

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМИ ТЕСТУВАННЯ ВІДЕОАДАПТЕРА.
DEVELOPMENT OF A VIDEO ADAPTER TESTING PROGRAM.

Виконав: студент(ка) 4 курсу, групи КНТ-217
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Майборода С.І.

(прізвище та ініціали)

Керівник Сердюк С.М.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Майбороди Сергія Ігоревича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програми тестування відеоадаптера.
Development of a Video Adapter Testing Program.

керівник проєкту (роботи) доц., к.т.н. Сердюк Сергій Микитович,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року
№ 103

2. Строк подання студентом проєкту (роботи) травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Характеристика об'єкта проєктування. 2. Матеріали та методи. 3. Основні рішення щодо реалізації компонентів системи. 4. Керівництво програміста. 5. Керівництво оператора. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Сердюк С.М., доцент		
Нормоконтроль	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання 28 лютого 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2–3 тижні	Розділ 1
3	Розробка архітектури програми.	4 тиждень	Розділ 2
4	Розробка програми.	5–6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми.	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8–9 тижні	Розділ 5, Додатки
7	Захист роботи.	10 тиждень	

Студент(ка)

(підпис) Майборода С.І.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Сердюк С.М.
(прізвище та ініціали)

(підпис)

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
86 с., 2 табл., 12 рис., 2 дод., 15 джерел.

C#, WINDOWS, NETFRAMEWORK, VIDEOCARD, GPU, CORE,
INTERFACE, СТРЕС-ТЕСТУВАННЯ.

Об'єкт дослідження – процес тестування відеоадаптеру.

Предмет дослідження – методи тестування відеоадаптеру.

Мета роботи – розробка програмного продукту для тестування характеристик відеоадаптеру за допомогою стрес-тесту.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування; мова програмування C# та середовище розробки Microsoft Visual Studio 2017 з інтегрованим фреймворком .NET Framework 4; персональний комп'ютер (ПК) з процесором не гірше Intel Celeron E1200 1.6 ГГц; відеокарта не гірше Nvidia GeForce 8100 GT 256 Мб під управлінням операційної системи (ОС) Windows 7 або вище.

Результати. Розроблено застосунок, який виконує стрес-тестування відеоадаптеру. Тест полягає у моделюванні системи динамічного руху частинок, що мають свій колір та текстуру.

Висновки. Розроблено утиліту для використання досвідченими користувачами ПК, яка забезпечує підвищення ефективності використання відео-підсистеми ПК.

Галузь використання – експлуатація та оптимізація комп'ютерних систем.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 86 pages, 12 figures, 2 tables, 2 appendixes, 15 sources.

C#, WINDOWS, NETFRAMEWORK, VIDEOCARD, GPU, CORE, INTERFACE, STRESS TESTING.

Object of study is a video adapter testing process.

Subject of study are methods of video adapter testing.

The purpose of the work is development of software product for testing the characteristics of a video adapter using a stress test.

development of a software product for testing the characteristics of a video adapter using a stress test.

Materials, methods and technical means: object-oriented programming; Windows executive part; C# programming language and development tools Microsoft Visual Studio 2017 with integrated framework .NET Framework 4; a personal computer (PC) with a processor no worse than the Intel Celeron E1200 1.6 GHz; video card is not worse than Nvidia GeForce 8100 GT 256 MB with support for DirectX 10 under the control of the Windows 7 or higher operating system (OS).

Results. An application has been developed that performs stress testing of the video adapter. The test consists in simulating a system of dynamic movement of particles with their own color and texture.

Conclusions. A software product for use by experienced PC users, which improves the efficiency of using the PC video subsystem.

The field of use is the operation and optimization of computer systems.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Характеристика об'єкта проєктування	11
1.1 Огляд типів тестів відеоадаптеру	11
1.1.1 Ігри для тестування GPU	11
1.1.2 Синтетичні тести GPU	11
1.1.3 Обчислювальні тести GPU	11
1.2 Аналіз існуючих аналогів	22
1.3 Розробка технічного завдання.....	25
1.3.1 Підстава для розробки	25
1.3.2 Мета розробки та її призначення.....	25
1.3.3 Основні вимоги до програмного продукту.....	25
1.3.4 Вимоги до видів забезпечення і компонентів	28
1.3.5 Обмеження на установку і виконання програми	29
1.3.6 Вимоги до програмної документації	29
1.3.7 Умови експлуатації	29
1.3.8 Вимоги до маркування та упаковки	29
2 Матеріали та методи.....	30
2.1 Огляд обраної мови програмування, операційної системи та середовища розробки.....	30
2.2 Огляд обраних для розробки застосунку технологій	33
3 Основні рішення щодо реалізації компонентів системи	37
3.1 Розробка класів застосунку	37
3.2 Розробка програмних модулів	39
3.3 Розробка інтерфейсу користувача	42
4 Керівництво програміста	47
4.1 Призначення і умови застосування програми	47
4.2 Характеристики програми	47

4.2.1 Юзабіліті характеристики програми	47
4.2.2 Загальні характеристики програми та її структура.....	48
4.3 Початкові та вихідні дані.....	49
4.4 Повідомлення програмісту	49
5 Керівництво оператора.....	50
5.1 Призначення програми	50
5.2 Умови використання програми.....	50
5.3 Виконання програми	50
5.4 Повідомлення оператору	53
Висновки.....	57
Перелік джерел посилання	58
Додаток А Текст програми	71
Додаток Б Копії слайдів презентації.....	82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CLR	– Common Language Runtime;
FPS	– Frames Per Second;
GPU	– Graphics Processing Unit;
БД	– база даних;
ГГц	– гігагерц;
Гб	– гігабайт;
МГц	– мегагерц;
ОС	– операційна система;
ПК	– персональний комп'ютер.

ВСТУП

Починаючи з відеоігор і до професійної роботи з 3D графікою, графічна карта є незамінним елементом комп'ютерної системи для тих, хто хоче вивести свої комп'ютери на новий рівень графічного рендеринга. Графічна карта (інша назва «відеокарта»), являє собою друковану плату в комп'ютері зі спеціалізованим обладнанням, оптимізованим для відображення високоякісної графіки з високою швидкістю.

Зазвичай вона містить виділену пам'ять з довільним доступом або оперативний запам'ятовуючий пристрій, мікросхеми, які використовуються для зберігання даних, що відносяться до візуальних носіїв, і спеціалізований чіп процесора, який називається графічним процесором (GPU), що оптимізований для роботи з відео [1].

Однак, як це зазвичай буває з додаванням додаткового обладнання до комп'ютера, відеокарти привносять свої власні проблеми в систему. Графічні карти, які вийшли з ладу, можуть викликати збій персонального комп'ютера (ПК). Ці збої можуть бути різними: від простого синього екрану до «зависання» (коли комп'ютер зависає, але не відображається синій екран), до випадкових перезапущів і виключення живлення. Не кожен збій буде викликаний відеокартою, але аналіз дамів пам'яті може обґрунтувати впевненість в тому, що даний збій стався з вини драйвера відеокарти.

Таким чином, проведення тестування відеокарт на етапах складання ПК, експлуатації або модернізації є актуальним завданням. Тестування – це не тільки відмінний спосіб дізнатися, наскільки стало краще завдяки апгрейду обладнання або програмних засобів. Результати тестування дозволяють оцінити продуктивність нового обладнання та оптимізувати склад комп'ютерної системи. Проведення стрес-тестування для перевірки живлення, температури і загального охолодження ПК також важливо, щоб користувачі не зіткнулися з проблемами в майбутньому. Ці тести особливо корисні для тих, хто намагається знайти стабільний розгін (тому іноді стрес-тестування називають

тестом стабільності відеокарти) або для вирішення різних проблем, з якими може зіткнутися користувач при оновленні компонентів ПК.

Метою даної дипломної кваліфікаційної роботи бакалавра є розробка програмного забезпечення для тестування характеристик відеоадаптеру за допомогою стрес-тесту, тобто при максимальних навантаженнях графічного процесору.

1 ХАРАКТЕРИСТИКА ОБ'ЄКТА ПРОЄКТУВАННЯ

1.1 Огляд типів тестів відеоадаптеру

Існує три основні типи тестів графічного процесора: реальні ігри, «синтетичні» тести відеокарт і обчислювальні тести. Як правило, кращий спосіб перевірити продуктивність відеокарти – запустити гру, в яку користувач вже грає або хоче пограти. Синтетичні тести, як правило, дуже легко запустити, але результати тестування тільки свідчать про те, як комп'ютер обробляє цей конкретний тест. Таким чином, результати тестування можна прирівнювати до реальної продуктивності комп'ютера з певною часткою похибки. Нарешті, обчислювальні тести GPU зазвичай сильно відрізняються за принципом роботи від ігор [2].

Навіть в тому випадку, коли користувач регулярно запускає на своєму комп'ютері серйозні наукові програми типу Folding@Home (проєкт розподілених обчислень для проведення комп'ютерного моделювання згортання молекул білка) або використовує застосунок з прискоренням на графічному процесорі, такий як LuxCoreRenderer (моделює потік світла відповідно до фізичних рівнянь, створюючи реалістичні зображення фотографічної якості), його комп'ютерна система може потребувати оптимізації для певних архітектур графічного процесора, щоб комп'ютер міг виконати будь-яке конкретне робоче навантаження.

На практиці, перш ніж починати використовувати будь-якої з тестів графічного процесора, необхідно встановити на комп'ютер, що тестується кілька корисних утиліт. По-перше, для множини реальних ігрових тестів знадобиться інструмент для фіксації часу кадру. Для цього часто використовують три основні програми.

PresentMon – це інтерфейс командного рядка для реєстрації часу кадрів. Це найменш зручний варіант, який останнім часом фактично був витіснений такими застосунками як OCAT (Open Capture and Analytics Tool) і FrameView. Обидві програми базуються на основних функціях PresentMon. Ще одна відома

утиліта FRAPS, призначена для підрахунку кількості FPS (кадрів в секунду) не оновлювалася з 2013 року і має деякі помилки в Windows 8/10 [2]-[3], тому краще використовувати вищезгадані сучасні утиліти.

OCAT був створений інженерами AMD і має повністю відкритий вихідний код, а FrameView – це розробка корпорації Nvidia. Між ними є незначні відмінності в інтерфейсах і функціональності, найбільша з яких полягає в тому, що FrameView реєструє дані про енергоспоживання.

Сайтом Tom's Hardware було протестовано енергоспоживання відеокарт з використанням вбудованого обладнання для вимірювання точних навантажень. Результат порівняння з даними, отриманими за допомогою FrameView показав, що показники потужності Nvidia для графічних процесорів Nvidia визначені з точністю до декількох ват. Однак графічні процесори AMD повідомляють про потужність, що споживається тільки для графічного процесора, що може означати різницю від 10 Вт до 100 Вт, в залежності від конкретного графічного процесора. В іншому, всі ці три інструменти видають один і той же загальний формат файлу, який надає інформацію про час кадрів, тактову частоту та безліч інших деталей.

Важливо відзначити, що багатьом іграм не «подобається» вбудована в ці програми функція накладення. Застосунки платформи Microsoft Store UWP і всі інші, що мають об'ємну систему захисту авторських прав (DRM - Digital Rights Management), часто не запускаються, якщо включений оверлей, тому рекомендується робота без використання оверлеїв. Тобто мова йде про те, що якщо тестувальнику необхідно відображення лічильника частоти кадрів в кутку його екрану, то гра може перестати запускатися. Крім того ігри, що використовують Vulkan API, за досвідом [2] несумісні з накладенням частоти кадрів, що може привести до того, що оверлей також не буде відображатися на екрані.

Ще один інструмент, який може знадобитися для тестування відеоадаптера – це застосунок для розгону або налаштування графічного процесора, щоб допомогти поліпшити результати тестів. Багато постачальників

відеокарт мають власне програмне забезпечення, яке працює тільки з певною маркою карти, але не всі ці пакети програмного забезпечення пропонують однаковий рівень налаштування. Зазвичай замість утиліт Asus, Gigabyte, ASRock, Sapphire і т. д. використовується відома безкоштовна утиліта розгону відеокарт MSI Afterburner, яка працює практично з будь-яким графічним процесором, випущеним за останнє десятиліття або більше. Крім того, можливість налаштування кривої швидкості обертання вентилятора/охолодження є відмінним способом знизити температуру графічного процесора і цим потенційно підвищити його продуктивність, хоча при цьому може значно збільшитися шум вентилятора.

1.1.1 Ігри для тестування GPU

У разі використання застосунку ОСАТ для тестування графічного процесора можна використовувати будь-яку гру. Головний ключовий момент – знайти розділ, що повторюється і протестувати його кілька разів. Крім того, в деяких іграх присутня опція блокування частоти кадрів або обмеження частоти кадрів. Тому, тестування ігор в жанрі «gun and gun» об'єднаних з платформером (наприклад, Cuphead) не є особливо ефективним, крім тих випадків, коли просто необхідно підтвердити, що можна досягти стабільних 60 кадрів в секунду.

Однак, в разі якщо необхідні прості тести графічного процесора, найкраще вибрати одну з відносно великої кількості ігор, в яких вже є вбудовані інструменти тестування. Тоді як і раніше можна використовувати програму ОСАТ для реєстрації часу кадру (рекомендується [2] як універсальний спосіб збору метрик продуктивності), а також для контролю точки початку/зупинки для ведення журналу часу кадру. Але якщо необхідно тільки оцінити, наскільки змінилася продуктивність після реконфігурації системи, вбудованих тестів цілком достатньо.

Недоліком ігрового методу тестування є те, що для його використання

необхідно бути власником гри, тобто купити її. Розглянемо кілька ігор, випущених після 2015 року, які мають вбудовані тести графічного процесора.

Гра Ashes of the Singularity (вимагає 22,0 Гб для установки) широко використовується в оглядах процесорів і графічних процесорів, бенчмаркінге AotS і його розширенні Escalation. Вона, в першу чергу, відмінно підходить для завантаження центрального процесора, так як виводить на екран численні юніти і об'єкти (рис.1.1). Саме в цьому випадку низькорівневі API функції часто працюють краще. Гра підтримує DirectX 11 (DX11), DirectX 12 (DX12) і рендеринг Vulkan, причому останні два краще DX11, оскільки вони дозволяють процесору обробляти об'єкти сцени більш ефективно.



Рисунок 1.1 – Скриншот з гри Ashes of the Singularity

Крім того, Ashes of the Singularity єдина гра, яка здійснює явну підтримку декількох графічних процесорів в DX12, що дозволяє спробувати змішувати і зіставляти графічні процесори різних рівнів – наприклад, RTX 2070 з RX 5700 XT.

Результати тестів даної гри для роздільної здатності 1440p ultra на процесорі Core i9-9900K з відеокартою GeForce RTX 2080 Ti і API DirectX 12

наступні: в середньому 108,8 кадру в секунду (FPS); мінімальний поріг «осідання» FPS 71,7 99-й перцентиль (перцентиль - це показник від 1 до 100, що демонструє який саме відсоток даних розташований нижче заданого рівня вибірки).

Гра Assassin's Creed Odyssey (вимагає 98,5 Гб для установки) має вбудований тест графічного процесора, який займає всього близько хвилини. У цьому тесті кількість і різноманітність хмар в небі може мати помірний (до 10%) вплив на продуктивність процесора. Рекомендується [2] виконати тест як мінімум 3-5 разів, намагаючись отримати «ясний/сонячний» день для порівняння, або виконати його п'ять разів і взяти середнє значення.

До недоліків тестування даним способом слід віднести необхідність перезавантаження між більшістю змін у налаштуваннях (не мається на увазі налаштування роздільної здатності). Внаслідок цього, тестувальник зазвичай витрачає не більше хвилини на підготовку до тесту, а потім 60 секунд власне на тест. Assassin's Creed досить популярна серія, тому останню частину Valhalla можна розглядати в якості тесту графічного процесора.

Результати тестів даної гри для роздільної здатності 1440p ultra на процесорі Core i9-9900K з відеокартою GeForce RTX 2080 Ti і API DirectX 11 наступні: в середньому 74,5 кадру в секунду; мінімальний поріг «осідання» FPS 58,7 99-й перцентиль.

Методика використання ігор без вбудованих функцій тестування графічного процесора полягає в наступному. У цьому випадку необхідно використовувати одну і ту ж послідовність випробувань, наскільки це можливо, тобто тестування, наприклад, в середині перестрілки з декількома ворогами не слід проводити, оскільки битва рідко буде кожен раз проходити однаково. Рекомендується використовувати як мінімум 60-секундну тестову послідовність, а також програму ОСАТ з наступних причин [2]:

- вона довела свою надійність у всіх іграх;
- вона використовує функцію звукової підказки. Так як накладення часто не працює, наявність чутного звуку моменту запуску/зупинки дозволяє

дізнатися, що тестувальник правильно запустив захоплення кадру.

Дану програму можна також налаштувати для відстеження продуктивності для процесів, що створюють додаткові файли (наприклад, для Explorer.exe), які можна видалити з каталогу результатів.

Важливо відзначити, що, оскільки ручне тестування не має заздалегідь встановленого сценарію, то не можна порівнювати результати різних тестів якщо всі вони не використовують одну і ту ж послідовність дій. Наприклад, якщо протестувати таку гру, як Battlefield V, на різних картах і навіть в різних областях однієї і тієї ж карти, продуктивність буде трохи відрізнятися. Ось чому більшість тестів також дотримуються розрахованого на одного користувача тестування, оскільки багатокористувацький режим вводить множину інших змінних.

1.1.2 Синтетичні тести GPU

Синтетичні тести дозволяють емулювати навантаження на окрему підсистему комп'ютера. Тестування в реальних іграх зазвичай краще, ніж тестування за допомогою синтетичних тестів GPU, так як вони не завжди відображають реальну продуктивність відеокарти або системи [2], [4]. Однак у випадку з розгоном, вони можуть краще продемонструвати приріст продуктивності, навіть самий незначний, за що і користуються популярністю у оверклокерів. Крім того синтетичні тести теж дають дуже точне уявлення про можливості технології, а отже, і про потенціал виробу [5].

Для того щоб оцінити наскільки добре працює «Гра/Завдання X», безпосереднє тестування з цією грою/завданням – єдиний спосіб отримати найбільш точну відповідь. Безглуздо використовувати одну гру, щоб передбачити продуктивність іншої. Використання синтетичних тестів графічного процесора для оцінки ігрової продуктивності також не обов'язково дасть точну відповідь. Однак у таких тестів є свої переваги.

По-перше, вони часто безкоштовні. Деякі з цих інструментів є

кросплатформними, завдяки чому можна порівнювати ПК з операційною системою (ОС) Windows з пристроями iOS, MacOS і Android. Це також є недоліком, оскільки швидкі відеокарти для ПК часто на порядок потужніші ніж мобільних пристроїв, а це означає, що тести часто дуже обмежені, щоб не перевантажувати смартфон. Нарешті, синтетичні тести мають набагато менший розмір – найбільший з тестових наборів, 3DMark, становить всього 6 Гб, що приблизно дорівнює розміру невеликої гри.

Розглянемо кращі тести графічного процесора.

3DMark – є одним з найстаріших, найпопулярніших і кращих пакетів тестів графічних процесорів, розроблених фінською компанією Futuremark. Він дає досить точні результати, якщо необхідно дізнатися загальну різницю в продуктивності між графічними процесорами. Оцінка видається як частота кадрів на секунду в комплексних тестах та в умовних одиницях у штучних тестах.

Для оцінка продуктивності ПК з використанням DirectX 11 та 12 рекомендовано [5] використовувати тести Fire Strike та Time Spy відповідно, які увійшли у склад пакету 3DMark у 2013 році. Багатоплатформовий тест Night Raid менш корисний, так як його потрібно масштабувати до мобільних пристроїв. Різниця в продуктивності між різними графічними процесорами може коливатися на 20% в будь-якому напрямку, в залежності від тесту, що більше, ніж розрив між рівнями графічних процесорів. Як і в будь-якому окремому тесті, це всього лише одна точка даних, але 3DMark, можливо, є найкращим відображенням продуктивності графічного процесора за межами ігрового тестування. Однак це ні в якому разі не універсальна метрика.

Графічний процесор Basemark в порівнянні з 3DMark, є ще більш швидким тестом, який здається менш репрезентативним для визначення реальної продуктивності. Цей бенчмарк доступний для установки безкоштовно. Для його запуску потрібно близько 30 секунд. Тестувальник може вибрати одну з трьох опцій: Vulkan (зазвичай дає кращі результати); DirectX 12 або API рендеринга OpenGL. Цей тест є кросплатформним, що означає, що він повинен

працювати на широкому спектрі обладнання при відносно непоганий точності тестування [2].

Утилита використовує движок Rocksolid від Basemark, який написаний на мові C++ і дозволяє запускати ідентичне навантаження на різних пристроях і технологіях, забезпечуючи, таким чином, просунуте профілювання графіки. Процесор Basemark використовує текстури високої роздільної здатності 4096×4096 , просунуті ефекти, близько 10 тисяч викликів відтворення і 2 млн полігонів на кадр, виводить зображення з роздільною здатністю 3840×2160 і вимагає 4 Гбайт відеопам'яті [6]. Дані характеристики відповідають стандарту сучасних ігор високого класу.

FurMark - один з кращих стрес-тестів графічного процесора для вирішення завдання максимального використання потужності відеокарти користувача. Ця утилита є комплексним набором інструментів для проведення тестування різного рівня навантаження під відеокарти, що підтримують спеціальний 3D інтерфейс API OpenGL. Спеціальна сцена, на якій зображений так званий «волохатий бублик», дозволяє гранично навантажити відеокарту, виміряти її продуктивність і перевірити стабільність роботи під навантаженням [7].

У більшості сучасних графічних процесорів є засоби захисту, що запобігають їх збою в таких інструментах, як FurMark. Проте, існують деякі графічні процесори, технічні можливості яких виходять далеко за рамки своїх офіційних обмежень. Рекомендується [2] використовувати FurMark в якості одного з інструментів при тестуванні енергоспоживання відеокарт.

Коротко охарактеризуємо ще один тест – Unigine Superposition Benchmark. Фірма Unigine створює тести графічного процесора більш десяти років, починаючи з Heaven в 2009 році, Valley в 2013 році і Superposition з 2017 року. Даний тест використовує передові методи рендеринга, такі як SSRTGI (глобальне освітлення з трасуванням променів в просторі екрану) для генерації об'єктів і оточення з фізично точними тінями, відображеннями, заломленнями і освітленням без необхідності в RTX. Є можливість покрокового виконання 14

«налагоджувальних» режимів візуалізації, при цьому тестувальник може бачити, на чому фокусується кожен прохід рендеринга. В даний час цей двигун використовували тільки кілька ігор. Гра Dual Universe, випуск якої запланований на 2021 рік можливо, стане першою великою грою, в якій це буде зроблено.

1.1.3 Обчислювальні тести GPU

Розглянемо останню групу тестів – обчислювальні тести GPU. Робочі навантаження викликані даними тестами можуть сильно відрізнятися від ігор, оскільки вони, як правило, не фокусуються на структурах і геометрії, а замість цього виконують різні наукові обчислення.

Необхідно відзначити, що деякі з розглянутих раніше тестів, такі як FurMark, можуть вивести відеокарти за межі їх звичайних «безпечних» характеристик. В даний час спостерігається бум у використанні відеокарт для майнінгу криптовалют. Є причина, по якій графічні процесори центрів обробки даних часто мають більш низьку тактову частоту (і більші вентилятори), ніж споживчі відеокарти. Справа в тому, що вони призначені для безперебійної роботи та виконання обчислювальних навантажень в режимі 24/7. За даними [2] споживчі графічні процесори, що забезпечують виробництво криптовалют часто розвалюються протягом року, якщо їх запускають кожен день зі 100% навантаженням протягом всього дня.

Це не означає, що всі обчислювальні навантаження діють руйнівно. Багато застосунків, такі як пакет Adobe Creative Cloud, в даний час мають прискорення графічного процесора. Деякі застосунки для 3D-візуалізації також використовують обчислювальні шейдери для підвищення продуктивності. Крім того, можна обмежити вимоги до деяких з цих робочих навантажень, наприклад, для наукової програми Folding@Home задати обмеження (у відсотках) на завантаження графічного процесора.

Розглянемо обчислювальні тести, що використовуються найбільш часто.

AIXPRT - це інструмент, що дозволяє оцінити ефективність вирішення завдань штучного інтелекту, який вимірює продуктивність машинного навчання системи під час класифікації зображень і виявлення об'єктів за допомогою мереж ResNet-50 і SSD-MobileNet v1 [8].

AIXPRT (табл. 1.1) включає підтримку наборів інструментарію Intel OpenVINO, TensorFlow і NVIDIA TensorRT для виконання наведених вище завдань, а також робочого навантаження у вигляді системи рекомендацій Wide і Deep з інструментарієм Apache MXNet. Обчислювальні тести мають рівні точності FP32 (одинарна точність), FP16 (половинна точність) і цілі числа типу INT8. Тестові системи повинні працювати під управлінням Ubuntu 18.04 LTS або Windows 10, а мінімальні вимоги до центрального процесора і графічного процесора залежать від набору інструментарію.

Таблиця 1.1 – Сумісність та потреби інструменту AIXPRT

SDK/Фреймворк	OpenVINO	TensorRT	TensorFlow	MXNet
Апаратні засоби	Intel CPU, Intel GPU, Intel Neural Compute Stick 2, Intel Vision Accelerator	NVIDIA GPU, NVIDIA Xavier	AMD CPU, AMD GPU, Intel CPU, NVIDIA GPU, NVIDIA Xavier	AMD CPU, Intel CPU, NVIDIA GPU
Операційна система	Ubuntu 18.04, Windows 10	Ubuntu 18.04, Windows 10	Ubuntu 18.04, Windows 10	Ubuntu 18.04,

Відносно простим тестом, який буде запускати фізику, графіку, обробку відео, виявлення обличч і 3D-рендеринг (трасування шляху) є CompuBench. Цей тест підтримує фреймворк OpenCL і програмно-апаратну архітектуру паралельних обчислень CUDA, причому остання працює тільки на графічних процесорах Nvidia, в той час як OpenCL працює на всіх відносно нових

графічних процесорах.

Мережа розподілених обчислень Folding@Home складається з набору обчислювальних пакетів, які допомагають вченим і дослідникам з'ясувати, як згортаються білки, що може привести до кращого розуміння (і, можливо, лікування) безлічі захворювань. Це розподілені обчислення в масовому масштабі. Будь-який користувач може надати обчислювальну продуктивність процесора або відеокарти для даного проєкту. В даний час ця мережа має велику обчислювальну потужність, в порівнянні навіть з найшвидшими суперкомп'ютерами TOP500 - близько 2,6 ексафлопсів (ExaFLOPS) [2], [9].

LuxMark - це багатоплатформовий інструмент 3D-рендеринга для тестування продуктивності OpenCL. У ньому передбачено п'ять тестових сцен різної складності і включені різні режими для їх рендеринга, оскільки OpenCL дозволяє використовувати ряд апаратних комбінацій. Деякі з тестів є інтерактивними. Тестувальник може комбінувати операції з процесором і графічним процесором або розділити їх окремо, щоб побачити переваги кожного сценарію використання [10].

Останні версії LuxMark повертають режим тестування без необхідності використання OpenCL, завдяки написанню движка рендеринга тільки на мові C++ з використанням ядер рендеринга від Intel Embree. Крім режиму продуктивності, підтримується режим максимального навантаження, в якому можна перевірити стабільність роботи комп'ютерного обладнання.

Ще один автономний тест від компанії ChaosGroup – тест V-Ray Next. Це остання версія тесту (2018 рік) з прискоренням на GPU. Він відстежує «шляхи» (маршрути) через CPU або GPU, обчислюючи кількість «шляхів», які дане обладнання може виконати за одну хвилину тесту. Наприклад, за даними [2] відеокарта RTX 2080 Ti управляла 363 mpath (мільйонами шляхів), процесор Core i9-9900K робив тільки 80 mpath, а разом процесор і GPU змогли обчислити 438 mpath. Даний приклад показує чому багато застосунків для відстеження шляху переходять на підтримку графічних процесорів на додаток до традиційного рендерингу за допомогою центрального процесора.

Таким чином, на основі результатів тестування просунутий користувач може оцінити продуктивність свого графічного процесора, переконатися в правильності роботи обладнання або ж обґрунтувати необхідність апгрейду свого обладнання.

1.2 Аналіз існуючих аналогів

Декілька найбільш відомих програми тестування відеоадаптеру, такі як 3DMark, FurMark були вже розглянуті у п. 1.1.2 при огляді синтетичних тестів.

Наступним аналогом є програма AIDA 64. AIDA64 – програмне забезпечення корпорації FinalWire Ltd, яке призначене для тестування і ідентифікації всіх апаратних компонентів персонального комп'ютера під управлінням операційних систем Windows, що надає детальні відомості про апаратне і програмне забезпечення.

Стрес тест програми AIDA64 дозволяє створити повне навантаження для процесора (або навіть декількох процесорів одночасно), підсистеми пам'яті, дисків та відеоадаптеру. Під час тесту можна відстежувати показники температури і напруги, а також швидкості обертання вентилятора і споживання енергії в реальному часі на графіку (рис. 1.2).

Якщо необхідно протестувати стабільність комп'ютера в цілому, можна створити навантаження одночасно для обраних компонентів на кілька годин. Якщо система не покаже будь-яких повідомлень про помилку або не перезавантажиться і не заблокується, значить роботу комп'ютера можна вважати стабільною.

Наступний застосунок MSI Afterburner – багатофункціональна програма для розгону відеокарт NVIDIA і AMD. Він має зручний інтерфейс з розвиненою системою підказок та наступні функціональні можливості [11]:

- регулювання напруги живлення GPU, відеопам'яті і частоти ядра;
- кілька режимів роботи з відеокартами;
- відстеження перемикачів між 2D і 3D режимами;

- створення персональних профілів (одночасно до п'яти);
- можливість управляти швидкістю обертання вентилятора;
- моніторинг частоти кадрів в 3D застосунках (вивід показників в меню і системний трей);
- налаштування температурної межі для графічного процесора Geforce NVIDIA і AMD Radeon.

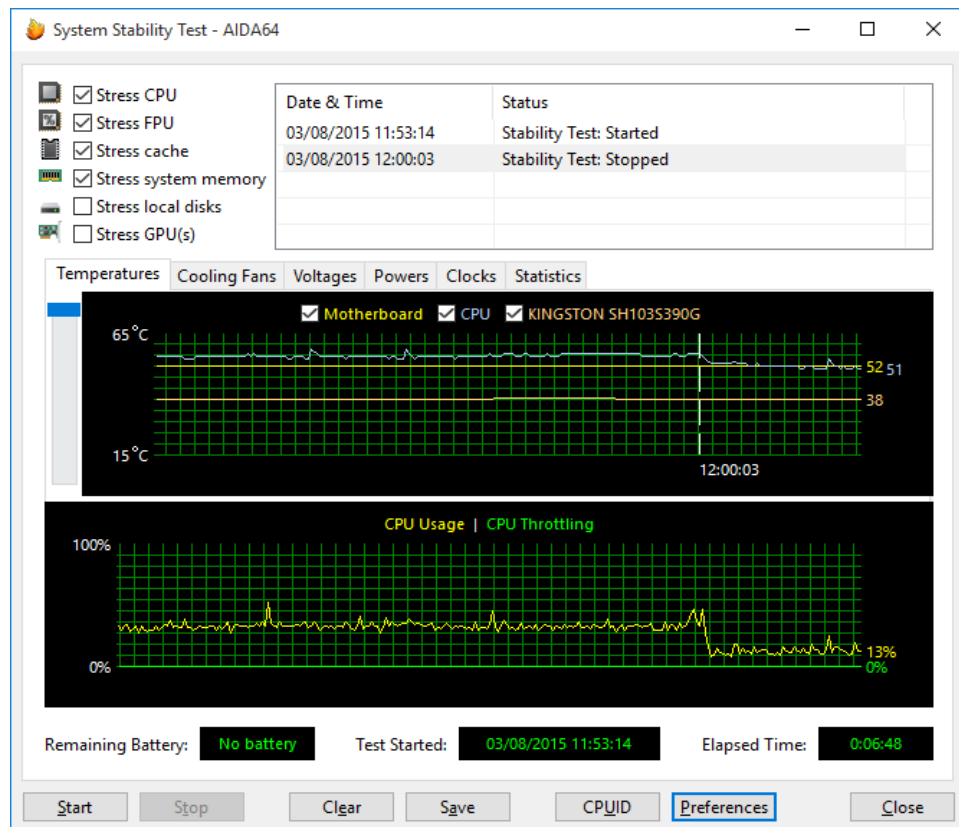


Рисунок 1.2 - Процес стрес тесту програми AIDA64

До недоліків програми MSI Afterburner слід віднести можливі пригальмовування при налаштуванні параметрів відеокарти.

Зручною програмою для оновлення драйверів на відеокарти NVIDIA GeForce є GeForce Experience. Вона користується популярністю серед геймерів, завдяки можливості оптимальної настройки графіки в іграх. Програма має наступні функціональні можливості [11]:

- доступно управління гарячими клавішами;
- передача комп'ютерних ігор на консолі SHIELD по Wi-Fi;

- власний хмарний центр обробки даних для зберігання записів;
- функція створення скріншотів (збереження на робочий стіл або в зазначену папку);
- пошук і оновлення версій драйверів для відеокарти GeForce в автоматичному режимі;
- підтримка технології GameStream для трансляції контенту на YouTube Live в роздільній здатності 4K HDR.

Результати аналізу безкоштовних альтернативних програмних продуктів за критеріями, необхідними для тестування та оцінки відеоадаптера, наведені в табл. 2.2

Таблиця 2.2 – Результати аналізу альтернативних програмних продуктів

Функції	AIDA64	MSI Afterburner	GeForce Experience	FurMark	3DMark
Стрес-тести	+	+	+	+	+
Вимірювання температури GPU	+	+	+	+	—
Перевірка стабільності відеокарти	+	+	+	+	—
Збереження результатів у файл	+	+	+	+	+
Визначення частоти кадрів в 3D-застосунках	+	+	+	+	—
Моніторинг швидкості обертання кулера	+	+	+	+	+
Технічні характеристики відеоадаптера	+	+	+	+	+
Вивід детальної інформації в процесі діагностики	+	+	—	—	—

На основі даних табл. 1.1 можна зробити висновок, що найбільшими функціональними можливостями для проведення тестування GPU мають утиліти AIDA64 та MSI Afterburner.

Порівняно з цими програмами безкоштовна версія утиліти 3D Mark Basic Edition має деякі функціональні обмеження. Однак, в разі придбання ліцензії (Professional Edition), наявність просунутих бенчмарків робить її однією з найбільш популярних і кращих серед аналогів.

Результати аналізу альтернативних програмних продуктів послужили основою для розробки технічного завдання програми тестування відеоадаптеру.

1.3 Розробка технічного завдання

1.3.1 Підстава для розробки

Підставою для розробки служить завдання на дипломну кваліфікаційну роботу бакалавра на тему "Розробка програми тестування відеоадаптеру", затверджене наказом № 103 від 30 березня 2021 р. по НУ «Запорізька політехніка».

1.3.2 Мета розробки та її призначення

Метою розробки є створення програмного продукту для тестування характеристик відеоадаптеру за допомогою стрес-тесту.

1.3.3 Основні вимоги до програмного продукту

На підставі огляду та аналізу проведених у п.п. 1.1 та 1.2, можна сформулювати функціональні вимоги до програми, що розроблюється.

Програма являє собою програмне забезпечення для тестування відеоадаптеру за допомогою стрес-тесту, який використовує відеоадаптер на повну потужність. Тест полягає у моделюванні системи динамічного руху

частинок, що мають свій колір та текстуру. Кожну таку частку повинен відобразити відеоадаптер. Кількість частинок може досягати декількох мільйонів. Стрес-тест має надавати можливість перевірити наступні характеристики відеоадаптеру:

- швидкодія графічного процесору;
- швидкодія відеопам'яті;
- швидкодія шейдерних ядер (якщо вони існують);
- ефективність системи охолодження.

Інтерфейс користувача має надавати наступні можливості:

- вибір повноекранного режиму роботи стрес-тесту;
- вибір роздільної здатності екрану роботи стрес-тесту зі списку чи встановлення своїх значень у пікселях у форматі довжина × ширина;
- запуск стрес-тесту натисканням на кнопку з наступними варіантами роздільної здатності екрану у 1280×720 , 1920×1080 , 2560×1440 чи 4096×2160 пікселей без попереднього ручного вводу значень;
- вибір кількості частинок для виконання стрес-тесту відеоадаптера.
- перегляд назви відеоадаптеру.

Під час виконання стрес-тесту, на екран має виводитися наступна інформація:

- роздільна здатність екрану, яка використовується;
- час тесту;
- поточна кількість кадрів за секунду;
- поточна температура графічного процесору під час тесту;
- поточна частота графічного процесору;
- поточна завантаженість графічного процесору у відсотках;
- поточна частота відеопам'яті;
- поточна завантаженість відеопам'яті у відсотках;
- поточна частота шейдерних ядер;
- поточна кількість обертів вентилятора(ів) на відеоадаптері;
- поточна завантаженість вентилятора(ів) у відсотках.

Необхідно передбачити також можливість зупину стрес-тесту, якщо температура графічного процесору досягла критичної (80 градусів), а також по натисканню на кнопку Esc.

Після проведення стрес-тесту повинна надаватися можливість переглянути наступну інформацію:

- максимальна температура графічного процесору під час тестування;
- розмір екрану під час тестування;
- кількість частинок, що генерувались;
- максимальне та мінімальне значення кількості кадрів за секунду;
- назва відеоадаптеру;
- обсяг відеопам'яті;
- використовувана версія DirectX;
- версія драйверу відеоадаптера;
- назва операційної системи;
- обсяг оперативної пам'яті;
- назва центрального процесору;
- максимальна частота графічного процесору;
- максимальна частота відеопам'яті;
- максимальна частота шейдерних ядер;
- час тестування.

Також необхідно забезпечити можливість зберігання результатів тестування відеоадаптеру у базу даних (БД), та можливість перегляду результатів тестування інших відеоадаптерів.

Вхідними даними для програми є база даних, назва відеоадаптеру, роздільна здатність екрану, кількість сгенерованих частинок, режим екрану.

Вихідними даними для програми є результати тестування відеоадаптеру, опис середовища у якому проводилось тестування (назва центрального процесору, кількість оперативної пам'яті, версія операційної системи, версія відеодрайверу, кількість відеопам'яті, версія DirectX) та параметри тестування відеоадаптеру.

1.3.4 Вимоги до видів забезпечення і компонентів

Програма повинна працювати під управлінням операційної системи Windows 7 або вище. У якості інструментальної мови програмування для виконання даної кваліфікаційної роботи слід використовувати мову програмування C# та середовище розробки Microsoft Visual Studio 2017. Для створення користувацьких форм слід використовувати можливості фреймворку Microsoft .NET Framework 4, а для створення стрес-тесту слід використовувати можливості бібліотеки SharpDX.

Для запуску самого стрес-тесту необхідна наявність на комп'ютері пакету SDK для операційної системи Windows так як він містить заголовки, бібліотеки, метаданні та інструменти для створення програмного забезпечення. Для того, щоб відстежувати поточні значення датчиків відеоадаптеру має використовуватися плагін OpenHardwareMonitorLib, який використовує NvAPI (базовий набір для розробки програмного забезпечення NVIDIA) та пошук, що залежить від аргументів (ADL). Для зберігання результатів тестування відеоадаптерів має використовуватися система керування БД Access. Запити до БД здійснюються завдяки SQL запитам.

Для нормального функціонування програми необхідне наступне технічне забезпечення:

- процесор не гірше Intel Celeron E1200 1.6 ГГц;
- кількість оперативної пам'яті не менше 2 Гб формату DDR3 1600 МГц;
- дискретна відеокарта не гірше Nvidia GeForce 8100 GT 256 Мб з підтримкою DirectX 10;
- операційна система не нижче Windows 7;
- жорсткий диск з вільним об'ємом пам'яті у 100 Мб;
- клавіатура і маніпулятор типу «миша».

1.3.5 Обмеження на установку і виконання програми

Програму, що розробляється, можна встановити на будь-який стаціонарний комп'ютер чи ноутбук з операційною системою Windows 7 або вище. Для запуску стрес-тесту необхідна наявність на комп'ютері пакету SDK для операційної системи Windows

1.3.6 Вимоги до програмної документації

Програмна документація повинна включати: керівництво програміста, керівництво оператора та вихідний код програми.

1.3.7 Умови експлуатації

Для запуску програми на персональному комп'ютері повинна бути встановлена дискретна відеокарта, а не лише вбудоване у центральний процесор графічне ядро. Програма повинна функціонувати під управлінням ОС Windows 7/8/10.

1.3.8 Вимоги до маркування та упаковки

Застосунок у своєму повному обсязі може бути записаний на компакт-диск ємністю не менше 100Мб. На упаковці має бути назва застосунку «GPU Test».

2 МАТЕРІАЛИ ТА МЕТОДИ

2.1 Огляд обраної мови програмування, операційної системи та середовища розробки

Згідно з вимогами технічного завдання (п. 1.3.4) для створення застосунку у якості мови необхідно використовувати мову C# та середовище розробки Microsoft Visual Studio 2017. Застосунок має працювати під управлінням ОС Windows 7 або вище. Розглянемо їх можливості.

C# - це сучасна об'єктно-орієнтована мова програмування загального призначення, яка розроблена корпорацією Microsoft і схвалена Європейською асоціацією виробників комп'ютерів (ECMA) та Міжнародною організацією зі стандартизації (ISO). C # був розроблений Андерсом Хейлсбергом і його командою під час розробки програмної технології .Net Framework.

C# розроблений для специфікації загальномовної інфраструктури (CLI), яка складається з коду, що виконується і середовища виконання. Специфікація дозволяє використовувати різні мови високого рівня на різних комп'ютерних платформах і архітектурі.

C# є широко використовуваною професійною мовою внаслідок наступних причин [12], [13]:

- це сучасна мова програмування загального призначення;
- вона об'єктно-орієнтована та компонентно-орієнтована;
- легка у вивченні структурована мова;
- її можна скомпілювати на різних комп'ютерних платформах;
- C# є частиною .Net Framework.

Мова C# дуже схожа на Java і має множину програмних функцій, які роблять її привабливою для широкого кола програмістів в усьому світі. Наведемо список найбільш важливих функцій C #:

- логічні умови;
- автоматичний збір сміття;
- стандартні бібліотеки;

- управління версіями збірки;
- властивості і події;
- делегати і пов'язані з подіями службові методи;
- прості у використанні узагальнені типи (Generics);
- індексатори;
- умовна компіляція;
- проста багатопоточність;
- вирази запитів LINQ і лямбда-вирази;
- інтеграція з ОС Windows.

Інтегроване середовище розробки (IDE) Visual Studio – це інструментарій, який можна використовувати для редагування, налагодження і складання коду, а потім для публікації застосунку. Крім стандартного редактора і відлагоджувача, що надається більшістю IDE, Visual Studio включає в себе компілятори, інструменти автозаповнення коду, графічні конструктори і багато інших функцій, що спрощують процес розробки програмного забезпечення.

У порівнянні з попередньою версією, в Visual Studio 2017 введені наступні зміни [13]:

- новий інтерфейс установки значно прискорює цей процес, забезпечуючи при цьому опції вибору компонентів;
- Visual Studio 17 запускається в два рази швидше, в порівнянні з Visual Studio 15, використовуючи при цьому менше пам'яті;
- підтримка розробки хмарних застосунків за допомогою вбудованого набору інструментів сервісу хмарних обчислень Azure. Visual Studio 17 спрощує налагодження, складання, відлагодження, упаковку і розгортання застосунків і служб в Azure;
- розробка застосунків для Windows з використанням шаблонів UWP. В цьому випадку створюється єдиний проєкт для всіх пристроїв під управлінням Windows 10 – ПК, планшет, телефон, Xbox, Surface Hub і т. д.;

- розробка мобільних застосунків за допомогою платформи Xamarin, яка об'єднує багатоплатформні мобільні вимоги з єдиної кодової базою і набором навичок;
- крос-платформна розробка. Надано можливість взаємної інтеграції робочих процесів фахівців з розробки програмного забезпечення з фахівцями з інформаційно-технологічного обслуговування (DevOps) на SQL Server за допомогою інструментарію Redgate Data Tools і безпечного автоматизованого розгортання баз даних з Visual Studio. Використання .NET Core для створення програмного забезпечення та бібліотек, забезпечить роботу без змін в операційних системах Windows, Linux і macOS;
- розробка ігор. За допомогою інструментів Visual Studio для Unity (VSTU) можна використовувати Visual Studio для написання сценаріїв ігор та редакторів на мові C#, а потім використовувати її потужний відладчик для пошуку і виправлення помилок;
- розробка штучного інтелекту. Інструменти Visual Studio для штучного інтелекту дозволяють використовувати функції Visual Studio для прискорення інновацій в області штучного інтелекту і створювати, тестувати і розгортати рішення глибокого навчання/штучного інтелекту, які легко інтегруються з Azure.

Операційні системи сімейства Windows NT в даний час працюють на процесорах з архітектурою x86, x86-64, Itanium, ARM. Ранні версії (до 4.0 включно) також підтримували деякі RISC-процесори: Alpha, MIPS і PowerPC. Всі операційні системи цього сімейства є повністю 32- або 64-бітними і не потребують ОС MS-DOS навіть для завантаження.

Тільки в цьому сімействі представлені операційні системи для серверів. До версії Windows 2000 включно вони випускалися під тією ж назвою, що і аналогічна версія для робочих станцій, але з додаванням суфікса, наприклад «Windows 2000 Datacenter Server». Починаючи з Windows Server 2003 серверні операційні системи називаються додаванням суфікса «Server» і року випуску. До цього сімейства входять також операційні системи Windows 7, 8, 8.1 та 10.

2.2 Огляд обраних для розробки застосунку технологій

Платформа Microsoft .NET Framework – програмна платформа, випущена компанією Microsoft в 2002 році. Основою платформи є загальномовне середовище виконання CLR, яке підходить для різних мов програмування. Функціональні можливості CLR доступні в будь-яких мовах програмування, що використовують це середовище.

Однією з основних ідей Microsoft .NET є сумісність програмних частин, написаних різними мовами. Наприклад, служба, написана на C++ для Microsoft .NET, може звернутися до методу класу з бібліотеки, написаної на Delphi. На C# можна написати клас, що успадковує від класу, написаного на Visual Basic .NET, а виключення, створене методом, написаним на C#, може бути перехоплено і оброблено в Delphi. Кожна бібліотека (збірка) в .NET має відомості про свою версію, що дозволяє усунути можливі конфлікти між різними версіями збірок.

Середовища розробки, що підтримують .NET:

- Microsoft Visual Studio (C#, Visual Basic .NET, Managed C++, F#);
- SharpDevelop;
- MonoDevelop
- Embarcadero RAD Studio (Delphi for .NET); раніше Borland Developer Studio (Delphi for .NET, C#);
- A#;
- Zonnon;
- PascalABC.NET;
- JetBrains Rider.

Також основною ідеєю при розробці .NET Framework було забезпечення свободи розробника за рахунок надання йому можливості створювати застосунки різних типів, здатні виконуватися на різних типах пристроїв і в різних середовищах.

Другим принципом стала орієнтація на системи, що працюють під управлінням сімейства операційних систем Microsoft Windows.

Далі розглянемо NVAPI – основний комплект програмних розробок NVIDIA, який дозволяє отримати прямий доступ до графічних процесорів та драйверів NVIDIA на всіх платформах Windows. NVAPI надає підтримку категорій операцій, які виходять за рамки тих, що знаходяться у знайомих графічних API, таких як DirectX та OpenGL.

Спочатку NVAPI можна було завантажити лише оригінальним виробникам та розробникам ігор, але тепер він доступний для завантаження всім розробникам, зацікавленим у створенні застосунків Windows на графічних процесорах NVIDIA.

За допомогою NVAPI можна отримати доступ до різних характеристик відеоадаптеру NVIDIA таких як: частота GPU; частота відеопам'яті; шейдерних ядер; температура GPU; швидкість вентиляторів і.т.д.

Також необхідно коротко охарактеризувати SDK-бібліотеку дисплеїв AMD (ADL). Ця бібліотека призначена для доступу до функціональних можливостей драйверів дисплея для відеоадаптерів AMD Radeon та AMD FirePro. Комплект для розробки програмного забезпечення підтримує сімейство графічних процесорів Radeon, включаючи інтегровані у процесор графічні ядра, дискретні відеоадаптери та продукти FirePro. Він також підтримує Windows XP, Vista, 7, 8, 10 та Linux в обох 32 та 64-бітних версіях. SDK можна використовувати як з керованих C/C ++, так і керованих C# програм. Документи та зразки надаються в пакеті для завантаження.

За допомогою ADL можна отримати доступ до різних характеристик відеоадаптеру AMD Radeon таких як: частота GPU; частота відеопам'яті; температура GPU; швидкість вентиляторів; напруга на відеоадаптер і.т.д.

Ще одна бібліотека – SharpDX являє собою керовану .NET-оболонку з відкритим вихідним кодом для створення застосунків за допомогою DirectX. Бібліотека містить класи та методи, які працюють з тривимірним простором та графічними примітивами. На жаль, вона не міститься у .NET Framework, тому

бібліотеку доводиться завантажувати окремо та підключати її до проекту C#. Завдяки SharpDX можна створити 3D вікно зі своєю роздільною здатністю, режимом екрану, розмістити у тривимірний простір будь-які як 2D так і 3D об'єкти та задати будь-який текст.

Наступний компонент для огляду – це DirectX, тобто набір API, розроблених для вирішення завдань, пов'язаних з програмуванням під операційну систему Windows. Найбільш широко використовується при написанні комп'ютерних ігор. Пакет засобів розробки DirectX під Microsoft Windows безкоштовно доступний на сайті Microsoft. Найчастіше оновлені версії DirectX поставляються разом з ігровими застосунками.

Багато сучасні пристроїв мають DirectX-сумісні драйвери, іншими словами, користувач повинен встановити DirectX для використання всіх можливостей пристрою. На даний момент система оновлення драйверів Windows Update дозволяє завантажувати тільки необхідні бібліотеки, а не весь драйвер цілком. DirectX підтримує усі версії Windows починаючи з версії 95 та закінчуючи 10.

Далі розглянемо OpenGL (Open Graphics Library) - специфікацію, що визначає платформонезалежний (незалежний від мови програмування) програмний інтерфейс для створення програмного забезпечення, що використовує двовимірну і тривимірну комп'ютерну графіку. Вона включає більш ніж 300 функцій для малювання складних тривимірних сцен з простих примітивів. Використовується при створенні комп'ютерних ігор, систем автоматизованого проектування, віртуальної реальності, візуалізації в наукових дослідженнях.

Стандарт OpenGL, з появою нових технологій, дозволяє окремим виробникам додавати в бібліотеку функціональність через механізм розширень. Розширення поширюються за допомогою двох складових: заголовка, в якому знаходяться прототипи нових функцій і констант, а також драйверу пристрою, що поставляється розробником. Кожен виробник має аббревіатуру, яка використовується при іменуванні його нових функцій і констант. Наприклад,

компанія NVIDIA має аббревіатуру NV, яка використовується при іменуванні її нових функцій.

Також буде використана технологія обчислень загального призначення на графічних процесорах (GPGPU). GPGPU – це використання GPU, який зазвичай обробляє обчислення тільки для комп'ютерної графіки, для виконання обчислень у застосунках, традиційно оброблюваних центральним процесором. Використання декількох відеокарт в одному комп'ютері або великої кількості графічних чіпів призвело до ще більшого розпаралелювання і без того паралельного характеру обробки графіки. Крім того, навіть одна структура GPU-CPU забезпечує переваги, які кілька процесорів самі по собі не пропонують через спеціалізацію кожного чіпу. По суті, конвеєр GPGPU - це свого роду паралельна обробка між одним або декількома графічними процесорами і процесорами, які аналізують дані, надані у вигляді зображення або іншої графічної форми. Хоча графічні процесори працюють на більш низьких частотах, у них зазвичай у багато разів більше ядер. Таким чином, графічні процесори можуть обробляти набагато більше зображень і графічних даних в секунду, ніж традиційні процесори [15].

Для тестування комп'ютерної системи у цілому буде використовуватися OpenHardwareMonitorLib –безкоштовний плагін з відкритим кодом, який працює з операційною системою Windows Vista та вище разом з Microsoft .NET Framework версії 2.0 та вище. Плагін використовує багато API завдяки яким можна відстежувати у режимі реального часу характеристики різних компонентів апаратної системи персонального комп'ютера таких як: центральний процесор; оперативна пам'ять; відеоадаптер; материнська плата та жорсткий диск. Серед API, які використовує плагін, присутні NVAPI та ADL.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Розробка класів застосунку

Програма «GPU Test» реалізована за допомогою п'яти класів, які взаємодіють між собою.

Клас «UserForm» реалізує перший користувацький інтерфейс застосунку. У його конструкторі створюється об'єкт класу DiscreteVideocard та викликається метод, за поверненим значенням якого можна перевірити, чи присутній відеоадаптер у ПК, чи ні. Якщо ні, то програма не запуститься та з'явиться відповідне повідомлення. «UserForm» надає можливість обрати параметри стрес-тесту відеоадаптеру такі як: роздільна здатність екрану; режим екрану та кількість частинок, що генеруються. Він створює об'єкт класу «Logic», передає у його конструктор значення зазначених вище параметрів та запускає стрес-тест.

Клас «Logic» реалізує проведення стрес-тесту відеоадаптеру. Відразу після запуску стрес-тесту запускається таймер, який відображається у форматі година:хвилина:секунда. У цьому класі створені об'єкти класів бібліотеки SharpDX, властивості яких відповідають за створення вікна, у якому проводиться стрес-тест з такими параметрами: розгортання вікна на весь екран; довжина та ширина вікна у пікселях; вертикальна синхронізація і відстежування натискання на клавіші клавіатури. Також у класі «Logic» створюється об'єкт плагіну OpenHardwareMonitorLib завдяки якому, можна отримувати значення датчиків відеоадаптеру та постійно змінювати їх використовуючи рекурсивні функції та цикли. Також у «Logic» створюється об'єкт класу «Particles» у конструктор якого передається розмір масиву (кількість частинок) після чого генерується масив, кожний елемент якого – це частинка. У цьому класі також присутня функція, яка за зазначеними координатами x та y розміщує текст у вікні.

Клас «Particles» реалізує динамічне створення масиву з переданої кількості частинок у конструктор з класу «Logic». Кожний елемент масиву – це 2D об’єкт, який має свою текстуру (рис 3.1) та координати x , y , z . Координати завдаються завдяки циклам та рекурсивним функціям. Також у цьому класі реалізовано відмалювання текстури частинки.

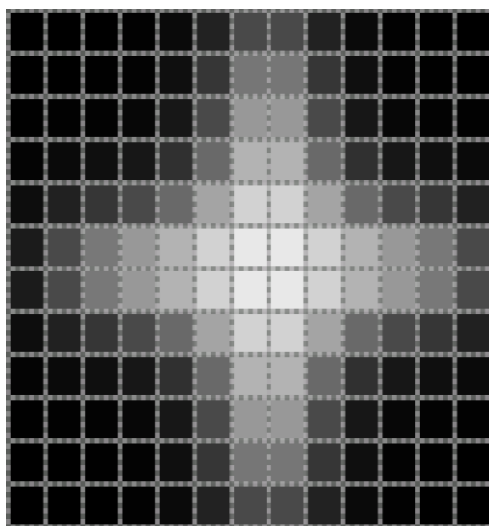


Рисунок 3.1 – Текстура частинки

Клас «PastTest» починає роботу після завершення стрес-тесту, реалізує другий користувацький інтерфейс, який у конструкторі отримує результати стрес-тесту відеоадаптеру. Цей клас має методи, які читають та записують інформацію у БД. У ньому створені об’єкти класів простору імен Win32 фреймворку .NET Framework, завдяки яким можна отримати назву центрального процесору, кількість оперативної пам’яті та відеопам’яті, версію відеодрайверу, назву операційної системи.

Клас «DiscreteVideocard» надає можливість перевірити, чи присутній відеоадаптер у ПК, чи існує лише вбудоване у центральний процесор графічне ядро. Ця перевірка здійснюється завдяки підключенням до класу OpenGL бібліотекам та методу, який повертає строку, яка містить у собі назву відеоадаптеру та ім’я шини відеокарти (PCIe/SSE2) . Якщо рядок не містить

ім'я відеокарти, то робиться висновок, що відеоадаптер відсутній у ПК та програма не запускається.

На рис. 3.2 зображено діаграму класів програмного забезпечення «GPU Test».

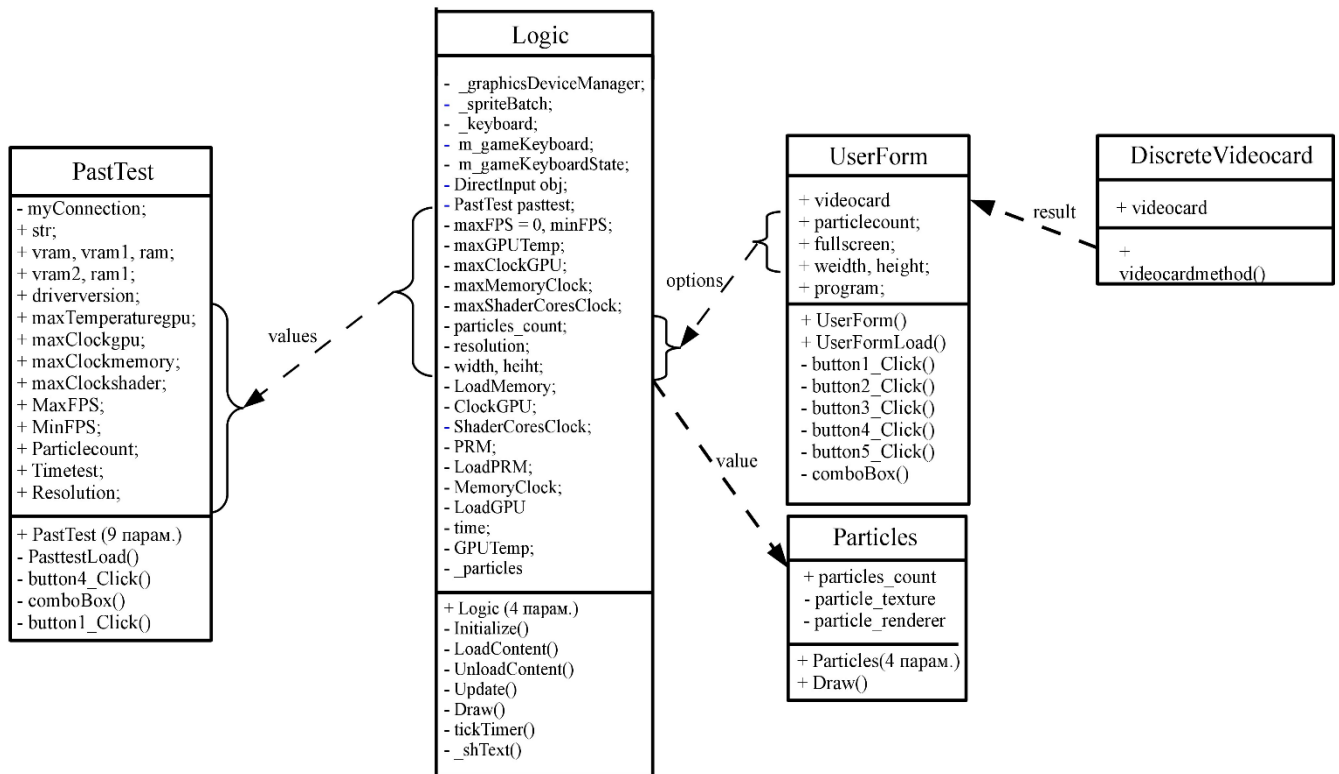


Рисунок 3.2 – Діаграма класів програми «GPU Test»

3.2 Розробка програмних модулів

Нижче будуть наведені методи розроблених класів, що реалізують програмне забезпечення та їх опис. Клас «Logic» програми «GPU Test» реалізується наступним чином:

```

public class Logic
    public Logic(int partcount, bool fulscreen, int weidth, int
height)

    protected override void Initialize()
    protected override void LoadContent()
    protected override void UnloadContent()
  
```

```
protected override void Update(GameTime gameTime)
protected override void Draw(GameTime gameTime)
private void tickTimer(object sender, EventArgs e)
private void _shText(string text, Vector2 position, Color
color)
```

Призначення складових класу наступне:

- Logic, це конструктор класу «Logic», створює вікно та запускає таймер;
- Initialize завдає назву title вікну на верхній панелі;
- LoadContent створює об'єкт класу «Particles» та завдає шрифт тексту;
- UnloadContent виводить створений масив у вікно;
- Update оновлює логіку стрес-тесту;
- Draw викликає функцію для виводу тексту у вікні, виводить значення датчиків відеоадаптеру, кількість FPS та таймер;
- tickTimer призначений для оновлення таймеру;
- _shText задає колір та координати тексту, який потрібно вивести.

Клас «UserForm» реалізується наступним чином:

```
public partial class UserForm
public UserForm()
private void UserForm_Load(object sender, EventArgs e)
private void button1_Click(object sender, EventArgs e)
private void button2_Click(object sender, EventArgs e)
private void button3_Click(object sender, EventArgs e)
private void button4_Click(object sender, EventArgs e)
private void button5_Click(object sender, EventArgs e)
private void comboBox1 (object sender, EventArgs e)
```

Призначення складових класу наступне:

- UserForm, це конструктор класу «UserForm», створює об'єкт класу «DiscreteVideocard» та отримує рядок з відеоадаптером та його шиною;

- `UserForm_Load` перевіряє, чи зазначено у рядку, який получено у конструкторі, шина відеоадаптеру, якщо ні, то видається відповідне повідомлення та програма не запускається;
- `button1_Click` запускає стрес-тест з обраною користувачем роздільною здатністю екрану;
- `button2_Click` запускає стрес-тест з роздільною здатністю екрану 2048x1080 пікселей;
- `button3_Click` запускає стрес-тест з роздільною здатністю екрану 1920x1080 пікселей;
- `button4_Click` запускає стрес-тест з роздільною здатністю екрану 1280x720 пікселей;
- `button5_Click` запускає стрес-тест з роздільною здатністю екрану 4196x2160 пікселей;
- `comboBox1` слідує за тим, чи обрана роздільна здатність екрану із наданого списку, чи користувач хоче встановити свої значення.

Клас «`PastTest`» реалізується наступним чином:

```
public partial class PastTest
{
    public PastTest(string maxTemperatureGpu, string maxClockGPU,
string maxClockMemory, string maxClockShader, int maxFPS, int
minFPS, int particlecount, string resolution, string timetest)
    {
    }
    private void PastTest_Load(object sender, EventArgs e)
    {
    }
    private void button4_Click(object sender, EventArgs e)
    {
    }
    private void comboBox1(object sender, EventArgs e)
    {
    }
    private void button1_Click(object sender, EventArgs e)
    {
    }
}
```

Призначення складових класу наступне:

- `PastTest`, це конструктор класу «`PastTest`», який у свої параметри отримує значення, що сформувались у ході виконання стрес-тесту та виводить їх, а також виводить інформацію про середовище (назва центрального процесору; кількість оперативної пам'яті; кількість відеопам'яті; версія ОС, версія відеодрайверу) у якому проводився стрес-тест відеоадаптеру;

- PastTest_Load забезпечує зчитування бази даних та заповнення списку назвами відеоадаптерів, які раніше проходили стрес-тест;
- button4_Click додає результати тестування до БД;
- comboBox1 забезпечує вибір назви відеоадаптеру зі списку та зчитування запису з БД за відповідним індексом обраного елемента зі списку;
- button1_Click – повернення до значень проведеного тесту, якщо був обраний інший відеоадаптер зі списку.

Клас «DiscreteVideocard» реалізується наступним чином:

```
public class DiscreteVideocard
    public string videocardmethod()
```

Метод videocardmethod повертає строку, яка містить у собі назву відеоадаптеру та ім'я його шини (PCIe/SSE2).

Клас «Particles» реалізується наступним чином:

```
public class Particles
    public Particles(GraphicsDevice graphics, Effect particlesRender,
Texture2D particleTexture, int partcount)
    public void Draw(GraphicsDevice graphics)
```

Призначення складових класу наступне:

- Particles, це конструктор класу «Particles», ініціалізує та заповнює масив об'єктами частинок та привласнює їм координати x,y,z. з розмірністю, відповідною до переданого значення кількості частинок у якості параметру;
- Draw – накладення текстури на об'єкт частинки та її відмалювання.

Код програми наведено у Додатку А.

3.3 Розробка інтерфейсу користувача

При розробці користувацьких інтерфейсів були враховані вимоги до них, що були перелічені у п. 1.3.3 технічного завдання. Головна задача при розробці інтерфейсів користувача – це зробити якомога зручний,

ненавантажений надлишковою інформацією, простий у використанні інтерфейс.

В програмі наявні дві користувацькі форми. Перша – головна, з якої починається програма та відбувається запуск стрес-тесту відеоадаптера, друга відкривається вже після проведення стрес-тесту. Обидві користувацькі форми відрізняються між собою лише кількістю елементів та мають однакову структуру.

Для ознайомлення зі структурою користувацьких форм у фреймворку .NET Framework нижче наведено частину коду файлу UserForm.Designer.cs. Оскільки файл містить однакові (дублюючі) функції для елементів одного типу (кнопка, текстове поле, та ін.) – для розуміння структури файлу достатньо буде ознайомлення тільки з частиною коду:

```
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();
    }
    base.Dispose(disposing);
}
private void InitializeComponent()
{
    System.ComponentModel.ComponentResourceManager
resources = new
System.ComponentModel.ComponentResourceManager(typeof(UserForm));
    this.label1 = new System.Windows.Forms.Label();
    this.label2 = new System.Windows.Forms.Label();
    this.label3 = new System.Windows.Forms.Label();

    ...

    // label1
    this.label1.AutoSize = true;
    this.label1.BackColor =
System.Drawing.Color.Transparent;
    this.label1.Font = new System.Drawing.Font("Microsoft
Sans Serif", 18F, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
    this.label1.ForeColor =
System.Drawing.Color.FromArgb(((int) (((byte) (128)))),
((int) (((byte) (255))))), ((int) (((byte) (255)))));
    this.label1.Location = new System.Drawing.Point(147,
6);
```

```

        this.label1.Name = "label1";
        this.label1.Size = new System.Drawing.Size(67, 29);
        this.label1.TabIndex = 1;
        this.label1.Text = "GPU";
        // label2
    ...

    // label3
    ...

    // checkBox1
    this.checkBox1.AutoSize = true;
    this.checkBox1.Font = new
System.Drawing.Font("Microsoft Sans Serif", 11.25F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
    this.checkBox1.ForeColor =
System.Drawing.Color.FromArgb(((int) (((byte) (128))))),
    ((int) (((byte) (255))))), ((int) (((byte) (255)))));
    this.checkBox1.Location = new System.Drawing.Point(42,
80);

    this.checkBox1.Name = "checkBox1";
    this.checkBox1.Size = new System.Drawing.Size(95, 22);
    this.checkBox1.TabIndex = 8;
    this.checkBox1.Text = "Fullscreen";
    this.checkBox1.UseVisualStyleBackColor = true;
    this.checkBox1.CheckedChanged += new
System.EventHandler(this.checkBox1_CheckedChanged);
    // comboBox1
    this.comboBox1.Font = new
System.Drawing.Font("Microsoft Sans Serif", 9.75F,
System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
    this.comboBox1.FormattingEnabled = true;
    this.comboBox1.Items.AddRange(new object[] {
        "800x600",
        "1024x768",
        "1280x720 (HD) ",
        "1600x900",
        "1920x1080 (Full HD) ",
        "2048x1080 (2K) ",
        "2560x1440",
        "4096x2160 (4K) ",
        "Custom"});
    this.comboBox1.Location = new System.Drawing.Point(42,
133);

    this.comboBox1.Name = "comboBox1";
    this.comboBox1.Size = new System.Drawing.Size(121,
24);

    this.comboBox1.TabIndex = 9;
        this.comboBox1.SelectedIndexChanged += new
System.EventHandler(this.comboBox1_SelectedIndexChanged);

```

```

...

    // button1
    this.button1.BackColor =
System.Drawing.SystemColors.ButtonHighlight;
    this.button1.Font = new System.Drawing.Font("Microsoft
Sans Serif", 11.25F, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, ((byte) (204)));
    this.button1.ForeColor = System.Drawing.Color.Black;
    this.button1.Location = new System.Drawing.Point(42,
291);

    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(131, 31);
    this.button1.TabIndex = 13;
    this.button1.Text = "Start test";
    this.button1.UseVisualStyleBackColor = false;
        this.button1.Click += new
System.EventHandler(this.button1_Click);

...

    // UserForm
    this.AutoScaleDimensions = new
System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
    this.BackColor =
System.Drawing.SystemColors.ActiveCaptionText;
    this.ClientSize = new System.Drawing.Size(399, 450);
    this.Controls.Add(this.comboBox2);
    this.Controls.Add(this.label8);
    this.Controls.Add(this.textBox3);
    this.Controls.Add(this.textBox2);
        this.Controls.Add(this.button6);

```

Місця пропуску коду позначено «...». Наданий вище код ініціалізує елементи користувацького інтерфейсу і саму форму. Для кращого розуміння опис кожного елементу відокремлений наступним коментарем:

```
// name
```

Функція Dispose очищає будь-яку використану базу ресурсів. Вона є системною, створюється автоматично засобами Visual Studio і не потребує модифікації.

Функція InitializeComponent запускається на початку роботи форми і відповідає за створення і розташування елементів керування інтерфейсу.

Кожен графічний елемент інтерфейсу (кнопка, список, текстове поле та інші) має наступні властивості:

- .Location - позиція елемента відносно форми;
- .Name – ім'я елемента в програмному коді;
- .Size – розмір елемента на формі;
- .Text – текстове повідомлення, яке міститься на елементі і відображається користувачеві.

В деяких графічних елементах наявні додаткові властивості, що враховують особливість елемента. Наприклад для елемента PictureBox, який зберігає і відтворює зображення наявна властивість Image, яка зберігає адресу зображення. Деякі властивості можуть бути відсутні. У такому разі для них буде застосовано значення за замовчуванням.

Блок UserForm описує властивості самої форми, тому його властивості відрізняються від інших блоків

В результаті обробки файлів розмітки формується інтерфейс вікна. За результатами обробки UserForm.Designer.cs створюється інтерфейс першої користувацької форми програми. За результатами обробки PastTest.Designer.cs створюється інтерфейс другої користувацької форми програми.

Повний код файлу UserForm.Designer.cs наведено у додатку А.

4 КЕРІВНИЦТВО ПРОГРАМІСТА

4.1 Призначення і умови застосування програми

Робоче ім'я програмного забезпечення – «GPU Test». Програма призначена для проведення стрес-тесту відеоадаптера.

Область застосування - перевірка працездатності та продуктивності компонентів апаратної системи персонального комп'ютера.

Застосунок дає можливість користувачу переконуватись у тому чи працездатний та достатньо потужний для складних 3D задач відеоадаптер, який встановлений у ПК.

Для нормального функціонування застосунку необхідний стаціонарний комп'ютер чи ноутбук з наступними апаратними характеристиками:

- процесор не гірше Intel Celeron E1200 1.6 ГГц;
- кількість оперативної пам'яті не менше 2 Гб формату DDR3 1600 МГц;
- дискретна відеокарта не гірше Nvidia GeForce 8100 GT 256 Мб з підтримкою DirectX 10;
- операційна система не нижче Windows 7;
- жорсткий диск з вільним об'ємом пам'яті у 100 Мб;
- клавіатура і маніпулятор типу «миша».

Також для функціонування програми необхідна наявність ОС Windows, не нижче версії 7 з встановленим .NET Framework версії не нижче 4.0.

4.2 Характеристики програми

4.2.1 Юзабіліті характеристики програми

До зручності у використанні застосунку «GPU Test» пред'являються наступні вимоги:

- програма реагує на натискання клавіш миші та клавіатури з затримкою не більше 0.5с;

- програма надає можливість обирати параметри стрес-тесту відеоадаптеру такі як роздільна здатність та режим вікна стрес-тесту, кількість генерованих частинок;
- програма надає можливість користувачу під час стрес-тесту відстежувати значення датчиків різних характеристик відеоадаптеру, кількість кадрів за секунду та час тесту;
- програма передбачає можливість зупину стрес-тесту натисканням на кнопку Esc або якщо температура GPU досягла критичної – 80 градусів;
- програма п надає можливість користувачу після завершення тестування відеоадаптеру переглядати результати, програмні та апаратні характеристики, при яких стрес-тест було проведено, результати тестування інших відеоадаптерів та внесення усіх поточних результатів до БД.

4.2.2 Загальні характеристики програми та її структура.

Програмне забезпечення «GPU Test» займає 15,5 Мб на жорсткому диску. Програма запускається у разі потреби стрес-тестування відеоадаптера.

Перелік файлів, з яких складається програма:

- UserForm.cs – код-логіка першої користувацької форми;
- UserForm.Designer.cs – інтерфейс першої користувацької форми;
- UserForm.resx – налаштування першої користувацької форми;
- PastTest.cs – код-логіка другої користувацької форми;
- PastTest.Designer.cs – інтерфейс другої користувацької форми;
- PastTest.resx – налаштування другої користувацької форми;
- Logic.cs – файл класу «Logic»;
- Particles.cs – файл класу «Particles»;
- DiscreteVideocard.cs – файл класу «DiscreteVideocard»;
- Particle16x16.dds – текстура частинки;
- DefaultFont.xml – налаштування тексту (шрифт, розмір), що виводиться під час стрес- тесту;

- ParticleRender.fx – код-логіка для відмалювання частинки;
- Program.cs – вхідна точка програми;
- App.config – користувацькі налаштування програми;
- Resources – папка, у якій містяться зображення при розробці програми.

4.3 Початкові та вихідні дані

Початкові дані: назва відеоадаптеру; роздільна здатність екрану; кількість частинок, що генеруються; режим екрану.

Вихідні дані: результати тестування відеоадаптеру; опис середовища у якому проводилось тестування (назва центрального процесору, кількість оперативної пам'яті, версія операційної системи, версія відеодрайверу, кількість відеопам'яті, версія DirectX) та параметри тестування відеоадаптеру.

4.4 Повідомлення програмісту

Повідомлення програмісту у програмі не передбачені.

5 КЕРІВНИЦТВО ОПЕРАТОРА

5.1 Призначення програми

Робоче ім'я програмного застосунку – «GPU Test». Програма призначена для проведення стрес-тесту відеоадаптера. Тест полягає у моделюванні системи динамічного руху частинок, що мають свій колір та текстуру. Кожну таку частинку повинен відобразити відеоадаптер. Кількість частинок може досягати декількох мільйонів. Стрес-тест надає можливість перевірити наступні характеристики відеоадаптеру:

- швидкодія графічного процесору;
- швидкодія відеопам'яті;
- швидкодія шейдерних ядер (якщо вони існують);
- ефективність системи охолодження.

Область застосування – перевірка працездатності та продуктивності компонентів апаратної системи персонального комп'ютера.

5.2 Умови використання програми

Умови використання застосунку зазначені у технічному завданні (див п.п. 1.3.4 та 1.3.7).

5.3 Виконання програми

Для початку роботи з програмою необхідно запустити файл «GPU Test.exe».

На рис. 5.1 наведено першу користувацьку форму програми «GPU Test» з позначками елементів керування.

Нижче наведено пояснення позначок.

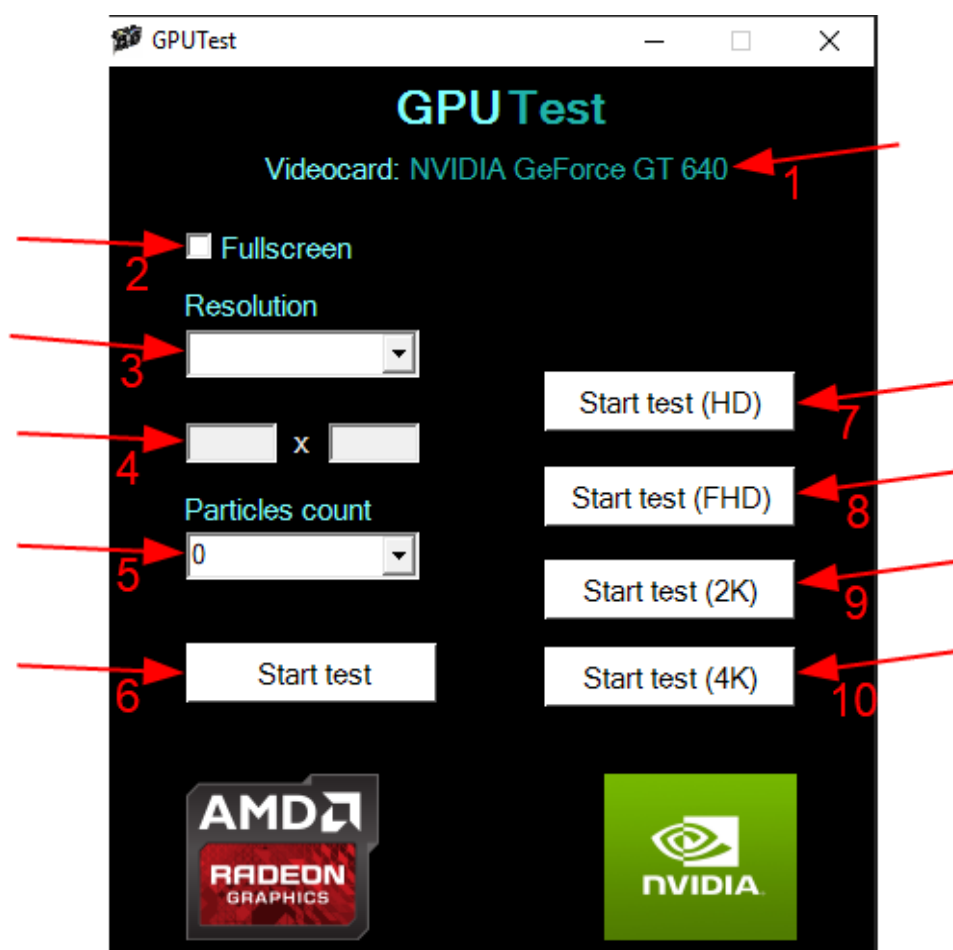


Рисунок 5.1 – Перша користувацька форма програми «GPU Test» з позначками елементів керування

- 1 – назва відеоадаптеру;
- 2 – повноекранний режим вікна стрес-тесту;
- 3 – роздільна здатність екрану;
- 4 – встановлення роздільної здатності екрану у форматі «довжина x ширина»;
- 5 – кількість частинок під час стрес-тесту;
- 6 – запуск стрес-тесту з зазначеними вище параметрами;
- 7 – запуск стрес-тесту з роздільною здатністю екрану 1280x720 пікселів;
- 8 – запуск стрес-тесту з роздільною здатністю екрану 1920x1080 пікселів;
- 9 – запуск стрес-тесту з роздільною здатністю екрану 2048x1080 пікселів;

10 – запуск стрес-тесту з роздільною здатністю екрану 4096x2160 пікселів.

На рис. 5.2 наведено інформацію, яка виводиться у вікні під час стрес-тесту відеоадаптеру програми «GPU Test» з відповідними позначками.

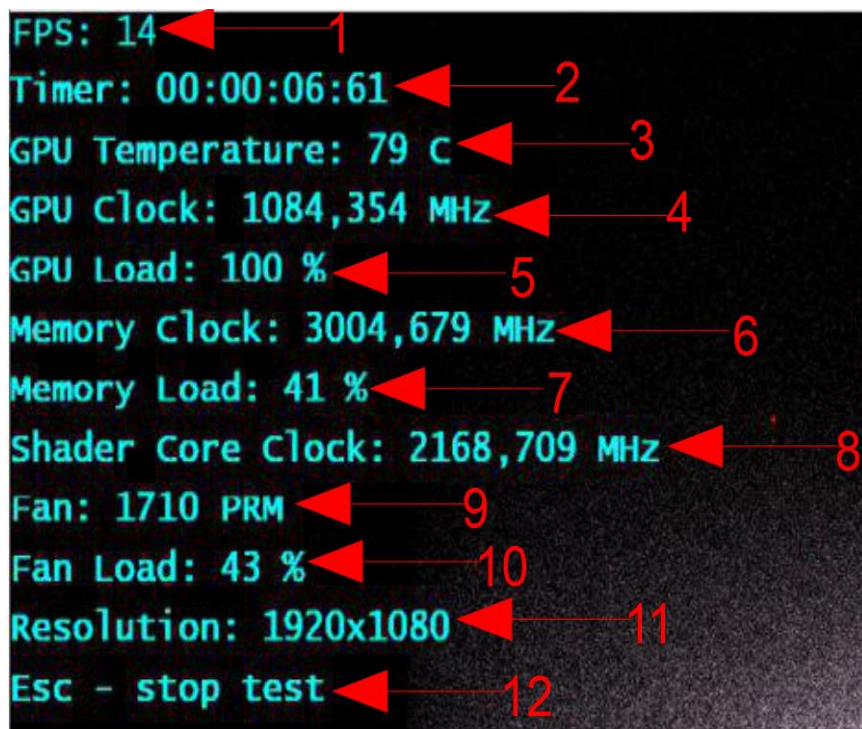


Рисунок 5.2 - Інформація, яка виводиться у вікні під час стрес-тесту відеоадаптеру програми «GPU Test»

На рис. 5.3 наведено другу користувацьку форму програми «GPU Test» з позначками елементів керування.

Нижче наведено пояснення позначок.

1 – назва відеоадаптеру, результати тестування відеоадаптеру, опис середовища у якому проводилось тестування (назва центрального процесору, кількість оперативної пам'яті, версія операційної системи, версія відеодрайверу, кількість відеопам'яті, версія DirectX) та параметри стрес - тесту відеоадаптера;

2 – вибір відеоадаптера з БД для перегляду результатів тестування, опису середовища у якому проводилось тестування та параметрів стрес-тесту;

3 – повернення до поточних результатів тестування, якщо обраний відеоадаптер з БД;

4 – внесення усіх поточних результатів тестування відеоадаптера до БД.

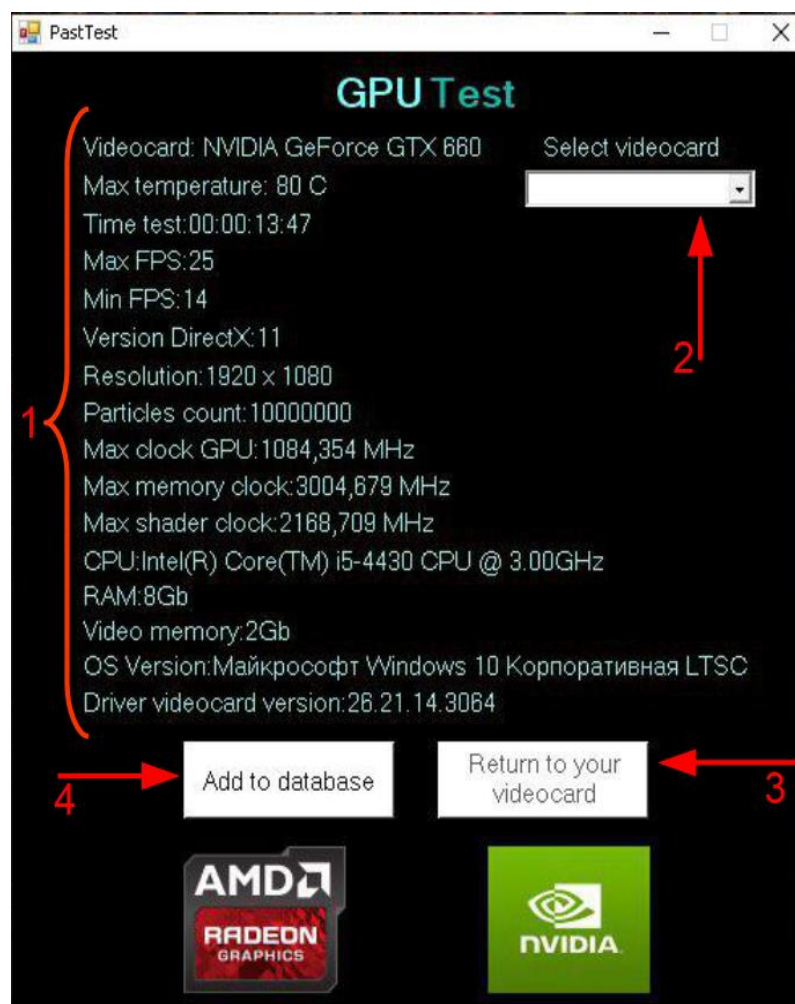


Рисунок 5.3 - Друга користувацька форма програми «GPU Test» з позначками елементів керування.

5.4 Повідомлення оператора

У програмі «GPU Test» передбачено п'ять повідомлень оператору.

На рис. 5.4 наведено перше повідомлення. Воно має текст «Videocard not found!» та кнопку «ОК» яка закриває це повідомлення. Таке повідомлення з'являється лише в тому випадку, якщо у момент запуску програми,

відеоадаптер у ПК не встановлений. Після закриття повідомлення програма не запускається.



Рисунок 5.4 – Повідомлення, якщо у момент запуску програми, відеоадаптер у ПК не встановлений.

На рис. 5.5 наведено друге повідомлення. Воно має текст «Select a screen resolution for the test!» та кнопку «OK» яка закриває це повідомлення. Таке повідомлення з'являється лише в тому випадку, якщо у момент натискання на кнопку «Start test», яка знаходиться на першій користувацькій формі, не обрано роздільну здатність вікна стрес-тесту відеоадаптера. Після закриття повідомлення стрес-тест не запускається.

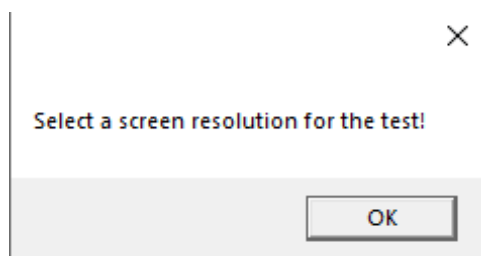


Рисунок 5.5 – Повідомлення, якщо у момент натискання на кнопку «Start test» не обрано роздільну здатність вікна

На рис. 5.6 наведено третє повідомлення. Воно має текст «Set the correct number of particles!» та кнопку «OK» яка закриває це повідомлення. Таке повідомлення з'являється лише в тому випадку, якщо у момент натискання на кнопку запуску стрес-тесту відеоадаптера, не обрано кількість частинок для генерації. Після закриття повідомлення стрес-тест не запускається.

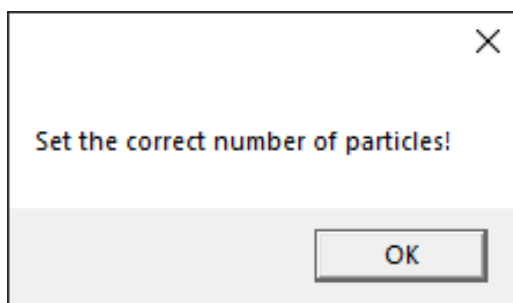


Рисунок 5.6 – Повідомлення, якщо у момент натискання на кнопку запуску стрес-тесту, не обрано кількість частинок для генерації

На рис. 5.7 наведено четверте повідомлення. Воно має текст «Critical temperature GPU 80 C!» та кнопку «OK» яка закриває повідомлення та вікно стрес-тесту відеоадаптера. Таке повідомлення з'являється лише в тому випадку, якщо у момент проведення стрес-тесту температура графічного процесора досягла небезпечних для відеоадаптера 80 градусів. Після появи повідомлення стрес-тест негайно зупиняється.

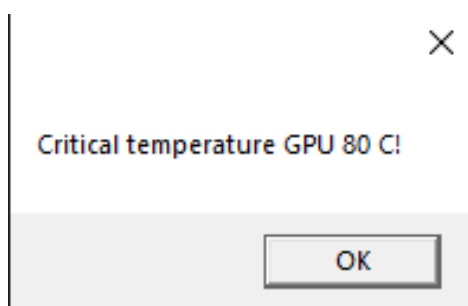


Рисунок 5.7 – Повідомлення, якщо у момент проведення стрес-тесту температура графічного процесора досягла 80 градусів

На рис. 5.8 наведено п'яте повідомлення. Воно має текст «Info test add to database successfully!» та кнопку «OK» яка закриває це повідомлення. Таке повідомлення з'являється, якщо натиснута кнопка «Add to database», яка знаходиться на другій користувацькій формі.

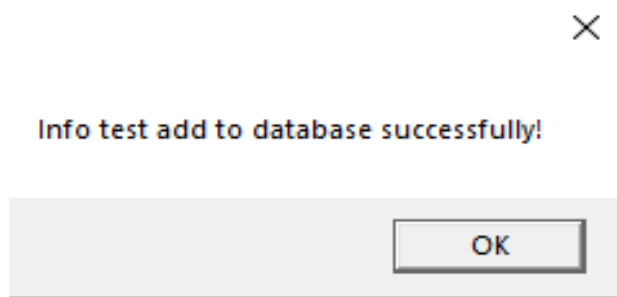


Рисунок 5.8 – Повідомлення, якщо натиснута кнопка «Add to database»

ВИСНОВКИ

Під час розробки даної дипломної кваліфікаційної роботи бакалавра було проведено огляд відомих типів тестів відеоадаптеру та аналіз існуючих аналогів.

Мовою програмування для створення програмного продукту було обрано C#, так як її можливостей вистачало для реалізації функціоналу ПЗ.

За середовище розробки було обрано Microsoft Visual Studio 2017, так як воно надає можливості створення користувацьких інтерфейсів з різноманітними елементами керування, підтримує імпорт бібліотек до проєкту як із фреймворку .NET Framework, так і із сторонніх джерел та має високу адаптованість до ОС Windows.

В результаті виконання кваліфікаційної роботи, було створено програмний продукт «GPU Test» тестування відеоадаптеру за допомогою стрес-тесту, який використовує відеоадаптер на повну потужність. Тест полягає у моделюванні системи динамічного руху частинок, що мають свій колір та текстуру. Кількість частинок може досягати декількох мільйонів.

Стрес-тест надає можливість перевірити наступні характеристики відеоадаптеру:

- швидкодія графічного процесору;
- швидкодія відеопам'яті;
- швидкодія шейдерних ядер (якщо вони існують);
- ефективність системи охолодження.

Розроблено комплект необхідної програмної документації, що включає в себе, опис програми, керівництво програміста та керівництво оператора.

Застосунок може експлуатуватись у якості програми перевірки працездатності та продуктивності компонентів апаратної системи персонального комп'ютера.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Melendez, S. The Purpose of a Graphics Card [Electronic resource] / S. Melendez. – Access mode: <https://smallbusiness.chron.com/purpose-graphics-card-55327.html>.
2. Walton, J. How to Test Your Graphics Card [Electronic resource] / J. Walton. – Access mode: <https://www.tomshardware.com/features/gpu-tests>.
3. Что означает 1% и 0,1% при тестировании видеокарт и процессоров в играх [Электрон. ресурс]. – Режим доступа: <https://club.dns-shop.ru/blog/t-99-videokartyi/37604-chto-oznachaet-1-i-0-1-pri-testirovanii-videokart-i-protssessorov/>.
4. В синтетических тестах производительности можно повысить результаты и без разгона [Электрон. ресурс]. – Режим доступа: <https://overclockers.ru/hardnews/show/89217/v-sinteticheskikh-testah-proizvoditelnosti-mozhno-povysit-rezultaty-i-bez-razgona>.
5. Ахмедов, Б. Тестирование видеокарт [Электрон. ресурс] / Б. Ахмедов. – Режим доступа: <https://compress.ru/article.aspx?id=9579>.
6. Ходаковский, К. Basemark GPU 1.0 позволяет сравнить производительность ПК и смартфона [Электрон. ресурс] / К. Ходаковский. – Режим доступа: <https://3dnews.ru/971541>.
7. FurMark [Электрон. ресурс]. – Режим доступа: <http://radeon.ru/downloads/utls/furmark/>.
8. Обзор процессора Intel Core i9-10980XE Extreme Edition [Электрон. ресурс]. – Режим доступа: <https://3dnews.ru/998360/obzor-intel-core-i910980xe>.
9. Очередной успех Nvidia. Сеть распределённых вычислений Folding@Home получила поддержку CUDA [Электрон. ресурс]. – Режим доступа: <https://www.ixbt.com/news/2020/09/29/nvidia-folding-home-cuda.html>.
10. Обзор: Luxmark измеряет производительность графики OpenCL 2021 [Электрон. ресурс]. – Режим доступа: <https://ru.joecomp.com/review-luxmark-measures-opencl-graphics-performance>.

11. MSI Afterburner [Электрон. ресурс]. – Режим доступа: <https://softcatalog.info/ru/programmy/msi-afterburner>.

12. C# - Overview [Electronic resource]. – Access mode: https://www.tutorialspoint.com/csharp/csharp_overview.htm.

13. C# Introduction [Electronic resource]. – Access mode: https://www.w3schools.com/cs/cs_intro.asp.

14. What's new in Visual Studio 2017 [Electronic resource]. – Access mode: <https://docs.microsoft.com/en-us/visualstudio/ide/whats-new-visual-studio-2017?view=vs-2017>.

15. Универсальные вычисления на графических процессорах - General-purpose computing on graphics processing units [Электрон. ресурс]. – Режим доступа: https://ru.qaz.wiki/wiki/General-purpose_computing_on_graphics_processing_units.

ДОДАТОК А
Текст програми

A.1 Код файла Program.cs

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using OpenTK;
using OpenTK.Graphics;
using OpenTK.Graphics.OpenGL;
using OpenTK.Input;
using System.Linq;
namespace GPUParticles
{
    /// Simple GPUParticles application using SharpDX.Toolkit.
    class Program
    {
        /// Defines the entry point of the application.
#if NETFX_CORE
        [MTAThread]
#else
        [STAThread]
#endif
        static void Main()
        {
            Application.Run(new UserForm());
        }
    }
}

```

A.2 Код файла Logic.cs

```

using System;
using System.Text;
using SharpDX;
using System.Linq;
using System.Windows.Forms;

```

```

using System.Runtime.InteropServices;
using System.IO;
using SharpDX.DirectInput;
using System.Xml;
using SharpDX.Toolkit.Graphics;
using SharpDX.Windows;
namespace GPUParticles
{
    using SharpDX.Toolkit;
    using SharpDX.Toolkit.Graphics;
    using SharpDX.Toolkit.Input;
    using PsychomixEngine.Dev;
    using GPUParticles.Particles;
    using PsychomixEngine.Render.Camera;
    using SharpDX.DirectInput;
    using SharpDX.DXGI;

    public class Logic : Game
    {
        private GraphicsDeviceManager _graphicsDeviceManager;
        private SpriteBatch _spriteBatch;
        private readonly KeyboardManager _keyboard;
        private readonly MouseManager _mouse;
        private Keyboard m_gameKeyboard;
        private SharpDX.DirectInput.KeyboardState m_gameKeyboardState;
        private DirectInput obj;
        public PastTest pasttest;
        private int maxFPS = 0, minFPS;
        private string maxGPUTemp;
        private string GPUTemp;
        private string maxClockGPU;
        private string ClockGPU;
        private string LoadGPU;
        private string maxMemoryClock;
        private string MemoryClock;
        private string LoadMemory;
        private string maxShaderCoresClock;
    }
}

```

```

private string ShaderCoresClock;
private string PRM;
private string LoadPRM;
public string time;
private string width, heiht;
public string resolution;
public int i = 0;
private SpriteFont _defaultFont;
DateTime date;
private GPUParticlesHandler _particles;
private PerspecitveCamera _camera;
public int countsensors;
public string name, type;
public int particlescount;
public Logic(int partcount, bool fullscreen, int weidth, int
height)
{
    resolution = weidth.ToString() + " x " + height.ToString();
    obj = new DirectInput();
    width = weidth.ToString();
    heiht = height.ToString();
    particlescount = partcount;
    _graphicsDeviceManager = new GraphicsDeviceManager(this)
    {
        IsFullScreen = fullscreen,
        SynchronizeWithVerticalRetrace = false,
        PreferredBackBufferWidth = weidth,
        PreferredBackBufferHeight = height,
        PreferMultiSampling = true,
        PreferredDepthStencilFormat = DepthFormat.None,
        DeviceCreationFlags
SharpDX.Direct3D11.DeviceCreationFlags.Debug
    };
    IsFixedTimeStep = true;
    IsMouseVisible = true;
    Content.RootDirectory = "Content";
    _keyboard = new KeyboardManager(this);

```

```

        _mouse = new MouseManager(this);
        m_gameKeyboard = new Keyboard(obj);
        m_gameKeyboardState = new
SharpDX.DirectInput.KeyboardState();
        date = DateTime.Now;
        Timer timer = new Timer();
        timer.Interval = 10;
        timer.Tick += new EventHandler(tickTimer);
        timer.Start();
    }
    protected override void Initialize()
    {
        Window.Title = "GPUParticles";
        base.Initialize();
    }
    protected override void LoadContent()
    {
        _spriteBatch = new SpriteBatch(GraphicsDevice)
        {
            DisposeContent = true
        };
        _defaultFont = Content.Load<SpriteFont>("DefaultFont");

        _particles = new GPUParticlesHandler(GraphicsDevice,
Content.Load<SharpDX.Toolkit.Graphics.Effect>("ParticleRender"),
Content.Load<SharpDX.Toolkit.Graphics.Effect>("ParticleSolver"),
        Content.Load<Texture2D>("Particle16x16"),
        particlescount);
        _camera = new DevCamera();
        _camera.Init(
            GraphicsDevice.BackBuffer.Width,
            GraphicsDevice.BackBuffer.Height,
            60f,
            0.01f,
            1000f);
        base.LoadContent();
    }
    protected override void UnloadContent()
    {

```

```

        _camera.Dispose();
        base.UnloadContent();
    }
    protected override void Update(GameTime gameTime)
    {
        base.Update(gameTime);
        _camera.Update(gameTime, _mouse, _keyboard);
    }
    protected override void Draw(GameTime gameTime)
    {
        OpenHardwareMonitor.Hardware.Computer computerHardware = new
OpenHardwareMonitor.Hardware.Computer();
        computerHardware.GPUEnabled = true;
        computerHardware.Open();
        countsensors = computerHardware.Hardware[0].Sensors.Count();
        PerfomanceCounter.DoFrame(gameTime);
        GraphicsDevice.Clear(Color4.Black);
        _particles.Draw(gameTime, GraphicsDevice, _camera);
    {
        if (maxFPS < PerfomanceCounter.Fps)
        {
            maxFPS = PerfomanceCounter.Fps;
            if (i == 0)
            {
                minFPS = PerfomanceCounter.Fps;
                i++;
            }
        }
        if (minFPS > PerfomanceCounter.Fps)
        {
            minFPS = PerfomanceCounter.Fps;
        }
        _shText("FPS: " + PerfomanceCounter.Fps, Vector2.One,
Color.Aqua);

        for (int i = 0; i < countsensors; i++)
        {
            m_gameKeyboard.Acquire();

```

```

        m_gameKeyboard.GetCurrentState(ref
m_gameKeyboardState);

        if
(m_gameKeyboardState.PressedKeys.Contains(Key.Escape))
        {
            pasttest = new PastTest(maxGPUTemp, maxClockGPU,
maxMemoryClock,    maxShaderCoresClock,    maxFPS,    minFPS,    particlescount,
resolution, time);

            pasttest.Show();
            Exit();
            break;
        }
        name = computerHardware.Hardware[0].Sensors[i].Name;
        type                                     =
computerHardware.Hardware[0].Sensors[i].SensorType.ToString();
        if (name == "GPU Core")
        {
            if (type == "Temperature")
            {
                GPUTemp                                     =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
                maxGPUTemp                                     =
computerHardware.Hardware[0].Sensors[i].Max.ToString();
            }
            if (type == "Clock")
            {
                ClockGPU                                     =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
                maxClockGPU                                     =
computerHardware.Hardware[0].Sensors[i].Max.ToString();
            }
            if (type == "Load" || type == "Control")
            {
                LoadGPU                                     =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
            }
        }
    }
}

```

```

        if (type == "Clock")
        {
            if (name == "GPU Memory")
            {
                MemoryClock =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
                maxMemoryClock =
computerHardware.Hardware[0].Sensors[i].Max.ToString();
            }
        }
        if (name == "GPU Memory Controller")
        {
            LoadMemory =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
        }

        if (name == "GPU Shader")
        {
            ShaderCoresClock =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
            maxShaderCoresClock =
computerHardware.Hardware[0].Sensors[i].Max.ToString();
        }
        if (name == "GPU Fan" || name == "GPU")
        {
            if (type == "Fan")
            {
                PRM =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
            }
            if (type == "Control" || type == "Load")
            {
                LoadPRM =
computerHardware.Hardware[0].Sensors[i].Value.ToString();
            }
        }
        if (GPUTemp == "80")

```

```

        {
            Exit();
            MessageBox.Show("Critical temperature GPU 80
C!");

            pasttest = new PastTest(maxGPUTemp, maxClockGPU,
maxMemoryClock, maxShaderCoresClock, maxFPS, minFPS, particlescount,
resolution, time);

            pasttest.Show();
            break;
        }
        _shText("Timer: " + time, new Vector2(0, 40),
Color.Aqua);

        _shText("GPU Temperature: " + GPUTemp + " C", new
Vector2(0, 80), Color.Aqua);
        _shText("GPU Clock: " + ClockGPU + " MHz", new
Vector2(0, 120), Color.Aqua);
        _shText("GPU Load: " + LoadGPU + " %", new
Vector2(0, 160), Color.Aqua);
        _shText("Memory Clock: " + MemoryClock + " MHz", new
Vector2(0, 200), Color.Aqua);
        _shText("Memory Load: " + LoadMemory + " %", new
Vector2(0, 240), Color.Aqua);
        _shText("Shader Core Clock: " + ShaderCoresClock + "
MHz", new Vector2(0, 280), Color.Aqua);
        _shText("Fan: " + PRM + " PRM", new Vector2(0, 320),
Color.Aqua);

        _shText("Fan Load: " + LoadPRM + " %", new
Vector2(0, 360), Color.Aqua);
        _shText("Resolution: " + width + "x" + heiht, new
Vector2(0, 400), Color.Aqua);
        _shText("Esc - stop test ", new Vector2(0, 440),
Color.Aqua);
    }
}
base.Draw(gameTime);
}
private void tickTimer(object sender, EventArgs e)

```

```

    {
        long tick = DateTime.Now.Ticks - date.Ticks;
        DateTime stopwatch = new DateTime();
        stopwatch = stopwatch.AddTicks(tick);
        time = String.Format("{0:HH:mm:ss:ff}", stopwatch);
    }
    private void _shText(string text, Vector2 position, Color color)
    {
        DepthStencilState          defaultDepthState          =
GraphicsDevice.DepthStencilStates[0];
        _spriteBatch.Begin();

        _spriteBatch.DrawString(_defaultFont, text,
                                position,
                                color);
        _spriteBatch.End();
        GraphicsDevice.SetBlendState(null);
        GraphicsDevice.SetDepthStencilState(defaultDepthState);
    }
}
}

```

A.3 Код файла PastTest.cs

```

using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.OleDb;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Management;
namespace GPUParticles
{
    public partial class PastTest : Form
    {
        public static string connectionString =
"Provider=Microsoft.Jet.OLEDB.4.0; Data Source = Результатытестов.mdb";
        private OleDbConnection myConnection;
        public string str;
        public string vram, vram1, ram;
        public ulong vram2, ram1;
    }
}

```

```

public string driverversion;
public string maxTemperaturegpu;
public string maxClockgpu;
public string maxClockmemory;
public string maxClockshader;
public int MaxFPS;
public int MinFPS;
public int Particlecount;
public string Timetest;
public string Resolution;
public PastTest(string maxTemperatureGpu, string maxClockGPU,
string maxClockMemory, string maxClockShader, int maxFPS, int minFPS, int
particlecount, string resolution, string timetest)
{
    maxTemperaturegpu = maxTemperatureGpu;
    maxClockgpu = maxClockGPU;
    maxClockmemory = maxClockMemory;
    maxClockshader = maxClockShader;
    MaxFPS = maxFPS;
    MinFPS = minFPS;
    Particlecount = particlecount;
    Resolution = resolution;
    Timetest = timetest;
    InitializeComponent();
    myConnection = new OleDbConnection(connectionString);
    myConnection.Open();
    OpenHardwareMonitor.Hardware.Computer computerHardware = new
OpenHardwareMonitor.Hardware.Computer();
    computerHardware.GPUEnabled = true;
    computerHardware.Open();
    ManagementObjectSearcher objvide = new
ManagementObjectSearcher("select * from Win32_VideoController");
    foreach(ManagementObject obj in objvide.Get())
    {
        vram = obj["AdapterRAM"].ToString() + "|";
        vram1 = vram.Substring(0, vram.IndexOf("|"));
        int count = vram1.Length;
        if (count == 9)
        {
            vram2 = Convert.ToUInt32(vram1) / 1000000;
            vram = vram2.ToString() + "Mb";
            driverversion = obj["DriverVersion"].ToString();
            label19.Text += driverversion;
            label17.Text += vram;
            break;
        }
        if (count == 10)
        {
            vram2 = Convert.ToUInt32(vram1) / 1000000000;
            vram = vram2.ToString() + "Gb";
            driverversion = obj["DriverVersion"].ToString();
            label19.Text += driverversion;
            label17.Text += vram;
            break;
        }
        if (count == 11)
        {
            vram2 = Convert.ToUInt32(vram1) / 10000000000;
            vram = vram2.ToString() + "Gb";
            driverversion = obj["DriverVersion"].ToString();
            label19.Text += driverversion;
            label17.Text += vram;
            break;
        }
    }
}

```

```

    }
}
ManagementObjectSearcher objvide1 = new
ManagementObjectSearcher("select * from Win32_Processor");
foreach (ManagementObject obj in objvide1.Get())
{
    label15.Text += obj["Name"].ToString();
}
ManagementObjectSearcher objvide2 = new
ManagementObjectSearcher("select * from Win32_OperatingSystem");
foreach (ManagementObject obj in objvide2.Get())
{
    label18.Text += obj["Name"].ToString();
    label18.Text = label18.Text.Substring(0,
label18.Text.IndexOf("|"));
    ram = obj["TotalVisibleMemorySize"].ToString();

    int count = ram.Length;
    if (count == 6)
    {
        ram1 = Convert.ToUInt32(ram) / 1000;
        ram = ram1.ToString() + "Mb";
        label16.Text += ram;
        break;
    }
    if (count == 7)
    {
        ram1 = Convert.ToUInt32(ram) / 1000000;
        ram = ram1.ToString() + "Gb";
        label16.Text += ram;
        break;
    }
    if (count == 8)
    {
        ram1 = Convert.ToUInt32(ram) / 1000000;
        ram = ram1.ToString() + "Gb";
        label16.Text += ram;
        break;
    }
    if (count == 9)
    {
        ram1 = Convert.ToUInt32(ram) / 1000000;
        ram = ram1.ToString() + "Gb";
        label16.Text += ram;
        break;
    }
}
label10.Text += computerHardware.Hardware[0].Name;
label6.Text += maxTemperatureGpu + " C";
label4.Text += minFPS.ToString();
label3.Text += maxFPS.ToString();
label11.Text += timetest;
label8.Text += resolution;
label9.Text += particlecount.ToString();
label7.Text += maxClockGPU + " MHz";
label12.Text += maxClockMemory + " MHz";
label13.Text += maxClockShader + " MHz";
int directxMajorVersion = 0;
var OSVersion = Environment.OSVersion;
// if Windows Vista or later
if (OSVersion.Version.Major >= 6)
{
    // if Windows 7 or later

```

```

        if (OSVersion.Version.Major > 6 ||
OSVersion.Version.Minor >= 1)
        {
            directxMajorVersion = 11;
        }
        // if Windows Vista
        else
        {
            directxMajorVersion = 10;
        }
    }
    // if Windows XP or earlier.
    else
    {
        using (RegistryKey key =
Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\DirectX"))
        {
            string versionStr = key.GetValue("Version") as
string;

            if (!string.IsNullOrEmpty(versionStr))
            {
                var versionComponents = versionStr.Split('.');
                if (versionComponents.Length > 1)
                {
                    int directXLevel;
                    if (int.TryParse(versionComponents[1], out
directXLevel))

                        {
                            directxMajorVersion = directXLevel;
                        }
                }
            }
        }
        label5.Text += directxMajorVersion.ToString();
    }
private void label8_Click(object sender, EventArgs e)
{
}
private void PastTest_Load(object sender, EventArgs e)
{
    button1.Enabled = false;
    string query = "SELECT Видеокарта FROM Результаты ORDER BY

Код";

    OleDbCommand command = new OleDbCommand(query, myConnection);
    OleDbDataReader reader = command.ExecuteReader();
    comboBox1.Items.Clear();
    while (reader.Read())
    {
        string podstroka = "Videocard:";
        string videocard = reader[0].ToString();
        videocard = videocard.Replace(podstroka, "");
        comboBox1.Items.Add(videocard);
    }
    reader.Close();
}
private void label10_Click(object sender, EventArgs e)
{
}
private void label6_Click(object sender, EventArgs e)
{
}

```

```

private void label9_Click(object sender, EventArgs e)
{
}
private void label7_Click(object sender, EventArgs e)
{
}
private void label12_Click(object sender, EventArgs e)
{
}
private void label13_Click(object sender, EventArgs e)
{
}
private void label4_Click(object sender, EventArgs e)
{
}
private void label3_Click(object sender, EventArgs e)
{
}
private void label11_Click(object sender, EventArgs e)
{
}
private void label5_Click(object sender, EventArgs e)
{
}
private void button4_Click(object sender, EventArgs e)
{
    string query = "INSERT INTO Результаты (Видеокарта,
МаксимТемпература,
Времятеста,
МаксФПС,
МинФПС,
DirectX,
Разрешение,
Колвочастиц,
ЧастотаGPU,
Частотапамяти,
Частоташейдеров,
ЦП,
ОЗУ,
ВРАМ,
ОСВерсия,
Driver) VALUES ('" + label10.Text + "','" + label6.Text + "','" +
label11.Text + "','" + label3.Text + "','" + label4.Text + "','" + label5.Text +
'"', "'" + label8.Text + "','" + label9.Text + "','" + label7.Text + "','" +
label12.Text + "','" + label13.Text + "','" + label15.Text + "','" +
label16.Text + "','" + label17.Text + "','" + label18.Text + "','" +
label19.Text + "')";
    OleDbCommand command = new OleDbCommand(query, myConnection);
    MessageBox.Show("Info test add to database successfully!");
    command.ExecuteNonQuery();
    button4.Enabled = false;
}
private void comboBox1_SelectedIndexChanged(object sender,
EventArgs e)
{
    button4.Enabled = false;
    button1.Enabled = true;
    int index = comboBox1.SelectedIndex;
    index = index + 1;
    string query = "SELECT Видеокарта, МаксимТемпература,
Времятеста, МаксФПС, МинФПС, DirectX, Разрешение, Колвочастиц, ЧастотаGPU,
Частотапамяти, Частоташейдеров, ЦП, ОЗУ, ВРАМ, ОСВерсия, Driver FROM Результаты
WHERE Код =" + index.ToString();
    OleDbCommand command = new OleDbCommand(query, myConnection);
    OleDbDataReader reader = command.ExecuteReader();
    while (reader.Read())
    {
        label10.Text = reader.GetString(0);
        label6.Text = reader[1].ToString();
        label11.Text = reader[2].ToString();
        label3.Text = reader[3].ToString();
        label4.Text = reader[4].ToString();
        label5.Text = reader[5].ToString();
        label8.Text = reader[6].ToString();
        label9.Text = reader[7].ToString();
    }
}

```

```

        label17.Text = reader[8].ToString();
        label12.Text = reader[9].ToString();
        label13.Text = reader[10].ToString();
        label15.Text = reader[11].ToString();
        label16.Text = reader[12].ToString();
        label17.Text = reader[13].ToString();
        label18.Text = reader[14].ToString();
        label19.Text = reader[15].ToString();
    }
    reader.Close();
}
private void label17_Click(object sender, EventArgs e)
{
}
private void label15_Click(object sender, EventArgs e)
{
}
private void button1_Click(object sender, EventArgs e)
{
    OpenHardwareMonitor.Hardware.Computer computerHardware = new
OpenHardwareMonitor.Hardware.Computer();
    computerHardware.GPUEnabled = true;
    computerHardware.Open();
    label10.Text = "Videocard: " +
computerHardware.Hardware[0].Name;
    label6.Text = "Max temperature:" + maxTemperaturegpu + " C";
    label4.Text = "Min FPS: " + MinFPS.ToString();
    label3.Text = "Max FPS: " + MaxFPS.ToString();
    label11.Text = "Time test: " + Timetest;
    label8.Text = "Resulotion: " + Resolution;
    label9.Text = "Particles count: " + Particlecount;
    label7.Text = "Max clock GPU: " + maxClockgpu + " MHz";
    label12.Text = "Max clock memory: " + maxClockmemory +
"MHz";

    label13.Text = "Max clock shader: " + maxClockshader + "MHZ";
    ManagementObjectSearcher objvide = new
ManagementObjectSearcher("select * from Win32_VideoController");
    foreach (ManagementObject obj in objvide.Get())
    {
        vram = obj["AdapterRAM"].ToString() + "|";
        vram1 = vram.Substring(0, vram.IndexOf("|"));
        int count = vram1.Length;
        if (count == 9)
        {
            vram2 = Convert.ToUInt32(vram1) / 1000000;
            vram = vram2.ToString() + "Mb";
            driverversion = obj["DriverVersion"].ToString();
            label19.Text = "Driver videocard version: " +
driverversion;

            label17.Text = "Video memory: " + vram;
            break;
        }
        if (count == 10)
        {
            vram2 = Convert.ToUInt32(vram1) / 1000000000;
            vram = vram2.ToString() + "Gb";
            driverversion = obj["DriverVersion"].ToString();
            label19.Text = "Driver videocard version: " +
driverversion;

            label17.Text = "Video memory: " + vram;
            break;
        }
        if (count == 11)

```

```

        {
            vram2 = Convert.ToUInt32(vram1) / 1000000000;
            vram = vram2.ToString() + "Gb";
            driverversion = obj["DriverVersion"].ToString();
            label19.Text = "Driver videocard version: " +
driverversion;

            label17.Text = "Video memory: " + vram;
            break;
        }
    }
    ManagementObjectSearcher objvide1 = new
ManagementObjectSearcher("select * from Win32_Processor");
    foreach (ManagementObject obj in objvide1.Get())
    {
        label15.Text = "CPU: " + obj["Name"].ToString();
    }
    ManagementObjectSearcher objvide2 = new
ManagementObjectSearcher("select * from Win32_OperatingSystem");
    foreach (ManagementObject obj in objvide2.Get())
    {
        label18.Text = "OS Version: " + obj["Name"].ToString();
        label18.Text = label18.Text.Substring(0,
label18.Text.IndexOf("|"));
        ram = obj["TotalVisibleMemorySize"].ToString();
        int count = ram.Length;
        if (count == 6)
        {
            ram1 = Convert.ToUInt32(ram) / 1000;
            ram = ram1.ToString() + "Mb";
            label16.Text = "RAM: " + ram;
            break;
        }
        if (count == 7)
        {
            ram1 = Convert.ToUInt32(ram) / 1000000;
            ram = ram1.ToString() + "Gb";
            label16.Text = "RAM: " + ram;
            break;
        }
        if (count == 8)
        {
            ram1 = Convert.ToUInt32(ram) / 1000000;
            ram = ram1.ToString() + "Gb";
            label16.Text = "RAM: " + ram;
            break;
        }
        if (count == 9)
        {
            ram1 = Convert.ToUInt32(ram) / 1000000;
            ram = ram1.ToString() + "Gb";
            label16.Text = "RAM: " + ram;
            break;
        }
    }
    button1.Enabled = false;
    button4.Enabled = true;
}
private void label18_Click(object sender, EventArgs e)
{
}
private void label16_Click(object sender, EventArgs e)
{
}

```

```

    }
    private void label19_Click(object sender, EventArgs e)
    {
    }
}

```

A.4 Код файла UserForm.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using GPUParticles.Particles;
using OpenTK;
using OpenTK.Graphics;
using OpenTK.Graphics.OpenGL;
using OpenTK.Input;
using System.Xml;
namespace GPUParticles
{
    public partial class UserForm : Form
    {
        public string videocard;
        public int particlecount;
        public bool fullscreen;
        public int weidth, height;
        public Logic program;
        public UserForm()
        {
            InitializeComponent();
            DiscreteVideocard obj = new DiscreteVideocard();
            videocard = obj.videocardmethod();
        }

        private void UserForm_Load(object sender, EventArgs e)
        {
            OpenHardwareMonitor.Hardware.Computer computerHardware = new
OpenHardwareMonitor.Hardware.Computer();
            computerHardware.GPUEnabled = true;
            computerHardware.Open();

            int countvideocard = computerHardware.Hardware.Count();
            if (countvideocard == 1)
            {
                if (videocard.Contains("/PCIe/SSE2"))
                {
                    label5.Text = computerHardware.Hardware[0].Name;
                }
            }
        }
    }
}

```

```

        else
        {
            MessageBox.Show("Videocard not found!");
            Application.Exit();
        }
    }
    else
    {
        label5.Text = computerHardware.Hardware[0].Name;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    if (comboBox1.SelectedIndex > -1)
    {
        if (comboBox1.SelectedIndex == 0)
        {
            width = 800;
            height = 600;
        }
        if (comboBox1.SelectedIndex == 1)
        {
            width = 1024;
            height = 768;
        }
        if (comboBox1.SelectedIndex == 2)
        {
            width = 1280;
            height = 720;
        }
        if (comboBox1.SelectedIndex == 3)
        {
            width = 1600;
            height = 900;
        }
        if (comboBox1.SelectedIndex == 4)
        {
            width = 1920;
            height = 1080;
        }
        if (comboBox1.SelectedIndex == 5)
        {
            width = 2048;
            height = 1080;
        }
        if (comboBox1.SelectedIndex == 6)
        {
            width = 2560;
            height = 1440;
        }
        if (comboBox1.SelectedIndex == 7)
        {
            width = 4096;
            height = 2160;
        }
        if (comboBox1.SelectedIndex == 8)
    }
}

```

```

        {
            weidth = Convert.ToInt32(textBox2.Text);
            height = Convert.ToInt32(textBox3.Text);
        }
        particlecount = Convert.ToInt32(comboBox2.Text);
        if (particlecount == 0)
        {
            MessageBox.Show("Set the correct number of particles!");
        }
        else
        {
            using (program = new Logic(particlecount,
checkbox1.Checked, weidth, height))
                program.Run();
        }
    }
    else
    {
        MessageBox.Show("Select a screen resolution for the test!");
    }
}
private void button2_Click(object sender, EventArgs e)
{
    weidth = 2048;
    height = 1080;
    particlecount = Convert.ToInt32(comboBox2.Text);
    if (particlecount == 0)
    {
        MessageBox.Show("Set the correct number of particles!");
    }
    else
    {
        using (program = new Logic(particlecount, checkbox1.Checked,
weidth, height))
            program.Run();
    }
}
private void button3_Click(object sender, EventArgs e)
{
    weidth = 1920;
    height = 1080;
    particlecount = Convert.ToInt32(comboBox2.Text);
    if (particlecount == 0)
    {
        MessageBox.Show("Set the correct number of particles!");
    }
    else
    {
        using (program = new Logic(particlecount, checkbox1.Checked,
weidth, height))
            program.Run();
    }
}
private void button4_Click(object sender, EventArgs e)
{
    weidth = 1280;

```

```

        height = 720;
        particlecount = Convert.ToInt32(comboBox2.Text);
        if (particlecount == 0)
        {
            MessageBox.Show("Set the correct number of particles!");
        }
        else
        {
            using (program = new Logic(particlecount, checkBox1.Checked,
weidth, height))
                program.Run();
        }
    }
    private void button6_Click(object sender, EventArgs e)
    {
        weidth = 4196;
        height = 2160;
        particlecount = Convert.ToInt32(comboBox2.Text);
        if (particlecount == 0)
        {
            MessageBox.Show("Set the correct number of particles!");
        }
        else
        {
            using (program = new Logic(particlecount, checkBox1.Checked,
weidth, height))
            {
                program.Run();
            }
        }
    }
    private void textBox2_TextChanged(object sender, EventArgs e)
    {
    }
    private void textBox3_TextChanged(object sender, EventArgs e)
    {
    }
    private void comboBox2_SelectedIndexChanged(object sender, EventArgs e)
    {
    }
    private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (comboBox1.SelectedIndex == 8)
        {
            textBox2.Enabled = true;
            textBox3.Enabled = true;
        }
        else
        {
            textBox2.Enabled = false;
            textBox3.Enabled = false;
        }
    }
}
}
}

```

ДОДАТОК Б
Копії слайдів презентації

ДИПЛОМНА РОБОТА

РОЗРОБКА ПРОГРАМИ ТЕСТУВАННЯ ВІДЕОАДАПТЕРА

Розробив: ст. групи КНТ-217
Керівник: доцент, к.т.н.

Майборода С.І.
Сердюк С.М.

Рисунок Б.1 – Слайд №1

Мета роботи

розробка програмного продукту для
тестування характеристик відеоадаптеру
за допомогою стрес-тесту

Об'єкт дослідження — процес
тестування відеоадаптеру

Предмет дослідження — методи
тестування відеоадаптеру



Задачі, що вирішуються

- огляд типів тестів відеоадаптеру;
- аналіз існуючих аналогів;
- розробка технічного завдання;
- огляд необхідних для розробки застосунку технологій;
- програмна реалізація складових застосунку;
- розробка необхідної документації.



НУ “ЗП”, Кафедра програмних засобів

Рисунок Б.3 – Слайд №3

3

Результати аналізу альтернативних програмних продуктів

Функції	AIDA64	MSI Afterburner	GeForce Experience	FurMark	3DMark
Стрес-тести	+	+	+	+	+
Вимірювання температури GPU	+	+	+	+	–
Перевірка стабільності відеокарти	+	+	+	+	–
Збереження результатів у файл	+	+	+	+	+
Визначення частоти кадрів в 3D-застосунках	+	+	+	+	–
Моніторинг швидкості обертання кулера	+	+	+	+	+
Технічні характеристики відеоадаптера	+	+	+	+	+
Вивід детальної інформації в процесі діагностики	+	+	–	–	–

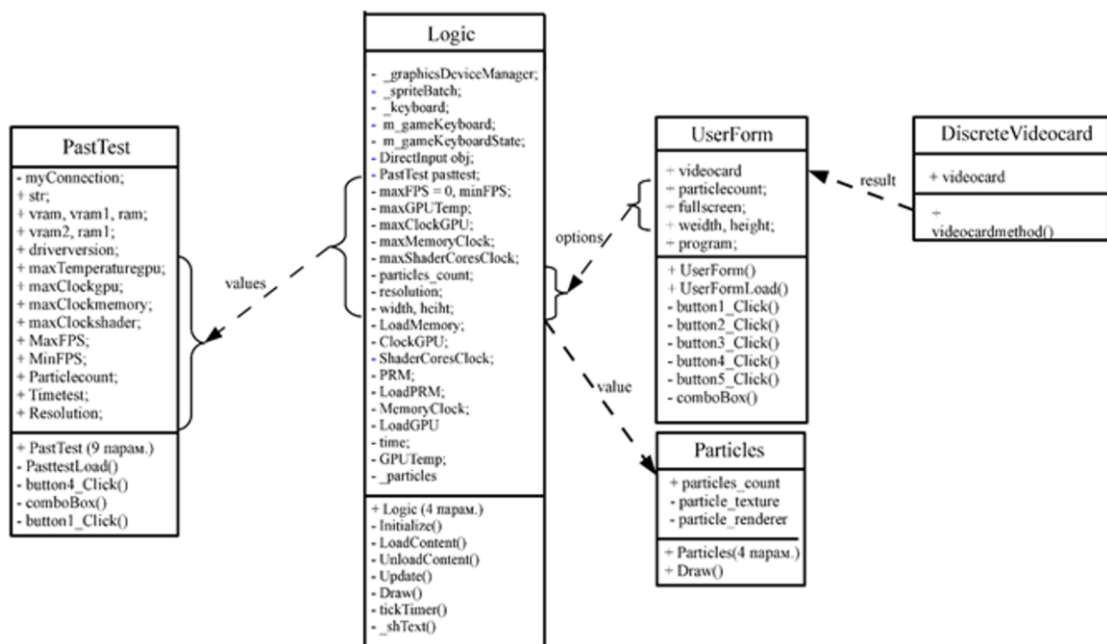


НУ “ЗП”, Кафедра програмних засобів

Рисунок Б.4 – Слайд №4

4

Діаграма класів програми «GPU Test»



НУ "ЗП", Кафедра програмних засобів

5

Рисунок Б.5 – Слайд №5

Приклад класу Particles

```
public class Particles
```

```

    public Particles(GraphicsDevice
graphics, Effect particlesRender,
Texture2D particleTexture, int
partcount)

```

```

    public void Draw(GraphicsDevice
graphics)

```

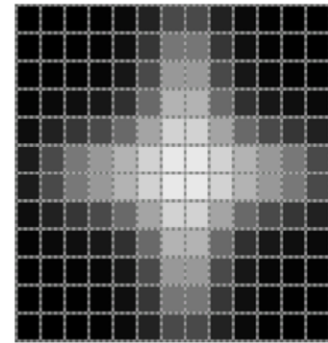
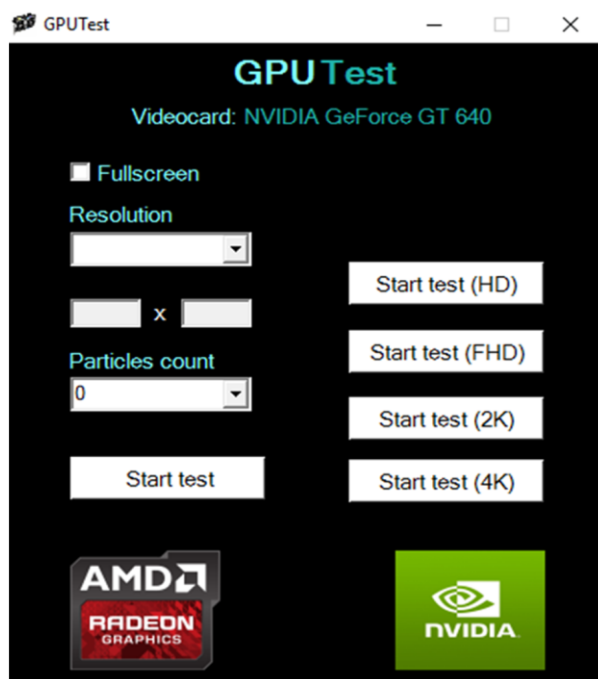


НУ "ЗП", Кафедра програмних засобів

6

Рисунок Б.6 – Слайд №6

Головна користувацька форма програми «GPU Test»



Текстура частинки для стрес-тесту

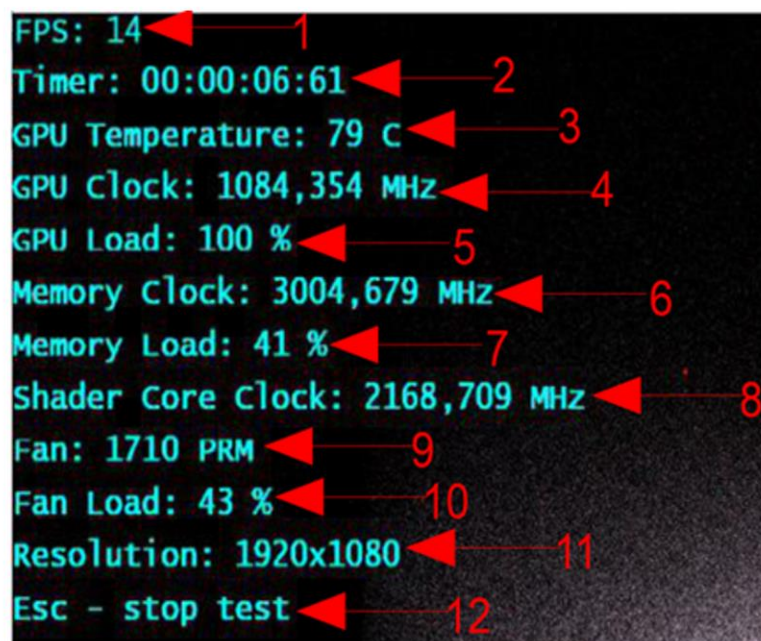


НУ "ЗП", Кафедра програмних засобів

7

Рисунок Б.7 – Слайд №7

Інформація, що виводиться у вікні під час стрес-тесту

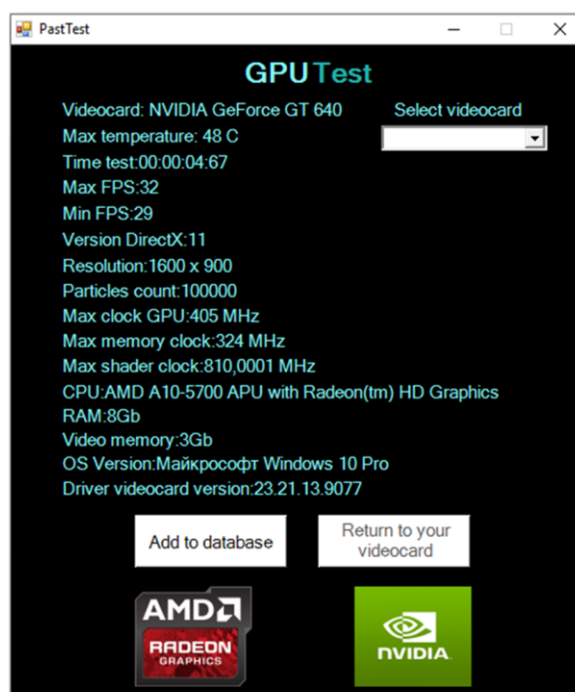


НУ "ЗП", Кафедра програмних засобів

8

Рисунок Б.8 – Слайд №8

Приклад результатів тестування (друга користувацька форма)



НУ "ЗП", Кафедра програмних засобів

9

Рисунок Б.9 – Слайд №9

Висновки

- проведено огляд типів існуючих тестів відеоадаптеру;
- проведено огляд існуючих програм-аналогів;
- розроблено технічне завдання;
- проведено огляд необхідних для розробки застосунку технологій;
- програмно реалізовано усі складові частини застосунку;
- розроблено необхідну документацію.



НУ "ЗП", Кафедра програмних засобів

10

Рисунок Б.10 – Слайд №10

**Доклад закінчено,
дякую за увагу!**



НУ "ЗП", Кафедра програмних засобів

Рисунок Б.11 – Слайд №11