знаходять посилання на змінну (точніше, на об'єкт OpenGL) із назвою uMvpMatrix (ця змінна була створена в шейдері вершин);
- установлюють матрицю виду та матрицю моделі;
- установлюють фоновий колір сцени (чорний);
- умикають функцію визначення глибини.

```
_uMvpMatrixLocation = GL.GetUniformLocation(program, "uMvpMatrix");
_viewMatrix = Matrix4.LookAt(eye: new Vector3(0f, 0f, 10f), target: new Vector3(0f, 0f, 0f), up: new Vector3(0f, 1f, 0f));
_modelMatrix = Matrix4.Identity;
GL.ClearColor(0f, 0f, 0f, 1f);
GL.Enable(EnableCap.DepthTest).
```

Залишилося три методи в класі Window.

Метод OnResize (рис. 30) виконується під час запуску форми та кожної зміни її розміру і установлює розмір поля зору та матрицю проєктування для огляду сцени.

```
protected override void OnResize(EventArgs e)
{
    base.OnResize(e);
    GL.Viewport(0, 0, Width, Height);
   _projMatrix = Matrix4.CreatePerspectiveFieldOfView(0.78f, (float)Width / Height, 1f, 100f);
}
```

Рис. 30. **Код, який виконується при зміні розміру форми**

Метод OnRenderFrame (рис. 31) виконується для малювання кожного фрейма (зазвичай настільки часто, наскільки це можливо технічно). GL.Clear очищує буфери кольору та глибини, Gl.UniformMatrix4 оновлює значення відповідних матриць. GL.DrawElements є головною функцією власне для малювання об'єктів (у нашому випадку трикутників), отримуючи на вхід також кількість вершин, їхній тип та зміщення (offset).

OpenTK реалізує подвійну буферизацію. Він використовує два буфери під час роботи, один із яких відображається, а інший у цей самий час – створюється (рендериться). Метод SwapBuffers відображає готовий для демонстрації буфер.

```
protected override void OnRenderFrame(FrameEventArgs e)
{
    base.OnRenderFrame(e);

    GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

    GL.UniformMatrix4(_uNormalMatrixLocation, false, ref _modelMatrix);
    GL.UniformMatrix4(_uMvpMatrixLocation, false, ref _mvpMatrix);

    GL.DrawElements(PrimitiveType.Triangles, _amountOfVertices, DrawElementsType.UnsignedInt, 0);

    SwapBuffers();
}
```

Рис. 31. **Код, який виконується при малюванні фрейму**

Останній метод OnUpdateFrame (рис. 32) викликається з указаною на початку створення вікна частотою. Він оновлює матрицю трансформацій об'єкта.

```
0 references
protected override void OnUpdateFrame(FrameEventArgs e)
{
    base.OnUpdateFrame(e);

    _modelMatrix = Matrix4.CreateScale(1f, 1f, 1f) * Matrix4.CreateRotationY(angleRad);
    _mvpMatrix = _modelMatrix * _viewMatrix * _projMatrix;
}
```

Рис. 32. **Код для оновлення фрейму**

Практична частина
Індивідуальні завдання

1. Змініть точку, у якій знаходиться камера. Нехай це буде точка (3; 7; 10).

Очікуваний результат наведено на рис. 33.

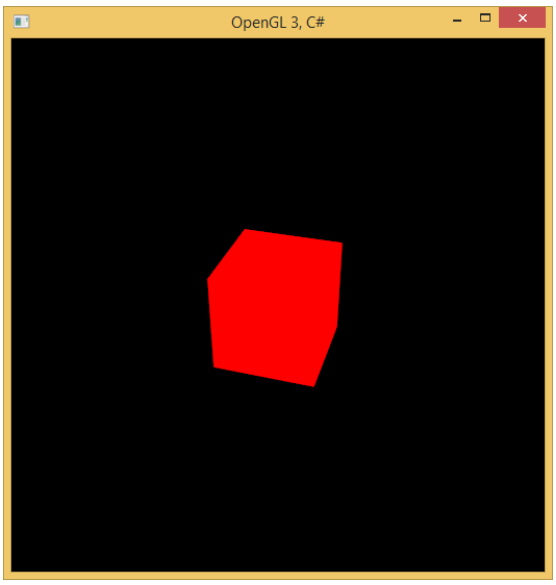


Рис. 33. **Результат після зміни положення камери**

2. Зробіть так, щоб куб обертався навколо осі Y на кут 0.01 радіана порівняно з попереднім фреймом.

3. Додайте код, який буде збільшувати куб на кожному фреймі в 1.01 раза, а як тільки куб стане вдвічі більшим – так само плавно зменшувати його до початкового розміру.

4. Додайте роботу з кольором.

4.1. Модифікуйте шейдер вершин. Він повинен отримати на вхід вектор нормалі (aNormal) і матрицю uNormalMatrix. Для вектора aNormal у початковому прикладі коду вже створюється відповідний буфер. Матриця uNormalMatrix також формується, але не записується в об'єкт uniform. Це необхідно додати (аналогічно до матриці uMvpMatrix).

Метод main рейдера вершин повинен додатково виконувати обчислення кута між вектором нормалі та вектором від джерела світла й заповнювати змінну на виході vNdotL:

```
vec4 normal = uNormalMatrix * aNormal;
vec3 lightDir = vec3(1, 5, 3);
vNdotL = max(dot(normalize(normal.xyz), normalize(lightDir)), 0.0);
```

4.2. Модифікуйте шейдер фрагментів таким чином:

```
var fShaderSource =
    @"
```

```
#version 130
precision mediump float;

in float vNdotL;
uniform vec4 uColor;
out vec4 fragColor;

void main()
{
   vec3 diffuseLight = vec3(1.0, 1.0, 1.0);
   vec3 diffuseColor = diffuseLight * uColor.rgb * vNdotL;

   vec3 ambientLight = vec3(0.2, 0.2, 0.2);
   vec3 ambientColor = ambientLight * uColor.rgb;

   fragColor = vec4(diffuseColor + ambientColor, uColor.a);
} ";
```

Тепер він обчислює значення кольору fragColor із використанням кута до джерела поширення світла та параметрів дифузного й постійного освітлення.

4.3. Шейдер фрагментів отримує uniform-змінну uColor, яка встановлює базовий колір. Цю змінну треба встановити наступним фрагментом коду:

```
_uColor = GL.GetUniformLocation(program, "uColor");
GL.Uniform4(_uColor, 1f, 0f, 0f, 1f);
```

Підготуйте звіт за результатами виконаної роботи. Звіт має містити титульний аркуш; опис виконаних завдань із коментарями, кодом, скриншотами; висновки з кожного завдання та з лабораторної роботи загалом. Висновки не повинні копіювати мету лабораторної роботи.

Підготуйтеся до захисту роботи за контрольними запитаннями. Запитання, позначені символом *, передбачають самостійне вивчення додаткової літератури.

Контрольні запитання до лабораторної роботи 5

1. Що таке OpenGL?
2. Чим OpenGL відрізняється від DirectX?
3. Навіщо потрібні шейдери?
4*. Для чого використовувати буфери для OpenGL?
5*. Яка різниця між перспективним і ортографічним проєктуванням?
6. Що таке триангуляція та теселяція?
7*. Для чого потрібен z-buffer?
8*. Які моделі освітлення вам відомі? Чим вони відрізняються?
9*. Що таке трасування променів?
10*. Що таке кидання променів?

Рекомендована література

Основна

1. Гаврилов В. П. 3D-графіка [Електронний ресурс] : навч. посіб. / В. П. Гаврилов ; Харків. нац. екон. ун-т ім. С. Кузнеця. – Електрон. текстові дан. (18,5 МБ). – Харків : ХНЕУ ім. С. Кузнеця, 2018. – 126 с.

2. Кобилін О. А. Методи цифрової обробки зображень : навч. посіб. / О. А. Кобилін, І. С. Творошенко. – Харків : ХНУРЕ, 2021. – 124 с.

3. Комп'ютерна графіка: конспект лекцій для студентів усіх форм навчання спеціальностей 122 "Комп'ютерні науки" та 123 "Комп'ютерна інженерія" з курсу "Комп'ютерна графіка" / О. П. Скиба. – Тернопіль : Тернопіль. нац. техн. ун-т ім. І. Пулюя, 2019. – 88 с.

Додаткова

4. Веселовська Г. В. Комп'ютерна графіка : навч. посіб. для студентів ВНЗ / Г. В. Веселовська, В. Є. Ходаков, В. М. Веселовський ; під ред. В. Є. Ходакова. – Херсон : Олді-Плюс, 2017. – 581 с.

5. Гороховатський О. В. Метод пошуку подібних об'єктів на зображенні в умовах невизначеності / О. В. Гороховатський, О. О. Передрій // Системи обробки інформації. – 2018. – № 2(153). – С. 152–158.

6. Fundamentals of Computer Graphics. Fourth Edition / S. Marschner, P. Shirley. – Boca Raton, Fla. : CRC Press, 2016. – 748 p.

7. Mandelbrot B. The fractal geometry of nature / B. Mandelbrot. – New York : Henry Holt and Company, 1983. – 468 p.

Інформаційні ресурси

8. Гороховатський О. Комп'ютерна графіка та обробка зображень [Електронний ресурс] / О. Гороховатський. – Режим доступу : https://pns.hneu.edu.ua/course/view.php?id=8007.

9. Комп'ютерна графіка [Електронний ресурс] : навч. посіб. : в 2 кн. Кн. 1. / уклад. О. В. Тотосько, А. Г. Микитишин, П. Д. Стухляк. – Тернопіль : Тернопіль. нац. техн. ун-т ім. І. Пулюя, 2017. – 304 с. – Режим доступу : http://elartu.tntu.edu.ua/bitstream/lib/22337/1/Komp_graf_knyga_1.pdf.

10. Eck D. Introduction to computer graphics [Electronic resource] / D. Eck. – Access mode : http://math.hws.edu/graphicsbook/.

11. Vries J. Welcome to OpenGL [Electronic resource] / J. Vries. – Access mode : https://learnopengl.com/.

Зміст

Вступ 3
Лабораторна робота 1. Формати графічних файлів 5
Лабораторна робота 2. Фрактальна графіка 13
Лабораторна робота 3. Геометричні перетворення зображень 20
Лабораторна робота 4. Фільтрація та поліпшення зображень 31
Лабораторна робота 5. Вступ до OpenGL 34
Рекомендована література 43
    Основна 43
    Додаткова 43
    Інформаційні ресурси 44

НАВЧАЛЬНЕ ВИДАННЯ

КОМП'ЮТЕРНА ГРАФІКА ТА ОБРОБКА ЗОБРАЖЕНЬ

Методичні рекомендації до лабораторних робіт

для здобувачів вищої освіти спеціальності 126 "Інформаційні системи та технології" освітньої програми "Інформаційні системи та технології" першого (бакалаврського) рівня

Самостійне електронне текстове мережеве видання

Укладачі: **Удовенко** Сергій Григорович
Гороховатський Олексій Володимирович
Передрій Олена Олегівна

Відповідальний за видання *С. Г. Удовенко*

Редактор *Н. Г. Войчук*

Коректор *В. О. Дмитрієва*

План 2024 р. Поз. № 101 ЕВ. Обсяг 46 с.

Видавець і виготовлювач – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Науки, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру ***ДК № 4853 від 20.02.2015 р.***