Міністерство освіти і науки України

Державний університет «Одеська політехніка»

Інститут штучного інтелекту та робототехніки

Кафедра комп’ютерних систем

Канюс Володимир Олегович,

студент групи УК-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Оптимізація програмно-апаратних ресурсів ігрового движка для зменшення навантаження їх компонент.

Галузь знань: 12 Інформаційні технології.

Спеціальність: 123 Комп’ютерна інженерія

Керівник:

Ситніков Валерій Степанович,

проф. кафедри

Одеса – 2021

Міністерство освіти і науки України
Державний університет «Одеська політехніка»

Інститут штучного інтелекту та робототехніки
Кафедра комп'ютерних систем

Рівень вищої освіти другий (магістерський)_____
Спеціальність__ 123 Комп'ютерна інженерія_____

(шифр і назва)
Спеціалізація/ освітня програма ___ Спеціалізовані комп'ютерні системи_____

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____*Ситніков В. С.*
"____" _____2021 року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

_____*Канюс Володимир Олегович*_____
(прізвище, ім'я, по батькові)

1. Тема роботи_____*Оптимізація програмно-апаратних ресурсів ігрового движка для зменшення навантаження їх компонент.*_____

Керівник роботи____ *Ситніков В.С.,проф.кафедри* ___,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджено наказом ректора ОП від "___"_____2021 року №___

2. Зміст роботи. *Розробка комплексного метода для оптимізації ігрового двигуна. Аналіз існуючих двигунів та їх особливості між собою, вибір ігрового двигуна для подальшого поглибленого аналізу та знаходження критичних моментів, які впливають на потужність. Аналіз алгоритмів оптимізації для двигуна та алгоритмів оптимізації для програмного коду.*
Ключові слова: Пляшкова шийка, оптимізація, тестування, програмний код, алгоритми оптимізації, апаратні ресурси .

3. Перелік ілюстративного матеріалу.

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1,2,3 Нормоконтроль	*Стрельцов О.В.*		

5. Дата видачі завдання 15 квітня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд існуючих ігрових двигунів*	*15 травня 2021 р.*	
2.	*Огляд алгоритмів оптимізації за допомогою двигуна та коду*	*15 травня 2021 р.*	
3.	*Розробка комплексного метода для оптимізації апаратних ресурсів та зменшення споживання оперативної пам'яті*	*15 червня 2021 р.*	
4.	*Збір інформації та аналіз комп'ютерів та їх комплектуючих*	*15 липня 2021 р.*	
5.	*Тестування 3 конфігурацій на мінімальних графічних налаштуваннях*	*15 вересня 2021 р.*	
6.	*Тестування 3 конфігурацій на середніх графічних налаштуваннях*	*15 жовтня 2021 р.*	
7.	*Тестування 3 конфігурацій на максимальних графічних налаштуваннях*	*15 листопада 2021 р.*	
8.	*Аналіз усіх тестів та підведення підсумків*	*28 листопада 2021 р.*	

Здобувач
магістерського ступеня *Канюс В.О.*

Керівник роботи *Ситніков В.С.*

ЗМІСТ

ВСТУП

Актуальність. Перспективи розвитку оптимізації комп’ютерних ігор в першу чергу залежить від програмного коду та розвитку апаратних ресурсів. В першу чергу продуктивність залежить від логіки або алгоритмів, які прописали у коді. В другу чергу продуктивність гри або будь якої програми залежить від апаратних ресурсів системи на яку встановили дане ПО(Програмне забезпечення).

Оптимізація потрібна у будь-якій сфері, але в цій роботі ми проаналізуємо сферу розробки ігор та використання ігрових двигунів для покращення продуктивності на більшості комп’ютерах.

Коли людина вперше чую про оптимізацію, то вона вважає, що це дуже складних механізм у якому задіяно дуже багато працівників. Справді так воно і є, оптимізація буває(для механік гри, для візуалізації, для більшої частоти кадрів і так далі). Над оптимізацією працюють всі працівники, які працювали при створені двигуна або гри.

Оптимізація – це дуже складний процес в який поділяють на 3 основні етапи(Аналіз проблеми, знаходження кращого рішення, тестування).

Перший етап – це мабуть найголовніший етап, потрібно виявити проблему. Це проблема із-за недостачі апаратних ресурсів, проблема у самому коді, проблем двигуна та неправлене налаштування самого двигуна. Якщо проблема апаратна, тоді потрібно проаналізувати навантаження на окрему частину систему(Процесор, відео карту, ОЗУ(Оперативне

запам’ятовуючий пристрій), ЗУ(Запам’ятовуючий пристрій)), побудувати графіки навантаження.

Проблема у самому коді при створенні коду для двигуна чи гри. Основними проблемами в цій сфері бувають(недотримання принців програмування, побудова неправильного дизайну коду, втрата пам’яті або втрата перемінної яка займає багато пам’яті та інші).

Проблеми двигуна. Тут вже складніше, якщо двигун був написаний працівниками іншої компанії або його взяли у використання, то немає повної технічної документації. Якщо двигун був написаний командую, яка розробляла проект, то може бути проблема, що на двигун потратили менше часу для того щоб встигнути у термін випуску ПО. Бувають ще проблеми за налаштуванням самого двигуна під конкретне ПО.

Другий етап. Після аналізу, приймається рішення щодо оптимізації програми або гри на одному із трьох основних рівнів. На першому рівні апаратному є два варіанті(Підвищення мінімальних системних вимог або покращення ситуації за допомогою коду та двигуна). Другий рівень оптимізація коду, знаходження smell code(дурний запах коду), знаходження кращих алгоритмів, які дію швидше та мають найменшу складність за нотацією Big O, групування однакових перемінних за значенням в єдиний клас, аналізування перемінних, які за час життя програми не змінюються та винесення їх у окремий клас та маркіруванням, як константа і так далі. Третій рівень знаходження проблем у самому двигуні та спроби покращити його, не злам інші методи, спробувати інтегрувати алгоритми для покращення швидкості роботи двигуна та зменшення складнощі принципи вираховування фізики.

Третій етап. Тестування методів оптимізації на різних збірках. Частіше за все беруть 10 різних конфігурацій системи, перевіряють швидкість і навантаження до оптимізації та після оптимізації. Будуть графіки та аналізують отриманий результат. Якщо на 8 із 10 систем оптимізація допомогла зменшити навантаження та прискорити процеси обчислення, тоді цей метод інтегрують у основну частину програми, якщо результат був 2 із 10, то аналізують і вирішують використати час на покращення цього метода, чи взяти іншій, та протестувати.

Мета та завдання дослідження. Метою КРМ є оптимізація програмно-апаратних ресурсів ігрового движка для зменшення навантаження їх компонентів, для покращення частоти кадрів та зменшення температури на головні частини комп’ютера.

Для досягнення поставленої мети в роботі вирішені наступні завдання;

- виконано аналіз ігрового двигуна Unity3d;
- досліджено алгоритми та методи оптимізації ігрових двигунів;
- аналіз навантаження на комплектуючі;
- створення комплексного метода для оптимізації;
- створення систем під тестування та первинне налаштування;
- тестування систем до оптимізації та збір даних;
- тестування систем після оптимізації та збір даних;
- аналіз отриманих даних;

Об’єктом дослідження процес оптимізації програмно апаратних ресурсів.

Предмет дослідження – комплексний метод оптимізації на прикладі комп’ютерної гри Escape from Tarkov.

Методи дослідження. З визначенням об’єкту та предмету дослідження випливають і методи за допомогою, яких можливо проаналізувати навантаження на комплектуючи та програми. Ігровий двигун буде проаналізовано з іншими двигунами та буде порівняння, буде пояснення, цьому саме розробники Escape from Tarkov саме вибрали цей двигун, а не інший. Буде аналіз самої гри, та які проблеми вона має саме зараз і як їх можливо буде вирішити у наступних оновленнях. За допомогою методів для аналізу та програм для перевірки навантаження можливо буде пообдувати графіки для подальшого аналізу оптимізації.

Створення та аналіз тестових конфігурацій для перевірки методів оптимізації на різних комплектуючих та побудова графіків для аналізу.

Наукова новизна.

- запропоновано використовувати алгоритми для оптимізації програмно-апаратних ресурсів через ігровий двигун та використання шаблонів для програмного коду.
- виконано створення комплексного методу для зменшення навантаження та об’єму використання оперативної пам’яті.
- більш детальний розвиток в сфері оптимізації ігрового двигуна Unity 3D для гри Escape from Tarkov.

Практичне значення. Зменшення навантаження для більш комфортного використання ігрового двигуна та зменшення вартості конфігурації.

Апробація результатів магістерської роботи. Направлені тези доповіді для участі в наукових семінарах кафедри комп’ютерних систем інституту штучного інтелекту та робототехніки впродовж 2021року.

1. АНАЛІЗ ІГРОВИХ ДВИГУНІВ

1.1. Ігровий двигун на основі Unity 3d

Ігровий двигун (англ. *Game engine*) — програмний рушій, центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну сторону, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її внутрішньої структури. Важливим значенням рушія є можливість створення багатоплатформових ігор (сьогодні найчастіше одночасно для ПК, PS4 та Xbox One).

Основну функціональність гри зазвичай забезпечує її рушій, до якого входить рушій рендерингу («візуалізатор»), фізичний рушій, звук, система скриптів, анімація, ігровий штучний інтелект, мережевий код, керування пам'яттю, багатонитевість і граф сцени. Часто на процесі розробки можна заощадити шляхом повторного використання одного рушія гри для створення декількох різних ігор. На рисунку 1.1 показано декілька популярних ігрових двигунів.

Рисунок 1.1 – Ігрові двигуни

На додаток до багаторазово використовуваних програмних компонентів, ігрові рушії надають набір візуальних інструментів для розробки. Ці інструменти зазвичай складають інтегроване середовище розробки для спрощеної, швидкої розробки ігор на зразок потокового виробництва. Такі рушії іноді називають «ігровим підпрограмним забезпеченням» *(скор. ППЗ; англ. middleware)*, тому що, з погляду бізнесу, вони надають гнучку й багаторазово використовувану програмну платформу з усією необхідною функціональністю для розробки грального застосунку, скорочуючи витрати, складність і час розробки — усі критичні фактори в сильно конкурентній індустрії відеоігор.

Як і інші рішення програмного забезпечення, ігрові рушії зазвичай платформонезалежні й дозволяють деякій грі запускатися на різних платформах, включаючи гральні консолі й персональні комп'ютери, з деякими внесеними у початковий код змінами (або взагалі без них). Часто гральне ППЗ має компонентну архітектуру, що дозволяє замінювати або розширювати деякі системи рушія

спеціалізованішими (і часто дорожчими) компонентами ППЗ, наприклад Havok — для фізики, FMOD — для звуку або SpeedTree — для рендерингу. Деякі рушії, такі як RenderWare, проектують як набір слабо зв'язаних компонентів ППЗ, які можуть вибірково комбінуватися для створення власного рушія, замість традиційнішого підходу розширення або налаштовування гнучкого інтегрованого рішення. Проте розширюваність досягнута й залишається високопріоритетною в ігрових рушіях через широкі можливості їхнього застосування. Попри специфічність назви, ігрові рушії часто використовуються в інших типах інтерактивних застосунків, що вимагають графіку в реальному часі, таких як рекламні демовідео, архітектурні візуалізації, навчальні симулятори й середовища моделювання.

Деякі ігрові рушії надають тільки можливості 3D рендерингу в реальному часі замість усієї функціональності, необхідної іграм. Ці рушії довіряють розробнику гри реалізацію іншої функціональності або її складання на основі інших грального компонентів ППЗ. Такі типи рушіїв зазвичай відносять до «графічних рушіїв», «рушіїв рендерингу» або «3D рушіїв» замість змістовнішого терміну «ігровий рушій». Однак ця термінологія використовується суперечливо: багато повнофункціональних грального 3D рушіїв згадані просто як «3D рушії». Деякі приклади графічних рушіїв: RealmForge, Ogre 3D, Power Render, Crystal Space і Genesis3D. Сучасні ігрові або графічні рушії зазвичай надають граф сцени — об'єктно-орієнтовані представлення 3D світу гри, що часто спрощує ігровий дизайн і може використовуватися для ефективнішого рендерингу величезних віртуальних світів.

Найчастіше 3D рушії або системи рендерингу в ігрових рушіях побудовані на графічному API, такому як Direct3D або OpenGL, що забезпечує програмну абстракцію GPU або відеокарти. Низькорівневі бібліотеки, наприклад, DirectX, SDL і OpenAL, також використовуються в іграх, тому що забезпечують апаратно-незалежний доступ до іншого апаратного забезпечення комп'ютера, такого як пристрої введення (миша, клавіатура й джойстик), мережеві й звукові карти. До появи апаратно прискорюваної 3D графіки використовувалися програмні візуалізатори. Програмний рендеринг усе ще використовується в деяких інструментах моделювання для рендеринга зображень, для яких візуальна достовірність важливіша за продуктивність (кількість кадрів за секунду) або коли апаратне забезпечення комп'ютера не задовольняє вимогам, наприклад, не підтримує шейдери.[1]

Найкращі ігрові движки

На відміну від ігор як таких, двигуни не змінюють один одного так часто. Тож деякі з них використовуються вже майже десять років. Ось вам кілька прикладів найбільш іменитих систем: RAGE, CryEngine, Naughty Dog Game Engine, Avalanche Engine, IW Engine, Anvil Engine, EGO Engine, Geo-Mod Engine, The Dead Engine і, звичайно ж, Unreal Engine.

4A Engine

Один з найпопулярніших двигунів для шутерів і екшенів, написаний українськими розробниками - вихідцями з GSC Game World. Ця платформа використовується тільки для внутрішніх потреб компанії і не доступна для інших розробників ні на якій основі (ні платно, ні безкоштовно). З технічної точки зору вона є

покращеним X-Ray з доопрацьованим PhysX, тесселяцією для поліпшення графіки, а також повною руйнівністю об'єктів.

Двигун забезпечує тривимірне позиціонування звуку, динамічне освітлення, безліч умов бою, відмінні умови для скриптування. Крім того, в ньому є система аналізу топології II, можливість наділити персонажів зором, слухом та іншими почуттями, а також задати їхню групову поведінку. Загалом, на основі цього движок можна писати справді складні шутери та екшени. На рисунку 1.2 показано логотип 4A engine.

Рисунок 1.2 – Логотип движка 4A engine

CryEngine 4

Німецька студія Crytek продовжує оновлювати та осучаснювати своє ігрове ядро, при цьому, як і колишні версії, CryEngine 4 поширюється практично безкоштовно – з мінімальною оплатою. При цьому за своїми можливостями він не урізаний, і ви можете чудово бачити це в іграх серії FarCry.

CryEngine 4 забезпечує відображення величезних безшовних локацій, інверсної кінематики транспорту та персонажів, відмінну імітацію нетвердих об'єктів, параметри штучного інтелекту, що настроюються, звучання формату 5.1, а також безліч інших переваг. Загалом, це відмінна платформа і для досвідчених розробників, і для гравців-початківців. На рисунку 1.3 показано інтерфейс CryEngine 4 та основний його вигляд під час роботи.

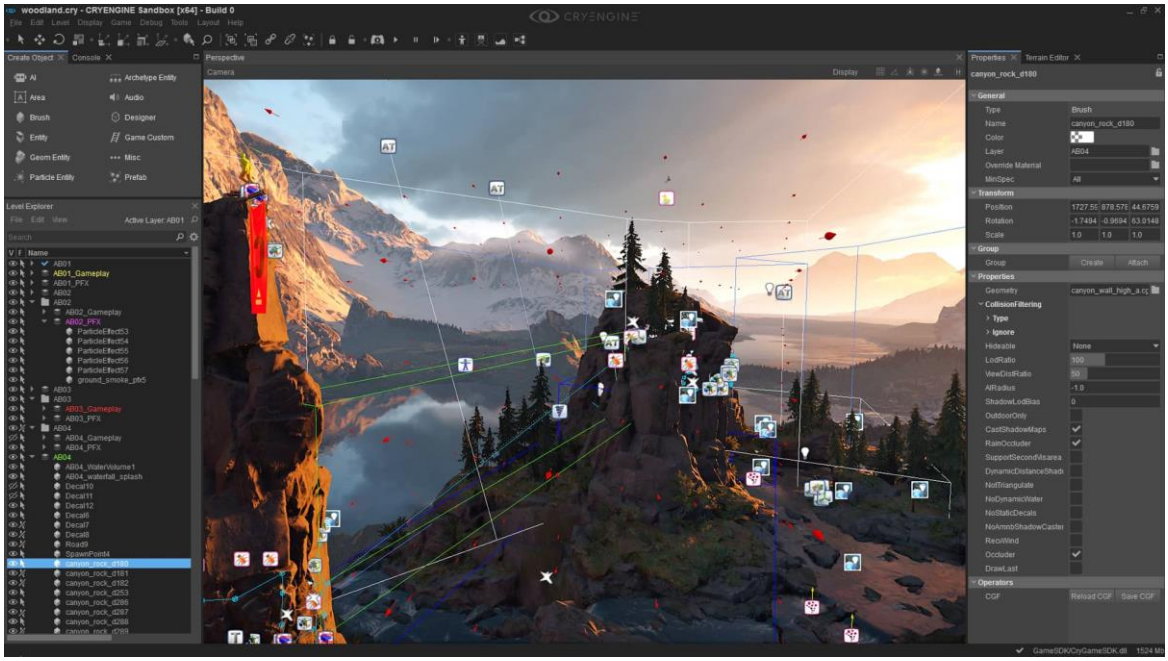

Рисунок 1.3 – Інтерфейс CryEngine 4

id Tech

Ще одна легенда, що існує вже в сьомій версії і протягом усього існування, що поширюється на повністю безкоштовній основі. При цьому не варто думати, що двигун урізаний або простий - на його основі створені такі хіти, як Wolfenstein, Quake, Rage, Doom. Тут якісно зроблено відображення текстур, є окремий потік для обробки кожної складової двигуна і навіть є півтіні для

затінення ділянок у кадрі. На рисунку 1.4 показано логотип Id tech engine

Рисунок 1.4 – Логотип Id tech engine

Frostbite

Це двигун, про який знають усі фанати ігор від Electronic Arts. Він використовується у багатьох іграх цієї студії, включаючи Battlefield, FIFA, DragonAge, PayBack. Простіше кажучи, це платформа-універсал, з урахуванням якої написані екшени, РПГ, гонки, спортивні симулятори тощо.

Другий дивовижний момент полягає в тому, що у движка практично немає недоліків, зате є безліч переваг: безліч пост-ефектів, відмінна руйнівність об'єктів, тривимірні та двовимірні спецефекти, реалістичні текстури і навіть окремий ігровий редактор для роботи з шейдерами та дрібними деталями.[2] На рисунку 1.5 показано роботу у движку Frostbite.

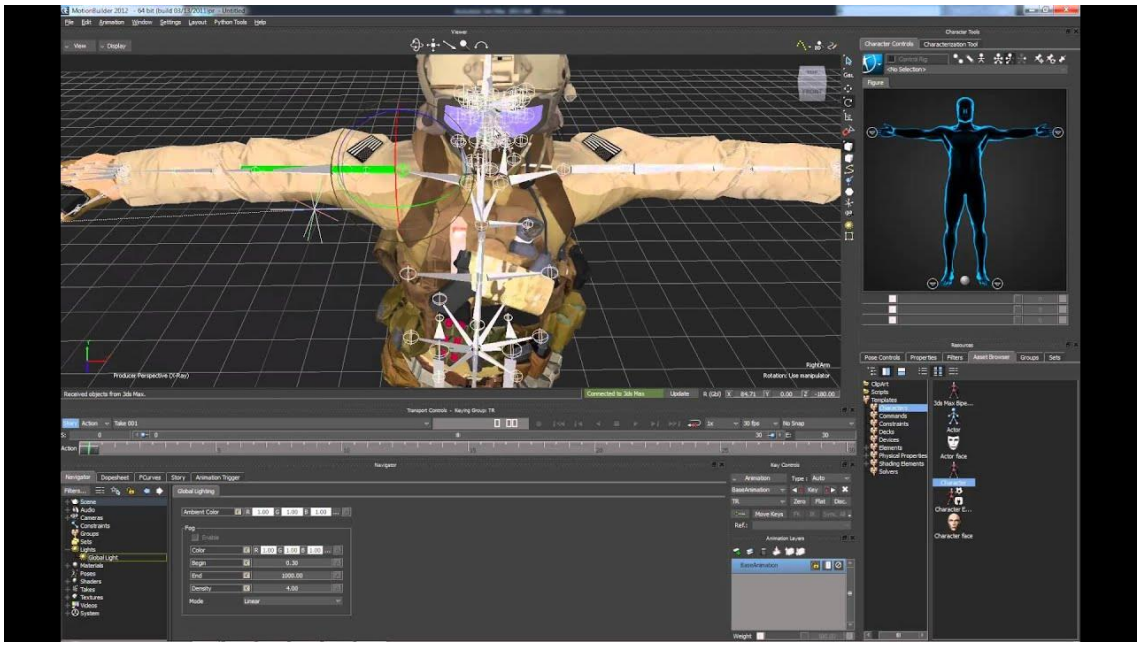

Рисунок 1.5 – Інтерфейс Frostbite

Unity 3d

Unity - міжплатформне середовище розробки комп'ютерних ігор, розроблене американською компанією Unity Technologies. Unity дозволяє створювати програми, що працюють на більш ніж 25 різних платформах, що включають персональні комп'ютери, ігрові консолі, мобільні пристрої, інтернет-програми та інші[3]. Випуск Unity відбувся у 2005 році і з того часу триває постійний розвиток.

Основними перевагами Unity є наявність візуального середовища розробки, міжплатформної підтримки та модульної системи компонентів. До недоліків відносять появу складнощів при роботі з багатокомпонентними схемами та утруднення при підключенні зовнішніх бібліотек.

На Unity написані тисячі ігор, програм, візуалізації математичних моделей, які охоплюють безліч платформ і жанрів. У цьому Unity використовується як великими розробниками, і

незалежними студиями. На рисунку 1.6 показано інтерфейс Unity 3D.

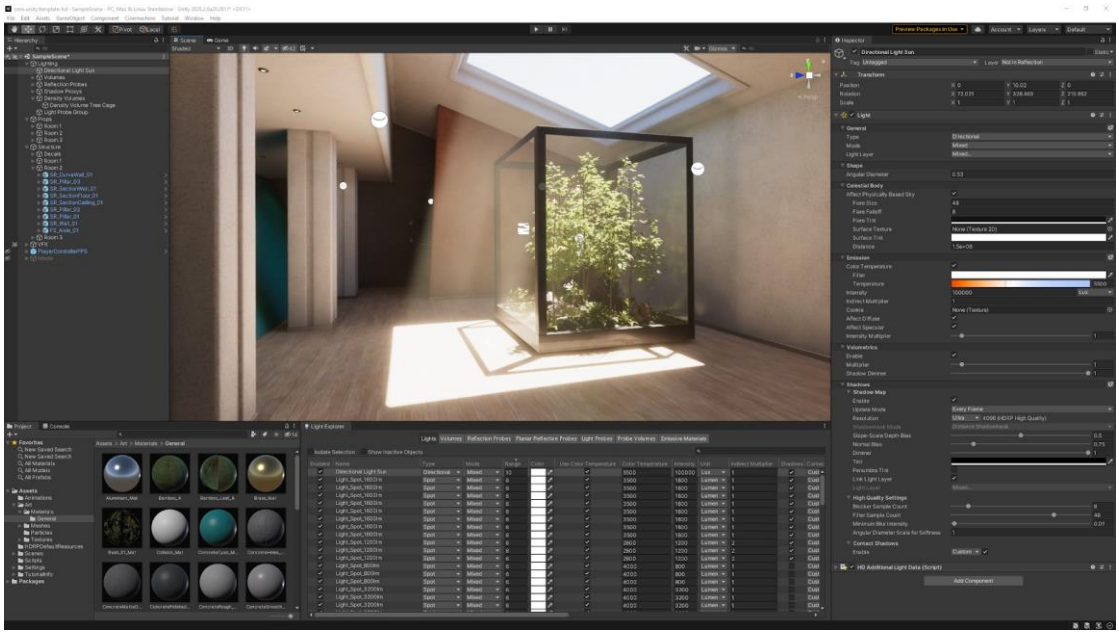

Рисунок 1.6 – Інтерфейс Unity 3d

Переваги та недоліки

Як правило, ігровий двигун надає безліч функціональних можливостей, що дозволяють їх задіяти в різних іграх, в які входять моделювання фізичних середовищ, карти нормалей, динамічні тіні та багато іншого. На відміну від багатьох ігрових двигунів, у Unity є дві основні переваги: наявність візуального середовища розробки та міжплатформова підтримка [3]. Перший фактор включає не тільки інструментарій візуального моделювання, а й інтегроване середовище, ланцюжок складання, що спрямоване на підвищення продуктивності розробників, зокрема етапів створення прототипів та тестування. Під міжплатформною підтримкою надається як місця розгортання

(установка на персональному комп'ютері, на мобільному пристрої, консолі тощо. буд.), а й наявність інструментарію розробки (інтегроване середовище можна використовувати під Windows і Mac OS).

Третьою перевагою називається модульна система компонентів Unity, за допомогою якої відбувається конструювання ігрових об'єктів, коли останні є комбінованими пакетами функціональних елементів. На відміну від механізмів успадкування, об'єкти в Unity створюються за допомогою об'єднання функціональних блоків, а не приміщення у вузли дерева успадкування. Такий підхід полегшує створення прототипів, що є актуальним при розробці ігор .

Як недоліки наводяться обмеження візуального редактора під час роботи з багатокомпонентними схемами, як у складних сценах візуальна робота не може. Другим недоліком називається відсутність підтримки Unity посилань на зовнішні бібліотеки, роботу з якими програмістам доводиться налаштовувати самостійно, і це також ускладнює командну роботу. Ще один недолік пов'язаний із використанням шаблонів екземплярів (англ. prefabs). З одного боку, ця концепція Unity пропонує гнучкий підхід візуального редагування об'єктів, але з іншого боку, редагування таких шаблонів є складним [3]. Також, WebGL-версія движка, в силу специфіки своєї архітектури (трансляція коду з C # C++ і далі в JavaScript), має ряд невирішених проблем з продуктивністю, споживанням пам'яті і працездатністю на мобільних пристроях [4] [5] [6].

1.2 Ігровий двигун Escape from Tarkov

Escape from Tarkov (з англ. — «Втеча з Тяркова») — розрахована на багато користувачів рольова онлайн-гра від першої

особи, що розробляється компанією Battlestate Games, що поєднує в собі жанри FPS і RPG з MMO-елементами.

Розробником та видавцем «Втечі з Таркова» виступає компанія Battlestate Games. Гра заснована на двигуні Unity. Гра для операційних систем Windows та macOS була анонсована 5 листопада 2015[7][8][9][10][11]. З 5 серпня 2016 року розпочалося закрите альфа-тестування, а з 29 грудня — розширене. 28 липня 2017 року розпочалося закрите бета-тестування.

Ігрові механіки

Розробники планують реалізувати реалістичну бойову модель з достовірною фізикою та балістикою; зміну дня та ночі; вплив виснаження, хвороб та травм на поведінку персонажа; та економічну систему, підпорядковану діям гравців. Анонсовано сюжетні місії та різноманітні тактичні завдання від боїв до розвідки та торгівлі[12][13][14].

Історія розробки

Розробники півроку освоювали двигун CryEngine і створили свій демонстраційний рівень. Потім почався пошук інвесторів, але нову компанію не вдалося залучити інвестиції, і проект був «заморожений»[15].

У 2009 році розробники продовжили опрацьовувати всесвіт «Росія 2028», у 2010 році зайнялися розробкою проекту Contract Wars на новому движку Unity. Потім було засновано компанію Battlestate Games. У 2011 році відбувся реліз гри Contract Wars, розробленої 8-ма співробітниками. Частина співробітників залишилася у компанії AbsolutSoft для доопрацювання проекту Contract Wars та розробки його клієнтської версії під назвою Hired

Ops. Розробка «Втечі з Таркова» розпочалася у березні 2014 року[15].

5 листопада 2015 - вийшов трейлер анонсу гри [16] [17].

24 листопада 2015 - розробники представили екшен-трейлер геймплея[18][19].

5 серпня 2016 - стартувала закрита альфа [20].

16 грудня 2016 - анонсований перехід до розширеної альфе [21].

30 травня 2017 року – анонсовано перехід до закритої бети[22].

27 липня 2017 року - стартувала закрита бета [23] [24].

29 грудня 2017 року — вихід книги «Хижак» Олександра Конторовича, за мотивами гри «Escape from Tarkov», вона розповідає історію простого системного адміністратора, який кинув виклик новому світу та став його частиною. Це перша книга у серії[25].

13 червня 2020 року - на онлайн конференції PC Gaming Show 2020, було оголошено про плани випуску гри на консолях поточного покоління[26], і так само була вперше показана нова карта "Вулиці Таркова"[27].

Система кастомізації зброї

Одна з найважливіших фішок гри - система глибокої кастомізації зброї: за словами розробників, в її основі лежить принцип "сама зброя - тільки ствольна коробка/ресивер, все інше можна замінити". Про те, як це працює в Escape від Tarkov, розповідають творці гри[28]. На рисунку показана унікальний інтерфейс для модернізації зброї.

Рисунок 1.7 – Інтерфейс модернізації зброї

Особливості гри

Найголовнішою особливістю гри є реалістична поведінка персонажа та зброї. Персонаж має власну вагу та характеристики, які можливо покращити. Зброя також має власну вагу, у кожної зброї свої унікальні характеристик і поведінка зброї максимально приближена до реального світу.
Усі ці особливості розробники змогли получити за допомогою двигуна Unity 3d. Він дозволяє дуже ретельно налаштувати майже усі механіки для гри. Наприклад: в цій грі кожен патрон – це окремий об’єкт, який сервер так само відстежує, як персонажа[29].

Недоліки

Сервера та оптимізація. Ще у Таркова погано з оптимізацією, і це за такої графіки. Якщо чесно, то до кінця не зрозуміло в чому проблема: у серверах, або у продуктивності гри. Наприклад, американці найчастіше скаржаться на сервери, які,

мовляв, зовсім нікуди не годяться, і якщо справа так піде і далі, то проект повністю втратить західний ринок. Наші ж частіше нарікають на оптимізацію, вказуючи на те, що гра пішла з альфи в бету, а кадрова частота краще не стала: просідання і лаги - вірні її союзники. Але цю проблему якось особливо не висвітлюють. Наша преса так і зовсім мовчить, а основна маса геймерів вважають за краще розповідати про, нібито, невідбалансовану зброю і маленьку карту[30].

1.3 Методи оптимізації ігор

Організуємо проект

Погано організований проект у Unity складно оптимізувати та позбавляти проблем. Особливо якщо над ним працює велика команда. А у нашому випадку ще й розподілена.

Це призводить до повторних матеріалів, текстур, непотрібного дублювання ассетів та заплутаності. З'являються вузькі місця в пам'яті, збільшуються вимоги до пристроїв і час розробки.

Набагато простіше заздалегідь визначити структуру проекту, щоб її дотримуватися. Опишіть все у документації та позначте, де шукати конкретні матеріали та ассети.

Кожному елементу дайте зрозумілі імена, щоб не бачити нескінченні Material (1) або сотні об'єктів з ім'ям Cube.

Створіть окремі папки у проекті для всього – плагінів, матеріалів, моделей, сцен, UI, текстур тощо. У імені кожного асета вкажіть, навіщо він використовується.

Наприклад:

– UI/Common/Icons/IconAbilityFreeze.png

– VFX/Textures/FXAmbientFog01.png

– Models/Characters/Enemies/Boss.fbx

Знаходимо вузькі місця

Unity Profiler — чудовий інструмент у Unity Pro, який дозволяє аналізувати проблемні місця у продуктивності та бачити завантаження доступних ресурсів (CPU/GPU/RAM).

Достатньо запустити білд у режимі розробника та профільник почне викачувати дані про продуктивність. Щоб підключитися до програвача, він повинен бути запущений з прапорцем Development Build (перебуває в діалоговому вікні Build Settings).

Unity Profiler показує графік завантаження процесора під час гри - просто тримайте пристрій підключеним до комп'ютера. Ви побачите всі падіння та зльоти в режимі реального часу: рендеринг, сценарії, фізика, збирач сміття, VSync і так далі. Якщо на мобільних проектах є падіння нижче 30 кадрів/с, то настав час зайнятися оптимізацією.

Зазвичай, проблеми мобільних ігор пов'язані з рендерингом. Графічна частина навантажує насамперед GPU (графічний процесор) та CPU (центральний процесор). Стратегії оптимізації для GPU та CPU дуже відрізняються — іноді навіть виникають ситуації, коли з GPU навантаження починає лягати на CPU і навпаки.

GPU обмежений філрейтом (fillrate) або пропускною здатністю пам'яті. Якщо нижча роздільна здатність екрана збільшила продуктивність — проблема у філлрейті.

CPU обмежено кількістю Draw Calls (виклик відтворення). Перевірте його у вікні Rendering Statistics - якщо він більше

кількох сотень (для мобільних пристроїв) або тисяч (для PC), то доведеться зменшувати кількість об'єктів.

Менш типові проблеми можуть бути, наприклад, у скриптах або фізиці, щоб знайти їх джерело, використовуйте Profiler[31].

FPS Monitor – це потужний спеціалізований інструмент для аналізу продуктивності ПК та ноутбуків. Програма виводить на екран FPS та показники різних сенсорів (CPU/GPU/RAM/NET/DRV та інших) поверх гри. Все працює відразу після встановлення і за бажанням гнучко налаштовується. Вам більше не потрібно перемикатися з гри на робочий стіл або підключати другий монітор для контролю стану заліза.

Основний функціонал:

Висновок FPS (включаючи мінімальний/середній/максимальний час генерації кадру).

Виведення докладних даних з процесора, відеокарти, пам'яті, мережевої активності, дисків і т.д. Усього понад 40 різних показників.

Виведення будь-яких даних за сенсорами та датчиками у числах та/або на наочних графіках та діаграмах.

Оповіщення про критичні значення всіх важливих параметрів (якщо, наприклад, перегрівається відеокарта, або заповнилася оперативна пам'ять, файл підкачки і т.д.)[32].

Знижуємо навантаження

Трохи теорії. Щоб візуалізувати будь-який об'єкт на екрані, CPU має:

З'ясувати, які джерела світла впливають об'єкт.

Налаштувати шейдер та його параметри.

Надіслати команди відтворення (Draw Calls) графічному драйверу, який підготує їх для відправки на відеокарту.

Це буде досить ресурсом, якщо у вас багато видимих об'єктів.

Команди Draw Calls можна порівняти з тим, як художнику доводиться щоразу мити свою кисть, перш ніж малювати іншим кольором. Колір фарби в нашому випадку - це різні матеріали та моделі, які рухаються незалежно один від одного.

Оптимізуємо 3D-моделі

Високодеталізовані об'єкти сильно навантажують залізо. Намагайтеся не використовувати полігонів більше, ніж потрібно, а також зменшуйте кількість швів на UV-карті та жорстких ребер, які подвоюють вершини. Поговоримо про деякі техніки оптимізації 3D-моделей. На рисунку 1.8 показана різниця між високо полігональною моделлю та низко полігональною.

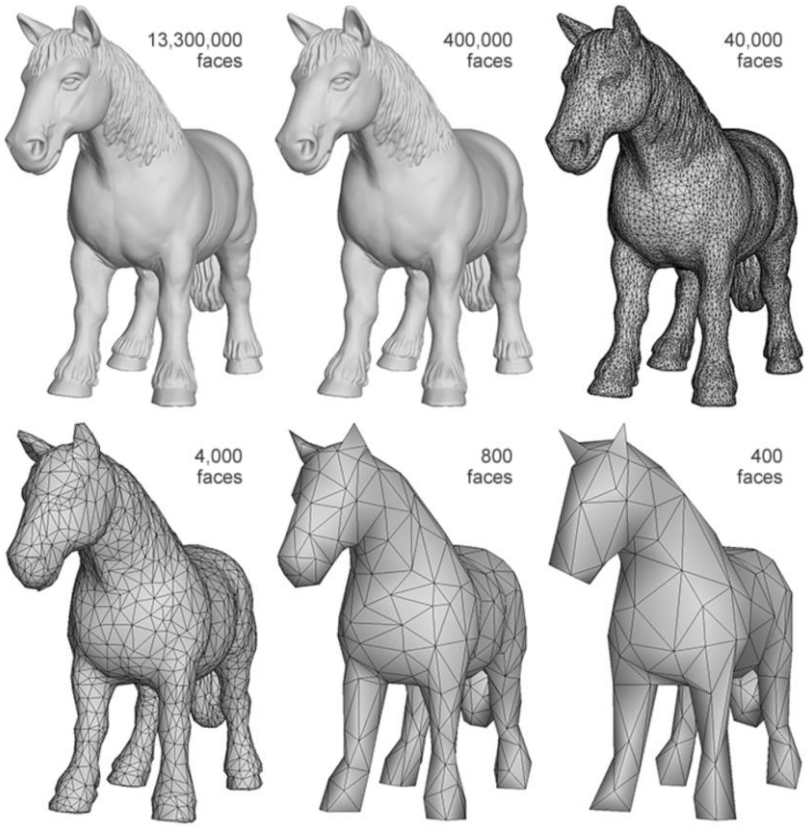

Рисунок 1.8 – Різниця кількості полігонів

Низько полігональні моделі (low-poly). Їх варто використовувати, коли деталізації можна досягти за допомогою текстур чи різних візуальних ефектів. Мета в тому, щоб створити модель, схожу на референс, за мінімальну кількість ребер та вершин. На рисунку 1.9 показана модель та вершини низко полігонального варіанту.

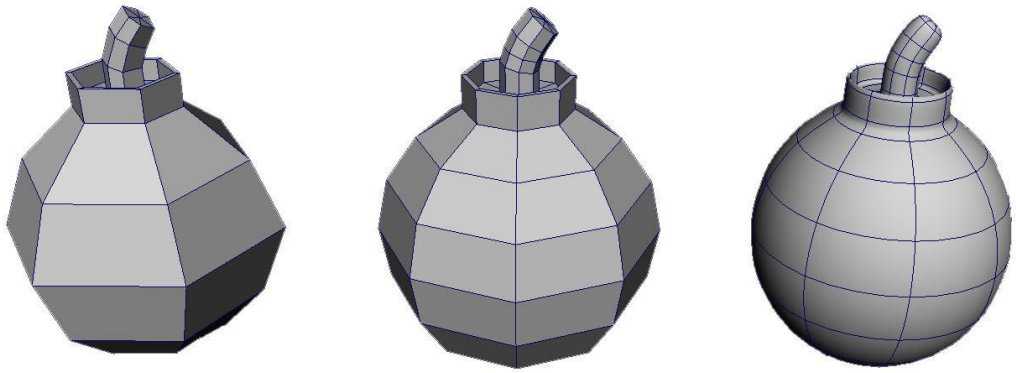

Рисунок 1.9 – приклад низько полігональної моделі

Levels Of Detail (LOD). Коли ви віддаляєте камеру від об'єкта, бачите менше деталей. Але при цьому двигун під час малювання, як і раніше, буде використовувати ту ж кількість трикутників, даремно навантажуючи пристрій.

Виправити це допоможе техніка LOD — вона знижує кількість трикутників, що візуалізуються, в міру видалення об'єкта від камери. Якщо об'єкт знаходиться далеко, то LOD знизить

навантаження на обладнання та підвищить продуктивність малювання. На рисунку 1.10 показано рівень деталізації об’єктів.

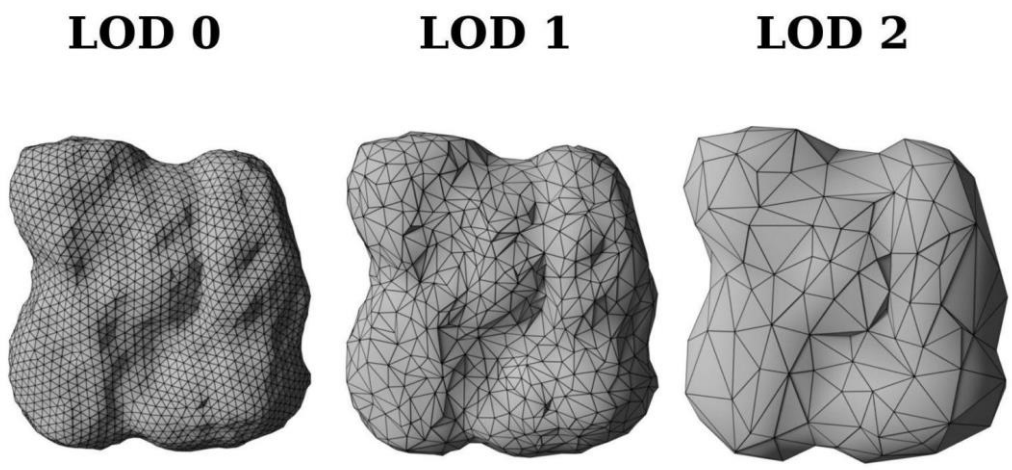

Рисунок 1.10 – Різний рівень деталізації об’єкта

Об'єднуємо об'єкти - використовуємо батчінг

Під час візуалізації кожного об'єкта виконується Draw Call, який надсилається графічному API (наприклад, OpenGL або Direct3D). Той у свою чергу обробляє кожен DC, що потребує багато ресурсів CPU. Але є рішення.

Unity може об'єднувати об'єкти під час виконання, щоб зменшити кількість Draw Call – ця називається батчингом (batching). Іншими словами: це угруповання кількох мішів в один із наступним викликом малювання.

Оптимізуємо текстури

POT текстури (Power Of Two). В ідеалі розміри текстури і у висоту, і в ширину повинні бути кратні двом: 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 і таке інше. У цьому вони мають бути саме квадратними. Наприклад, 2048×256 – це теж POT.

Налаштовуємо камери

Камера – один із головних способів демонстрації вашої гри. Вона може бути змінюваною, заскриптованою або успадкованою для отримання різних візуальних ефектів. Наприклад:

Для пазла статична камера, що охоплює весь рівень.

Для шутера від першої особи камера на рівні очей персонажа.

Для гоночної гри - камера позаду автомобіля, яка слідує за ним.

На рисунку 1.11 показано основний принцип роботи камери між об’єктами та, як далі буде виглядати картина з камери.

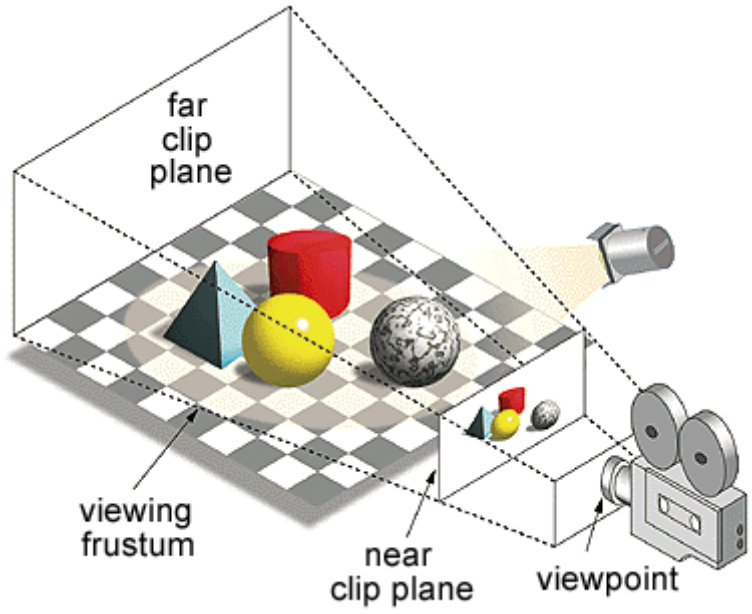

Рисунок 1.11 – Принцип дії камери

Працюємо зі світлом та тінями

Рендеринг освітлення буває двох видів: vertex та pixel.

Вертексне освітлення розраховує світло тільки для вершин ігрової моделі і потім інтерполює його (знаходить проміжні значення) на всю поверхню. На рисунку 1.12 показано вертексне освітлення на движку Unity 3D.

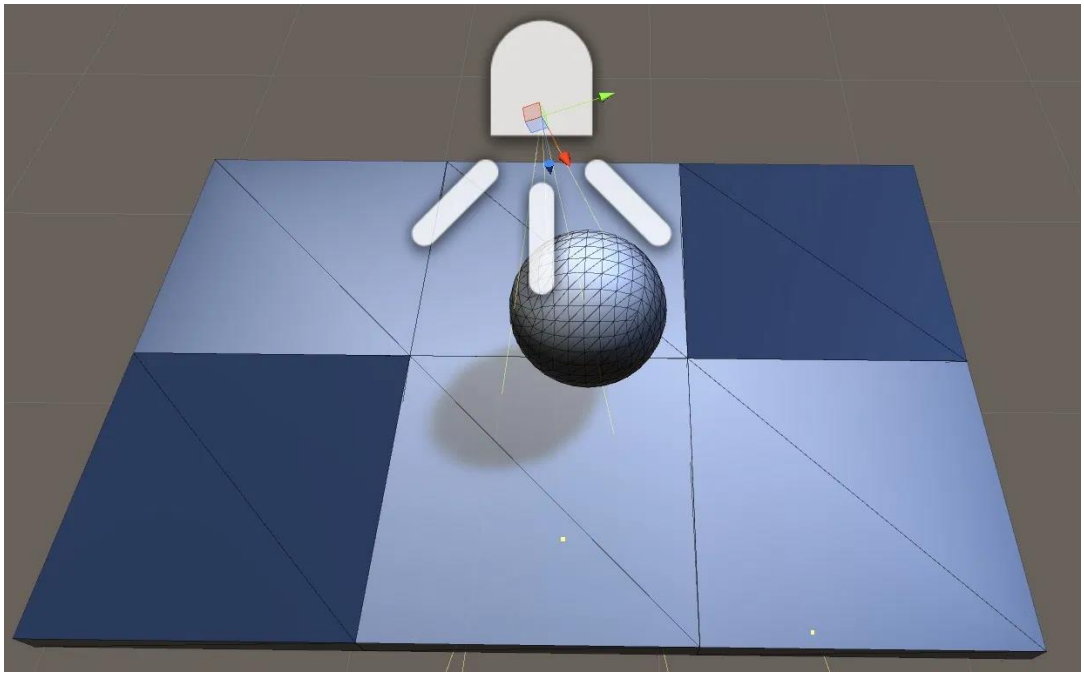

Рисунок 1.12 – Вертексне освітлення

Піксельне освітлення обчислює світло для кожного об'єкта, що відображається. Цей метод забирає більше ресурсів, а й видає якісніший результат[31]. На рисунку 1.12 показано піксельне освітлення.

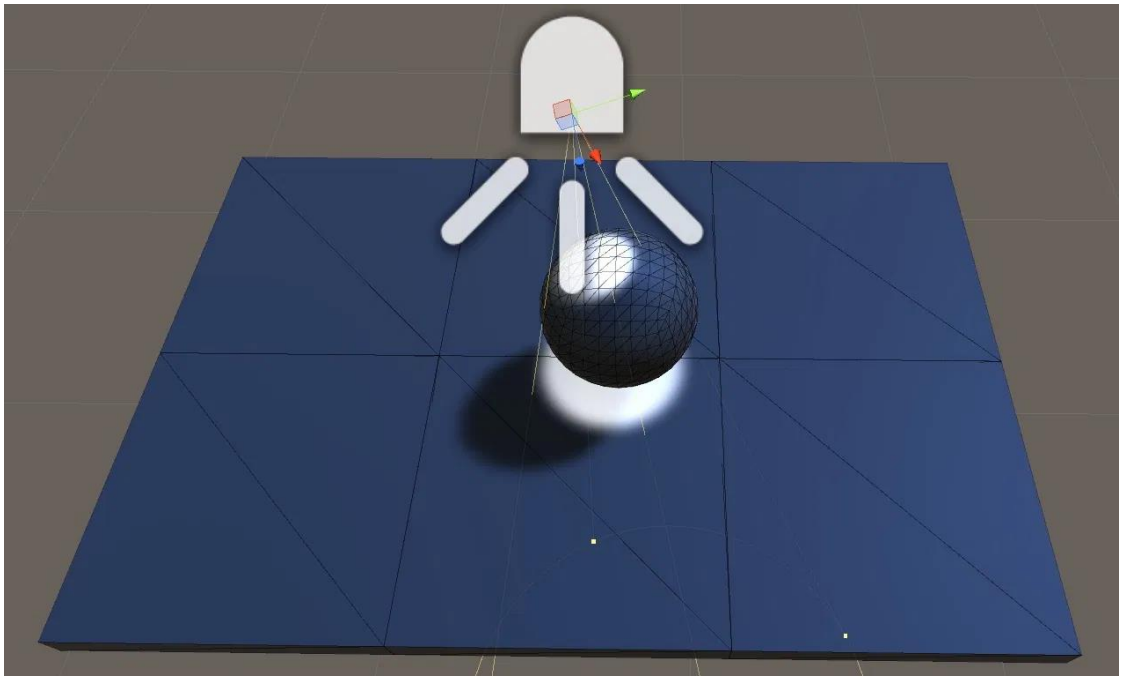

Рисунок 1.13 – Піксельне освітлення

Оптимізація коду

Оптимізація коду - різні методи перетворення коду заради покращення його характеристик та підвищення ефективності. Серед цілей оптимізації можна вказати зменшення обсягу коду, обсягу використовуваної програми оперативної пам'яті, прискорення роботи програми, зменшення кількості операцій введення-виведення.

Головне з вимог, які зазвичай пред'являються методу оптимізації - оптимізована програма повинна мати той самий результат і побічні ефекти тому ж наборі вхідних даних, як і неоптимізована програма. Втім, ця вимога може і не відігравати особливої ролі, якщо виграш за рахунок використання оптимізації може вважатися більш важливим, ніж наслідки від зміни поведінки програми[33]. На рисунку 1.14 приведено простий приклад оптимізації коду.

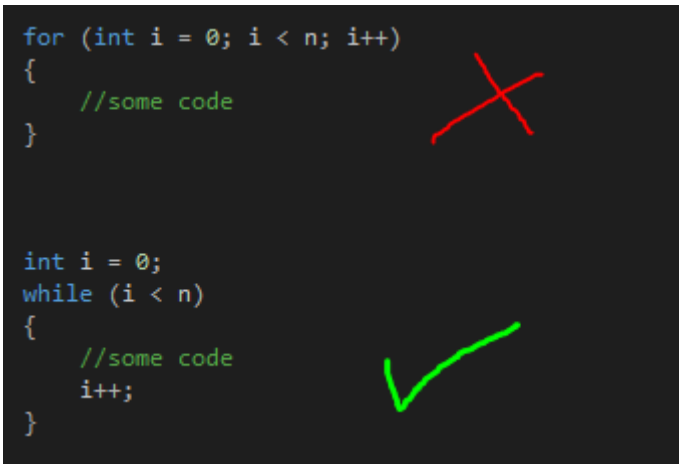

Рисунок 1.14 – Приклад оптимізації коду

1.4 Висновок

Ігрові двигуни дуже важлива річь у грі та їх оптимізація грає дуже важливу роль. Різні ігрові двигуни виконують майже одну і ту саму функції, але кожний має індивідуальну особливість, які відрізняє їх один від одного і дають переваги у створені гри та особливостей для неї. Двигун надає можливість грі бути індивідуальної в плані механіки. В деяких іграх розробник дуже багато ресурсів та сил команди віддає на покращення або більш детальне обчислення фізики, щоб гра була приближена до правдоподібної фізики, яку ми спостерігаємо кожен день.

Ігровий двигун для гри Escape from Tarkov було обрано Unity 3d. Він надає багато можливостей для опрацювання більш детальної фізики об’єктів та обчислення. Для цієї гри двигун був перероблений для опрацювання більшої кількості об’єктів та деталей. Розробники зіткнулись з проблемою потужності на початкових етапах випуску гри. Головною проблемою стала велике споживання оперативної пам’яті із-за великої кількості об’єктів, які потрібно зразу завантажувати на карту.

Оптимізація ігрового двигуна дуже важлива річь у випуску ігор та розробки, так як дуже сильно впливає на відчуття під час гри. Оптимізація може бути направлена на зменшення споживання програмно-апаратних ресурсів або програмного коду. Ці дві оптимізації несуть одну думку, але їх виконання зовсім різне. Зменшити споживання апаратних ресурсів можливо за допомогою алгоритмів двигуна або зменшення деталізації об’єктів. Оптимізація коду виконується за допомогою шаблонів та знаходження критичних крапок в яких йде через мірне використання пам’яті.

2. МЕТОДИ ОПТИМІЗАЦІЇ ПРОГРАМНО-АПАРАТНИХ РЕСУРСІВ

2.1 Методи оптимізації за допомогою алгоритмів двигуна

Bottleneck

Термін "bottleneck" досить очевидний, якщо ми уважніше на нього подивимося. Горловина є найвужчою частиною пляшки, яка веде до отвору, і її мета - зробити налив будь-якої рідини, що міститься всередині, більш контрольованої, обмежуючи кількість рідини, яка може вийти з пляшки.

Тепер, коли ми говоримо про вузьке місце в контексті комп'ютерного обладнання, ми говоримо про низький рівень продуктивності, який викликаний нездатністю одного компонента йти в ногу з іншими, що уповільнює здатність ПК обробляти дані.

Тим не менш, "bottleneck" може бути створено практично будь-яким компонентом, який значно слабший, ніж інші компоненти, що становлять ПК. Це може бути процесор, графічний процесор, оперативна пам'ять або жорсткий диск, які знижують продуктивність - технічно це може бути кілька компонентів, якщо один компонент сильніший за інші.

Однак у 2020 році вузькі місця в процесорах та графічних процесорах є головними факторами, які необхідно враховувати, коли йдеться про вузькі місця у продуктивності, тому нижче ми розглянемо трохи докладніше.

Пляшкова шийка CPU та GPU

Коли йдеться про вузькі місця ЦП і ДП, він може працювати в обох напрямках: або ДП не працює на 100% ємності через повільніше ЦП, або навпаки. У будь-якому випадку, ви

отримуватимете менше FPS.[34] Рисунок 2.1 демонструю принцип bottleneck.

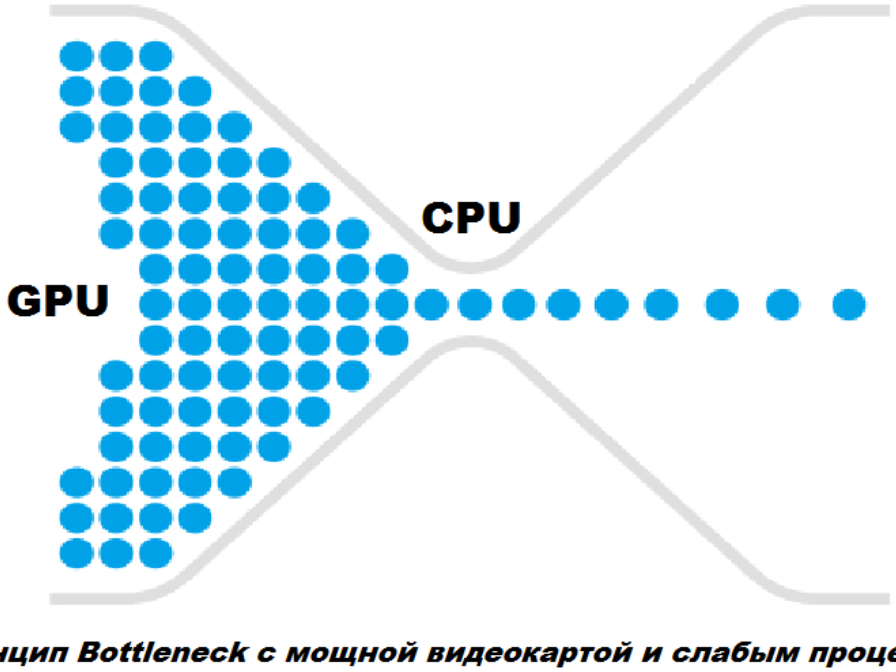

Рисунок 2.1 – Проблема у співвідношенні потужностей

Обрізка у 3D-пакеті

І все ж таки краще, якщо художник може обрізати все зайве у себе в редакторі, скорочуючи таким чином кількість етапів підготовки контенту. Це виправдано, коли модель використовується у сцені лише з одним заздалегідь визначеним поворотом щодо камери. Раніше об'єкти, які точно будуть повернені до користувача однією стороною, часто спрощувалися перед інтеграцією в проект. Важливо відзначити, що здійснювати таке спрощення програмно в Unity значно важче через складність упаковки UV-розгортки, тому автоматизація на етапі 3D-пакету часом полегшує життя художника.

Один із інструментів для роботи з 3D-моделями у нашій компанії — Blender. До нього ми й залізли. Начебто такий «дорослий» софт, як Blender, повинен мати подібний функціонал. Однак виявилося, що не винен. Довелося пиляти свій власний велосипед.

Першою ідеєю було використати всім знайомий інструмент виділення — по суті, повторити частину ручної роботи художника для одного ракурсу камери: виділити видимі полігони, інвертувати виділення, видалити. План був такий: переміщати камеру, у кожній позиції визначати AABB проекції моделі, потім запросити результат виділення полігонів області, що відповідає AABB, отримати об'єднання безлічі полігонів поточного ракурсу з попередніми та в кінці видалити невиділені полігони.

Однак під час реалізації скрипту було виявлено суттєвий недолік з погляду поставленого завдання. Інструменти виділення в Blender (rectangle select, circle select) втрачають точність зі зростанням кількості елементів, що виділяються на одиницю площі екрану (деякі полігони залишаються невиділеними), що робить їх використання в наших засобах автоматизації неможливим. Цікавий факт: у тому ж 3ds Max такої проблеми немає.[35] На рисунку 2.2 та на 2.3 показана виділена область 3д моделі.

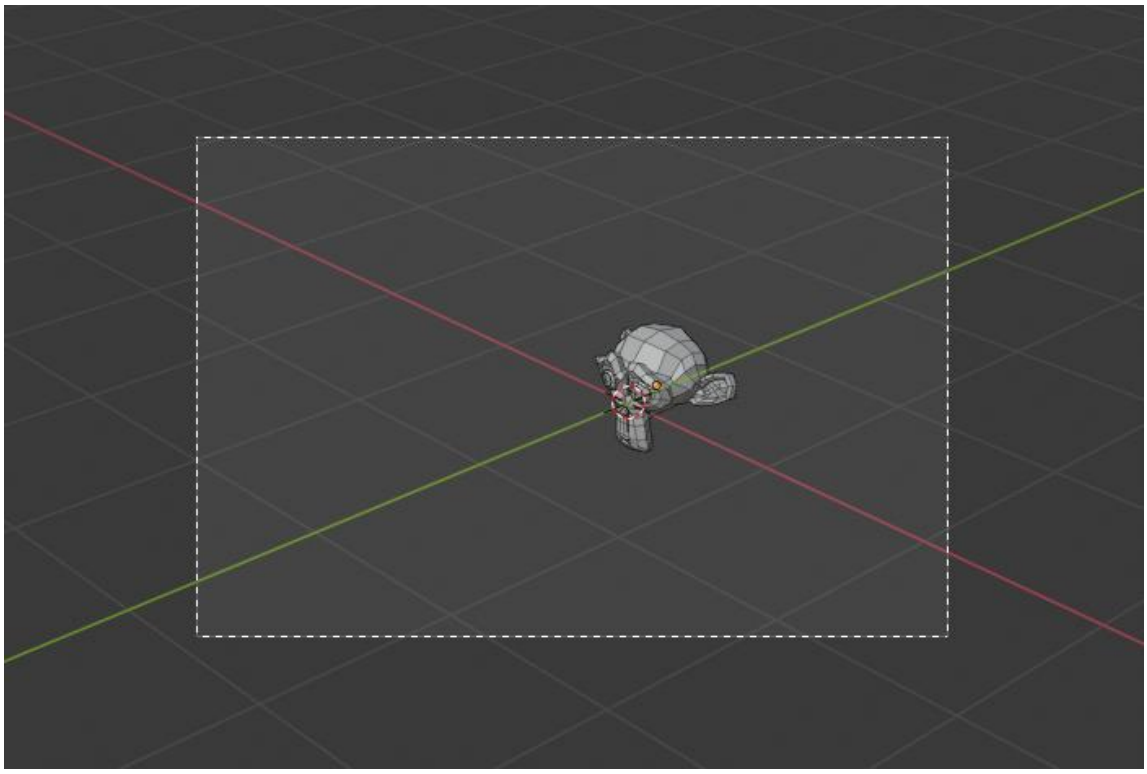

Рисунок 2.2 - Виділення області

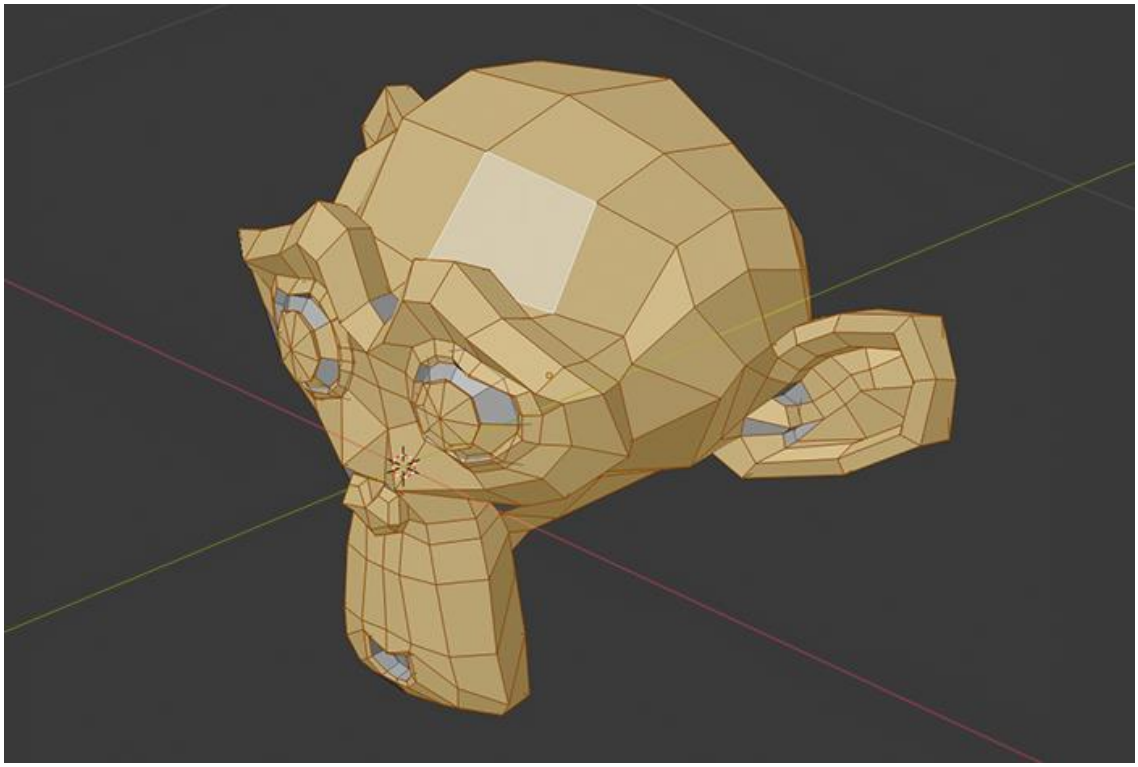

Рисунок 2.3 – Результат

Рівень деталей(LOD)(Level of Detail)

Сітка з високим рівнем деталізації має багато полігонів і добре виглядає крупним планом. Але коли сітка знаходиться далі (і детальність вже не видно), висока полігональність уповільнює роботу без потреби.

Одним із рішень цієї проблеми є використання високодеталізованих сіток для об'єктів, що знаходяться поряд з камерою, та малодеталізованих сіток для об'єктів, віддалених від камери. Коли гравець переміщається по сцені, ви повинні продовжувати замінювати близькі об'єкти більш детальними сітками, а далекі об'єкти менш деталізованими сітками. Мета — зберегти на передньому плані мало високоякісних об'єктів повільного рендерингу та безліч низькоякісних об'єктів швидкого рендерингу у фоновому режимі. (Дослідні користувачі можуть порівняти цей підхід із системою територій JME TerraMonkey, яка внутрішньо використовує спеціалізований алгоритм GeoMipMapping для генерування рівнів деталізації місцевості).

Тепер ви бачите, чому вам знадобиться автоматична генерація рівнів деталізації для складних геометрій.[36] На рисунку 2.4 показано приклад оптимізації за допомогою LOD.

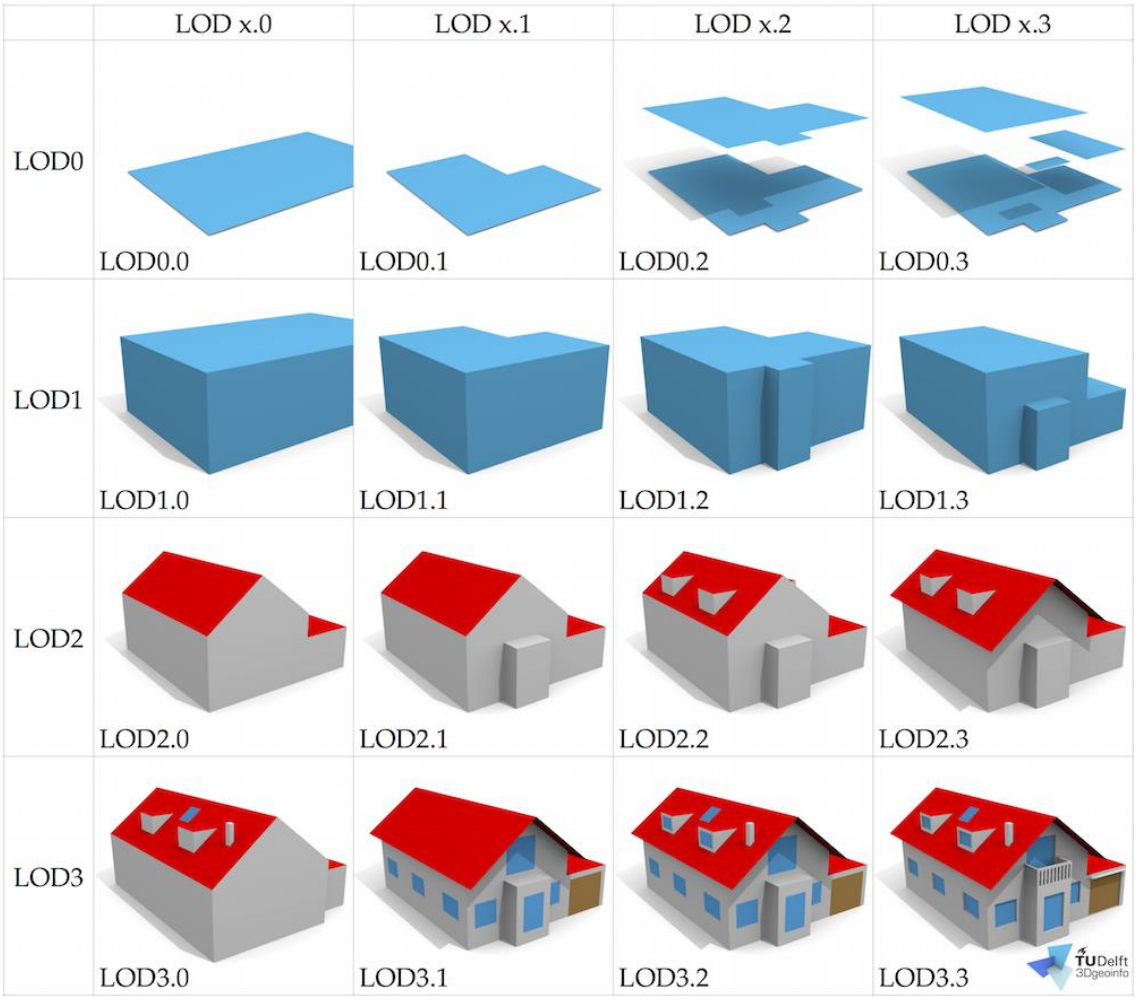

Рисунок 2.4 – Приклад оптимізації за допомогою LOD

Батчінг викликів малювання (Draw Call Batching)

Для відтворення об'єкта на екрані двигун відправляє команду (draw call) графічному API (наприклад, OpenGL або Direct3D). Графічний API робить значну роботу для кожного DC, що дуже впливає на продуктивність CPU.

Unity може об'єднувати об'єкти під час виконання, щоб малювати їх разом у межах одного DC. Ця операція називається "батчінг" (batching). Більшість об'єктів Unity може бачитись для отримання найкращої продуктивності на стороні CPU.

Вбудована підтримка батчингу в Unity має значну перевагу над простим об'єднанням геометрії в програмі моделювання і перед використанням скриптів скрипта CombineChildren, що йде в пакеті Standart Assets). Батчінг в Unity працює після етапу визначення видимості. Двигун обрізає кожен об'єкт індивідуально і кількість геометрії, що візуалізується, залишається таким же як і без батчингу. Поєднуючи ж геометрію в програмі моделювання або іншим шляхом, ви знижуєте ефективність обрізки і, як наслідок, візуалізується значно більша кількість геометрії .

Матеріали

Батчаться тільки об'єкти, що мають один і той же матеріал. Відповідно, для ефективного батчингу вам необхідно робити матеріали загальними для безлічі об'єктів, якщо це можливо.

Якщо у вас є два однакові матеріали, які відрізняються лише текстурами, можна об'єднати ці текстури в одну велику — процес, який часто називають створенням текстурного атласу. Так ви зможете використати один матеріал замість двох.

Якщо потрібно отримати доступ до властивостей загального матеріалу зі скрипту, важливо пам'ятати, що зміна Renderer.material призведе до створення копії матеріалу. Натомість слід використовувати властивість Renderer.sharedMaterial, щоб матеріал залишився загальним.

Динамічний батчінг (Dynamic Batching)

Unity може автоматично бачити об'єкти, що рухаються в один DC, якщо вони використовують загальний матеріал і відповідають ряду інших критеріїв. Динамічний батчінг застосовується автоматично і не вимагає додаткових дій з вашого боку.

Динамічний батчинг пов'язаний з додатковим навантаженням для кожної вершини, так що він застосовується тільки до мішів, що містять менше 900 вершин у сумі.

Якщо ваш шейдер використовує Vertex Position, Normal та єдиний UV, то ви можете бачити до 300 вершин; тоді як, якщо шейдер використовує Vertex Position, Normal, UV0, UV1 та Tangent, то лише 180 вершин.

Зверніть увагу: обмеження кількості атрибутів у майбутньому може бути змінено

Як правило, об'єкти повинні мати один масштаб.

Виняток для неоднорідно масштабованих об'єктів; якщо кілька об'єктів мають різне неоднорідне масштабування, всі вони все ще можуть бачитися.

Використання різних екземплярів матеріалів — навіть якщо вони по суті є одним матеріалом — унеможливить динамічний батчинг.

Об'єкти з картами освітлення мають додаткову властивість: індекс карти освітлення та зміщення/масштаб усередині карти освітлення. Для батчинга об'єкти повинні посилатися на те саме місце в карті освітлення.

Багатопрохідні шейдери можуть порушити батчінг. Багато шейдерів Unity підтримують кілька джерел світла при forward rendering, і ефективно робити додатковий прохід для них. DC для “додаткових піксельних джерел світла” не бачать.

Об'єкти, які приймають тіні в реальному часі, не бачать.

Статичний батчінг (Static Batching)

Статичний батчінг дозволяє двигуну знизити кількість DC для геометрії будь-якого розміру, якщо вона не рухається і використовує загальний матеріал. Статичний батчинг ефективніший, ніж динамічний. Ви повинні намагатися використовувати статичний батчінг, щоб знизити навантаження на CPU.

При використанні статичного батчингу ви повинні переконатися, що об'єкти статичні та не рухаються, не обертаються та не масштабуються під час виконання. Якщо ці умови дотримуються, можна позначити об'єкти як статичні, поставивши галочку Static у Inspector: [37] На рисунку 2.5 та 2.6 показані дані до батчінгу та після батчінгу.

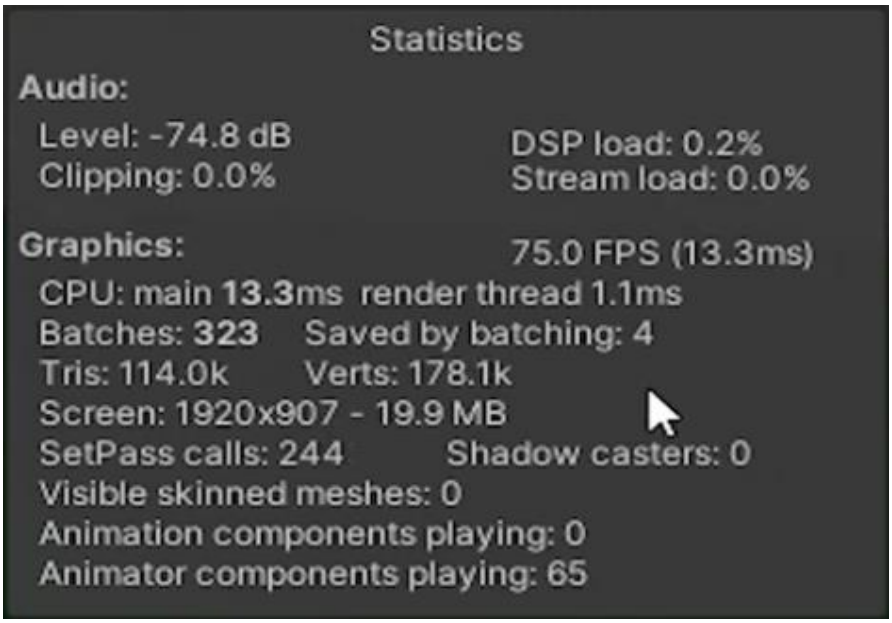

Рисунок 2.5 – До оптимізації

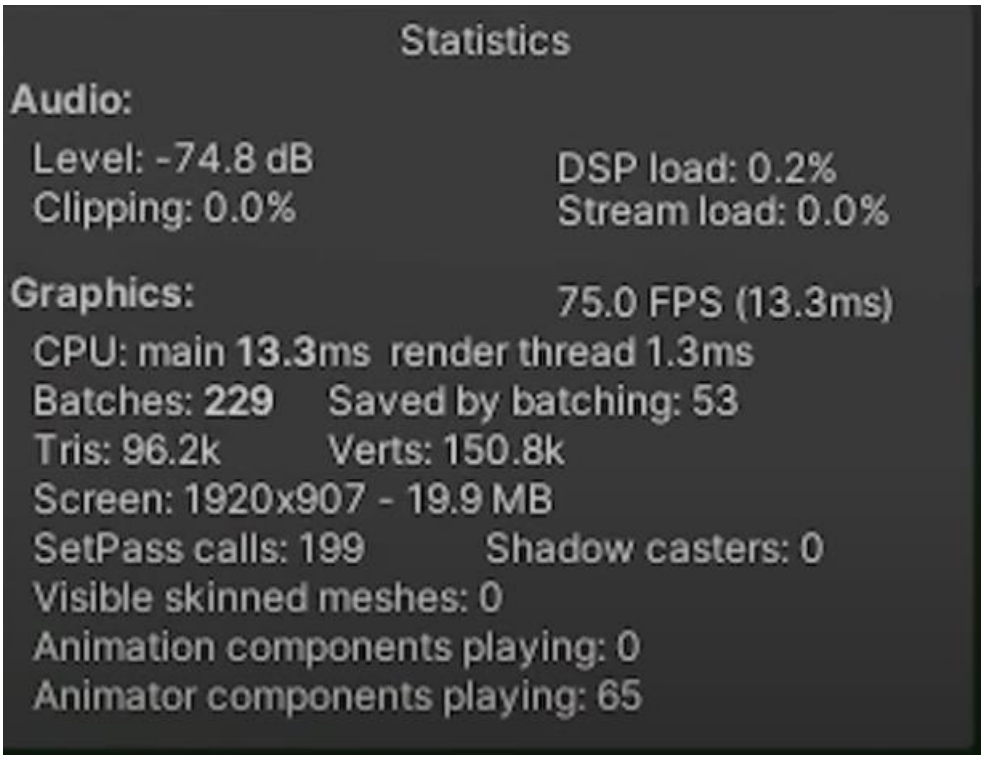

Рисунок 2.6 – Після оптимізації

На рисунку 2.5 ми ще не використовували батчінг, тому там показник Batches: 323. Після увімкнення динамічного батчінгу ми отримали значення в 229 одиниць.

2.2 Методи оптимізації за допомогою програмного коду

Оптимізація заради гравців та власного психічного здоров'я

Досить часто інді-розробники імітують методи оптимізації великих компаній. Це не завжди погано, але прагнення до оптимізації гри вже після проходження точки неповернення - хороший спосіб звести себе з розуму. Розумною тактикою відстеження ефективності оптимізації буде сегментування цільової аудиторії та вивчення характеристик її машин. Бенчмаркінг гри з урахуванням комп'ютерів та консолей потенційних гравців допоможе зберегти баланс між оптимізацією та власним психічним здоров'ям.

Основи оптимізації коду

Насправді, є досить мала кількість оптимізацій, які майже завжди можна використовувати для підвищення швидкості гри. Більшість із них не прив'язана до конкретної платформи (деякі движки та фреймворки враховують їх), тому нижче я покажу приклади на псевдокоді, щоб ви знали, з чого починати.

Мінімізація впливу об'єктів за межами екрану

Нерідко цим займаються двигуни, а іноді навіть самі GPU. Мінімізація обсягу обчислень для об'єктів за межами екрана є надзвичайно важливою. У власній архітектурі краще розділяти об'єкти на два «шари» — перший буде графічним уявленням об'єкта, другий — даними та функціями (наприклад, його розташування). Коли об'єкт знаходиться за межами екрана, нам більше не потрібно витрачати ресурси на його рендеринг і займатися його відстеженням. Відстеження за допомогою таких змінних параметрів, як позиція і стан, значно знижує потреби в ресурсах.

В іграх з великою кількістю об'єктів або об'єктів з великими обсягами даних може виявитися корисним зробити ще один крок і створити окремі процедури оновлення. Одна процедура буде виконувати оновлення, коли об'єкт знаходиться на екрані, інша - коли він за межами. Налаштувавши такий поділ, ми зможемо вберегти систему від необхідності виконання багатьох анімацій, алгоритмів та інших оновлень, які необов'язкові, коли об'єкт прихований.

На рисунку 2.7 показано приклад псевдокоду класу об'єкта, що використовує прапори та обмеження розташування:

```
Object NPC {
    Int locationX, locationY; //текущая позиция объекта на 2d-плоскости

    Function drawObject() {
        //функция отрисовки объекта, вызываемая в цикле обновления экрана
    }

    //функция, проверяющая, находится ли объект в текущем вьюпорте
    Function pollObjectDraw(
        array currentViewport[minX,minY,maxX,maxY]
        ) {

    //если он находится внутри вьюпорта, сообщаем, что его нужно отрисовывать
        If (this.within(currentViewport)) {
            Return true;
        }
        Else {
            Return false;
        }
    }

}
```

Рисунок 2.7 - Приклад псевдокоду

Хоча цей приклад дуже спрощений, він дозволяє нам опитувати об'єкти та визначати їх видимість перед відмальовуванням, завдяки чому можна виконувати спрощену функцію замість виконання повного виклику. Щоб відокремити функції, що не є графічними викликами, може знадобитися створення додаткового буфера - наприклад, функція, в яку включено все, що незабаром може побачити гравець, а не тільки те, що він може бачити зараз.

Незалежність від оновлення кадрів

У движках і фреймворках зазвичай є об'єкти, що оновлюються в кожному кадрі або циклі (tick). Це сильно

навантажує процесор, і щоб зменшити навантаження, ми повинні по можливості позбутися оновлення в кожному кадрі.

Перше, що потрібно відокремити, — це функції рендерингу. Такі виклики зазвичай дуже активно використовують ресурси, тому інтеграція виклику, який повідомляє нам, чи змінилися візуальні властивості гравця, сильно знизити обсяг рендерингу.

Можна зробити ще один крок та використовувати для наших об'єктів тимчасовий екран. Малюючи об'єкти безпосередньо в тимчасовий контейнер, ми можемо гарантувати, що вони будуть малюватись лише за потреби.

Безпосередні обчислення та пошук значень

Ця оптимізація застосовувалася з перших днів ігрової промисловості. Вибравши компроміс між обчисленнями та пошуком значень, можна значно знизити час обробки. В історії геймінгу добре відомим прикладом такої оптимізації є зберігання значень тригонометричних функцій у таблицях, тому що в більшості випадків ефективніше зберігати велику таблицю та отримувати дані з неї, а не виконувати розрахунки на льоту, що збільшує навантаження на процесор.

Сьогодні нам рідко доводиться робити вибір між збереженням результатів та виконанням алгоритму. Однак все ще зустрічаються ситуації, в яких такий вибір може знизити обсяг використовуваних ресурсів, що дозволяє додати нові можливості, не перевантажуючи при цьому систему.

Реалізацію такої оптимізації можна почати з визначення обчислень, що часто виконуються в грі, або частин обчислень: чим більше обчислення, тим краще. Одноразове виконання частин алгоритму, що повторюються, і збереження їх значень часто може

заощадити значну частку обчислювальних ресурсів. Навіть виділення цих частин окремі ігрові цикли допомагає оптимізувати продуктивність.

Наприклад, у багатьох шутерах із видом зверху часто є великі групи ворогів, які виконують однакові дії. Якщо у грі є 20 ворогів, кожен із яких рухається по дузі, то замість обчислення кожного руху окремо, більш ефективно зберігатиме результати роботи алгоритму. Завдяки цьому їх можна змінювати виходячи з початкової позиції ворога.

Щоб зрозуміти, чи буде цей метод корисним у вашій грі, спробуйте використати бенчмарки для порівняння різниці ресурсів, що використовуються при обчисленнях та зберіганні даних.

Використання часу простою процесора

Це більше стосується використання недіючих ресурсів, але при правильній реалізації для об'єктів і алгоритмів ви можете розташовувати завдання таким чином, щоб підвищити ефективність коду.

Щоб почати застосовувати чутливість до простою у своєму ПЗ, вам спочатку потрібно виділити ті внутрішньоігрові завдання, які не критичні на час і можуть обчислюватися до того, як вони стануть потрібні. Насамперед варто шукати код із подібним функціоналом у тому, що відноситься до атмосфери гри. Погодні системи, які не взаємодіють із географією, фоновими візуальними ефектами та фоновим звуком зазвичай можна віднести до обчислень під час простою.

Крім атмосферних обчислень, до області обчислень під час простою належать обов'язкові обчислення. Можна зробити більш

ефективними обчислення штучного інтелекту, що відбуваються незалежно від гравця (тому що вони або не враховують гравця, або поки не взаємодіють з гравцем), а також обчислювані рухи, наприклад скриптові події.

Створення системи, що використовує режим простою, не просто забезпечує підвищену ефективність - її можна застосовувати для масштабування візуальної якості. Наприклад, на слабкій машині гравцю може бути доступний лише базовий ("ванільний") ігровий процес. Однак якщо система виявляє кадри, в яких майже не виконується обчислень, ми можемо використовувати їх для додавання частинок, графічних подій та інших атмосферних штрихів, що надають грі більше пафосу.

Для реалізації такої можливості потрібно використовувати функціонал, наявний у вибраному движку, фреймворку або мові, що дозволяє визначати, наскільки використовується процесор. Задайте у своєму коді прапори, що дозволяють з легкістю визначити об'єм «зайвих» обчислювальних ресурсів та налаштовуйте підсистеми таким чином, щоб вони перевіряли ці прапори та поводилися відповідно.[38]

2.3 Аналіз апаратних комплектуючих

Навантаження на процесор в іграх

Якщо говорити загалом, то серце вашої системи відповідає за різні математичні розрахунки, швидкість виконання яких безпосередньо залежить від його продуктивності. Приріст продуктивності досягається шляхом збільшення тактової частоти або шляхом збільшення кількості ядер і потоків. У дорогих процесорів, як відомо висока герцовка і, як правило, всі вони є представниками багатоядерного сімейства, а отже, справляються з

поставленим перед ними завданням набагато швидше, ніж урізані моделі. Щоб краще розуміти, що саме дає користувачеві високопродуктивний камінь, наведу кілька прикладів.

Обробка команд користувача

За допомогою посередника в особі материнської плати процесор доставляє відсортовані за типом дані до різних комплектуючих і від них приймає певну інформацію, а потім обробляє її. Виходить кругообіг інформації всередині системи, в центрі якої знаходиться цей кремнієвий шматок. Якість та швидкість будь-яких взаємодій користувача з комп'ютером за допомогою пристроїв введення даних залежить безпосередньо від продуктивності ЦПУ. Тобто, за можливість керувати в грі персонажем за допомогою натискань на клавіатуру та пересування мишею, можете подякувати в першу чергу ЦПУ. Кожне натискання на клавішу відправляє процесору інформацію, він її обробляє й у грі відбувається певну дію. Так ось між натисканням і результатом вашого натискання проходить n кількість часу, яке потрібно процесору на обробку. Чим процесор продуктивніший, тим швидше відбудеться обробка сигналу, а відповідно затримка відгуку буде мінімальною. Ви можете спостерігати затримку відгуку, якщо запустите важку ігрову програму на старому процесорі. Повертаючи мишкою у грі, ви побачите, що поворот камери відбудеться за одну-дві секунди після того, як ви посунули гризуна. Це говорить про недостатню потужність ЦПУ. На кількість кадрів на секунду ця обставина не впливає, але, на зручність геймплея ще як. Безумовно, якщо ви не досвідчений дорогими залізняками користувач, то можна пограти і так, але на враження від гри безпосередньо впливає ще кілька факторів, що залежать від процесора.

Побудова навколишнього середовища

Для того, щоб розібратися за що відповідає процесор у грі, доведеться трохи торкнутися теми 3D моделювання. Майже все, що ви бачите у грі, є моделями. Будинки, персонажі, машини, зброя, дерева тощо – все це окремі моделі. За їхню деталізацію відповідає графічний прискорювач, а ось за їхню побудову та розстановку у просторі відносно один одного – процесор. Тобто виходить, що першим у роботу включається саме ЦПУ, він збирає всі необхідні дані та відправляє їх відеокарті, щоб вона зайнялася малюванням та деталізацією кожного об'єкта. Якщо говорити більш простою мовою, то діалог між двома комплектуючими буде виглядати так:

Процесор: "Гей, пссс, подруго, я тут побудував каркас нашого спільного проекту, але не завдання, виглядає все якось не дуже, ти не могла б мені допомогти?"

Ну і відеокарта, як представниця жіночого роду, який дуже любить красу, не може відмовити своєму другові-технарю і відповідає йому: "Так, звичайно, я зроблю з цього неотесаного шматка каменю цукерку".

Якщо рівень вашого шматочка кремнію буде сильно відставати від мінімальних системних вимог гри, то в грі ви спостерігатимете не повне завантаження об'єктів і в такому випадку діалог між комплектуючими вже виглядатиме таким чином:

Відеокарта: "Прийом, ти там живий? Я вже закінчила свою роботу, інші вказівки є?"

На що процесор відповідає: "Чекай, я тут трохи задумався і не можу зрозуміти земля повинна бути під танком або над ним ..."

У такому разі і відбуваються фризи та мікростаттери, коли картинка зависає ненадовго, а процесор у цей час напружує всі свої звивини, щоб не помилитися з підрахунками. Тому в сучасних іграх, де налічується величезна кількість моделей та різноманітних взаємодій між ними, наявність високопродуктивного процесора – обов'язкова.

Математичні алгоритми

У будь-якому тривимірному ігровому додатку дуже багато речей працюють на певних закладених розробниками алгоритмах, прорахунками яких займається процесор. Найбанальніший і наочний приклад практично в будь-якій грі це тіні, що відкидаються об'єктами. Для найпростішої появи тіні, скажімо від дерева, процесору потрібно розрахувати відстань від джерела світла до об'єкта, що відкидає тінь, кут падіння світлових променів, динамічна зміна об'єктів у просторі, взаємодія з іншими об'єктами оточення, інтенсивність освітлення та багато іншого. І це лише для якоїсь нікчемної тіні, на яку гравець навіть не звертає уваги. Тепер уявіть, скільки об'єктів може одночасно перебувати у полі зору гравця, взаємодіючи при цьому між собою. І всіма цими підрахунками має зайнятися ЦПУ. І подібного роду алгоритмів у будь-якій грі налічується безліч, які стосуються практично будь-якого елемента геймплею. Скажімо, ваш персонаж стоїть на місці та не діє. Через певний час, при дотриманні величезної кількості умов, ваш персонаж буде говорити одну з кількох фраз, яка буде обрана виходячи із знову ж таки виконаних умов алгоритму. І чим різноманітнішими розробники намагаються зробити своє дітище, тим вищими будуть системні вимоги до заліза, а зокрема до процесора. Візуальну складову в будь-якій грі досить просто покращити, просто створивши більш детальні моделі в парі з

якісним та реалістичним освітленням. Те саме стосується і даунгрейду, коли спеціально погіршують зовнішній вигляд. Таке часто можна спостерігати з портованими проектами на консолі, адже вони не відрізняються високопродуктивною начинкою. А ось для того, щоб гра виглядала, як можна реалістичніше, якщо можна так висловитися, творцям додатків доводиться засовувати у свої проекти безліч математичних формул. Для того, щоб ви розуміли, наскільки сильно процесор задіяний у будь-якій грі, наведу ще один стомлюючий приклад. У сучасних іграх найчастіше зустрічаються так звані NPC. NPC – це персонажі, які перебувають під керуванням гравця, і які запрограмовані певні дії у разі певних подразників.

Розрахунки фізики

Грунтуючись на вищезгаданих математичних розрахунках у сучасних іграх, присутня величезна кількість об'єктів, які схильні до фізики ігрового двигуна. Безумовно, вона відрізняється від реальної фізики з тієї простої причини, що нинішні процесори не мають достатньої продуктивності для таких складних розрахунків. Посудіть самі, коли на автомобілі в грі ви падаєте з урвища, ви летите вниз з певною швидкістю і певною траєкторією. Зіткнувшись із землею, конструкція машини змінюється певним чином і після ДТП авто продовжує рух без зусиль гравця згідно з інерцією. Все це є фізика в грі. І що більш вона реалістична, тим, як ви вже здогадалися, потрібний більш продуктивний камінь. У реальному житті результат подібної події залежить від величезної кількості факторів: швидкість машини до зльоту вниз, прискорення вільного падіння, висота обриву, матеріали автомобіля, щільність поверхні та багато іншого. Насправді подібних змінних за умов такої події просто не порахувати, а тому

відтворити таку складну з погляду фізики подію у грі неможливо. Ви просто уявіть, які зусилля повинні бути докладені для створення таких алгоритмів і яка обчислювальна потужність буде потрібна, щоб все це належним чином розрахувати. Тому в іграх нашого часу існує дуже спрощена система фізичних розрахунків.

P.S. У серпні 2009 року англомовний журнал Game Developer, присвячений розробці комп'ютерних ігор, опублікував статтю про сучасні ігрові движки та їх використання. За даними журналу, найбільш популярним серед розробників є двигун nVidia PhysX, який займає 26,8% ринку. На другому місці знаходиться Havok, який займає 22,7% ринку. Третє місце належить движку Bullet Physics Library (10,3%), а четверте – Open Dynamics Engine (4,1%).

Як і у випадку з класичними математичними розрахунками, процесор, будучи альфонсом, не гидує допомогою відеокарти і тут, перекладаючи на неї частину своїх обов'язків. Наприклад, вищезгаданий знаменитий двигун від компанії Nvidia – PhysX, адаптований для прискорення фізичних розрахунків на графічних чіпах з архітектурою CUDA. Але це не означає, що ЦПУ менш важливий, як ви могли зрозуміти, йому реально є чим зайнятися, він у нас хлопець взагалі різнобічний та багатозадачний.

Взагалі кажучи, про фізику в іграх, слід розуміти, що чим більше в грі об'єктів, які піддаються фізичним законам движка, тим, як ви вже здогадалися, продуктивніше буде потрібно ЦПУ. Уявіть, наскільки сильно виросло б навантаження на залізо, якби всі внутрішньоігрові об'єкти мали поведінкові особливості згідно з фізикою. Взяти, наприклад, ту ж рослинність у будь-якій фантастичній грі з відкритим світом, де багато красивих пейзажів природи. Модельки трави часто не мають взагалі ніяких

здібностей взаємодії з навколишнім світом, в основному просто звуковий супровід при контакті з гравцем, прописане в скрипті. Якщо в грі присутня динамічна зміна погоди, то трава все одно поводитиметься однаково, просто нібито погойдуючись від вітру, однак це не результат взаємодії з погодними умовами, а просто запрограмована поведінка моделі. І до речі саме через брак обчислювальної потужності заліза, ми досі бачимо низькодеталізовані 2D моделі чагарників, що коливаються. Та сама історія і з зачісками головних персонажів, які виглядають відносно загальної картини значно гірше. На рисунку 2.8 показано математичний алгоритм в іграх.

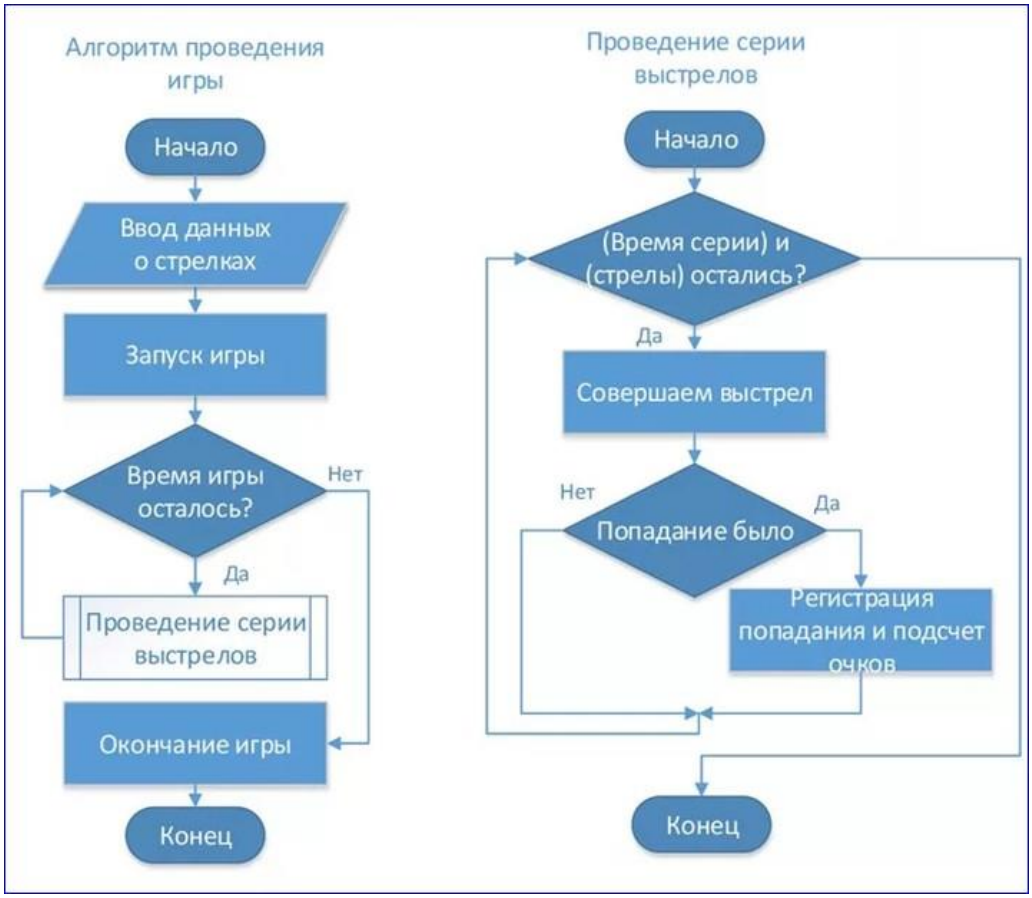

Рисунок 2.8 – Математичний алгоритм

Навіщо потрібна відеокарта

Відеокарти є сполучною ланкою між користувачем та ПК. Вони переносять інформацію, оброблювану комп'ютером, на монітор, цим сприяючи взаємодії між людиною та ЕОМ. Крім стандартного виведення зображення, цей пристрій виконує обробні та обчислювальні операції, в деяких випадках, розвантажуючи цим процесор. Давайте докладніше розглянемо дію відеокарти за різних умов. На рисунку 2.9 показано приклад взаємодії 3 компонентів.

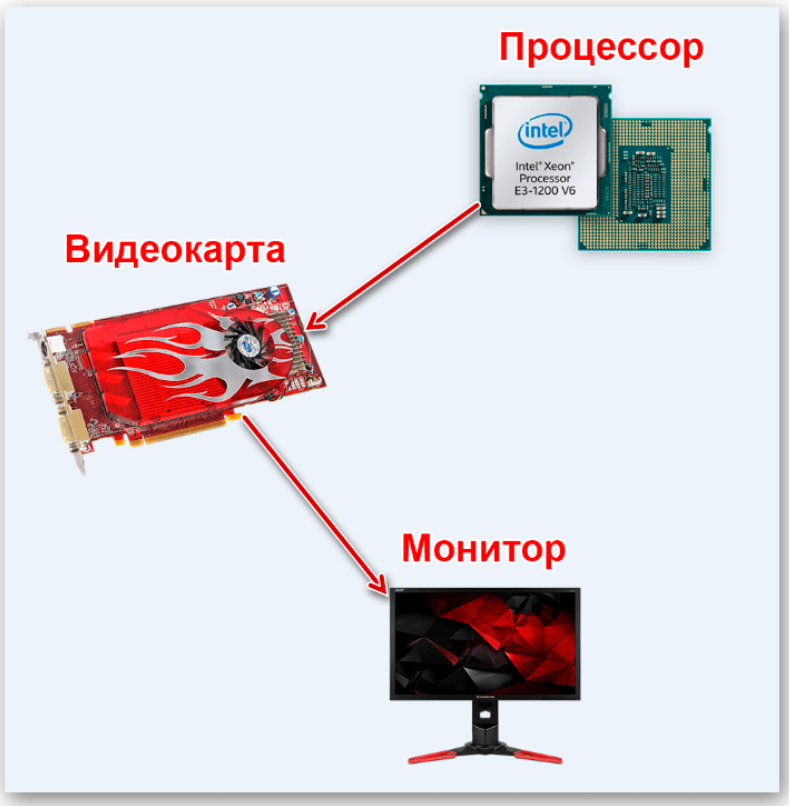

Рисунок 2.9 – Принцип зв’язку

Основна роль відеокарти

Ви бачите зображення на своєму моніторі завдяки тому, що відеокарта обробила графічні дані, перевела їх у відеосигнали та відобразила на екрані. Сучасні відеокарти (GPU) є автономними

пристроями, тому розвантажують оперативну пам'ять та процесор (CPU) від додаткових операцій. Не можна не відзначити, що зараз графічні адаптери дозволяють підключити монітор за допомогою різних інтерфейсів, тому пристрої здійснюють перетворення сигналу активного типу підключення.

Підключення через VGA поступово старіє, і якщо на відеокартах ще зустрічається цей роз'єм, то на деяких моделях моніторів він відсутній. DVI трохи краще передає зображення, проте нездатний приймати звукові сигнали, через що поступається підключенню через HDMI, що удосконалюється з кожним поколінням. Найпрогресивнішим вважається інтерфейс DisplayPort, він схожий на HDMI, проте має ширший канал передачі інформації. На нашому сайті ви можете ознайомитися з порівнянням інтерфейсів підключення монітора до відеокарти та вибрати відповідний для себе.

Крім цього, варто звернути увагу на інтегровані графічні прискорювачі. Оскільки вони є частиною процесора, підключення монітора здійснюється тільки через роз'єми на материнській платі. А якщо ви маєте дискретну карту, то підключайте екрани тільки через неї, так ви не будете задіяти вбудоване ядро і отримаєте більшу продуктивність.

Роль відеокарти у іграх

Багато користувачів отримують потужні відеокарти виключно для запуску сучасних ігор. Графічний процесор перебирає виконання основних операцій. Наприклад, для побудови видимого гравцю кадру відбувається прорахунок видимих об'єктів, освітлення та постобробка з додаванням ефектів та фільтрів. Все це лягає на потужності GPU, а CPU виконує лише невелику частину всього процесу створення зображення.

З цього і виходить, що чим потужніша відеокарта, тим швидше відбувається обробка необхідної візуальної інформації. Висока роздільна здатність, деталізація та інші налаштування графіки вимагають велику кількість ресурсів та часу на обробку. Тому одним із найважливіших параметрів при підборі є обсяг пам'яті GPU. Докладніше про вибір ігрової карти ви можете прочитати в нашій статті.[39] На рисунку 2.10 показані різні рівні графічних налаштувань.

Рисунок 2.10 – Приклад графіки у іграх

2.4 Розробка метода для зменшення навантаження на апаратні ресурси

Для розробки метода потрібно проаналізувати основні фактори, які впливають на продуктивність компонентів. Головними об’єктами для розгляду були взяті процесор та відеокарта. Для зменшення навантаження на процесор потрібно зменшити кількість об’єктів, які потребують обчислювальну

потужність. Це об’єкти, які мають фізичні властивості для реалістичної поведінки. Говорячи про ігрову фізику, ми маємо на увазі фізику твердого тіла (rigid body physics, RBP). Вона описує та відтворює фізичні закони, які застосовуються до твердих мас речовини. Фізика м'якого тіла (soft body physics, SBP) описує об'єкти, що деформуються. Вона використовується рідше і сильно урізана у іграх через величезну кількість обчислень.

М'які тіла – це одяг, волосся, скупчення частинок типу диму чи туману. Точки твердого тіла завжди залишаються на тому самому відстані один від одного. А м'яке тіло може деформуватися і рухатися так, що відстань між його точками змінюватиметься.[40]

Зменшити використання фізика м’якого тіла, щоб зменшити навантаження на процесор для зменшення потужності виділеною процесор на обчислювання фізики та підвищення продуктивності в інших функціях. На рисунку 2.11 приведено приклад з’єднання твердих тіл в єдиний об’єкт.

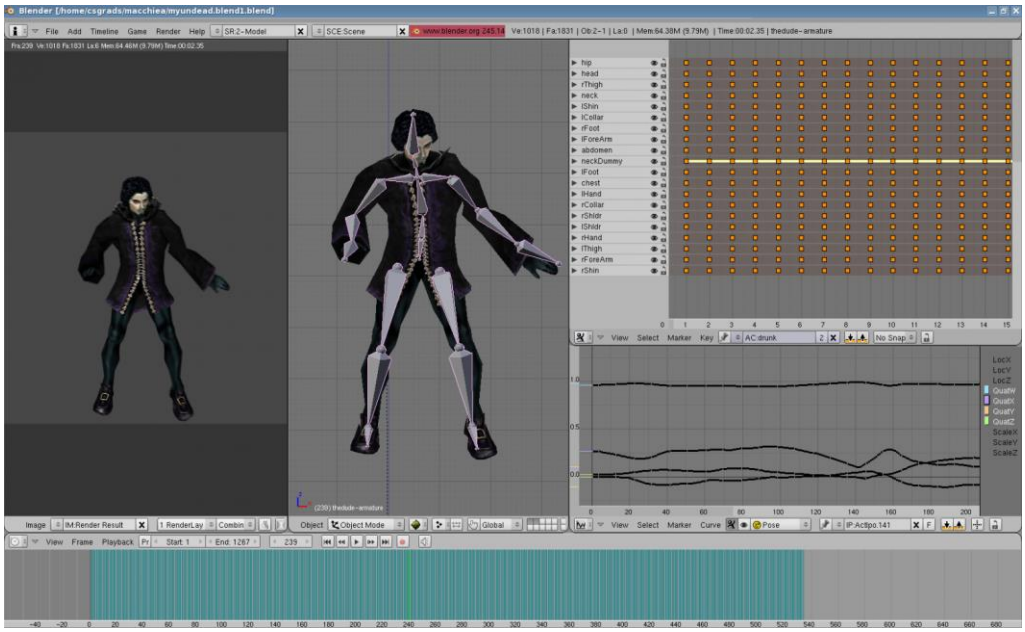

Рисунок 2.11 – Приклад з’єднання твердих тіл в єдиний об’єкт

Наступним етапом при оптимізації апаратних ресурсів є баланс між комплектуючими. Самим головним балансом має буду між процесором і відеокартою. Процесор та відеокарта виконують різні завдання. Процесор тягне двигун, скрипти, AI, фізику, він готує кадри (точніше - команди для відеокарти). Він займається фоновим провантаженням світу (з диска в ОЗУ, а звідти - у відеопам'ять). Він же зайнятий системними завданнями - на ньому запущено ОС, драйвери, служби тощо.

Відеокарта приймає команди та малює графіку. Причому завантаження відеокарти залежить і від налаштувань графіки, і від роздільної здатності, і від згладжування... А ось на завантаження процесора налаштування графіки зазвичай не впливають. Вже на цьому етапі зрозуміло, що баланс - річ дуже тендітна. Для покращення частоти кадрів у грі та зменшення навантаження на відеокарту. Потрібно правильно підібрати технології, які зможе обчислити відеокарта. На рисунку 2.12 продемонстровано, як графічні технології впливають на продуктивність та на частоту кадрів на різних рівнях.

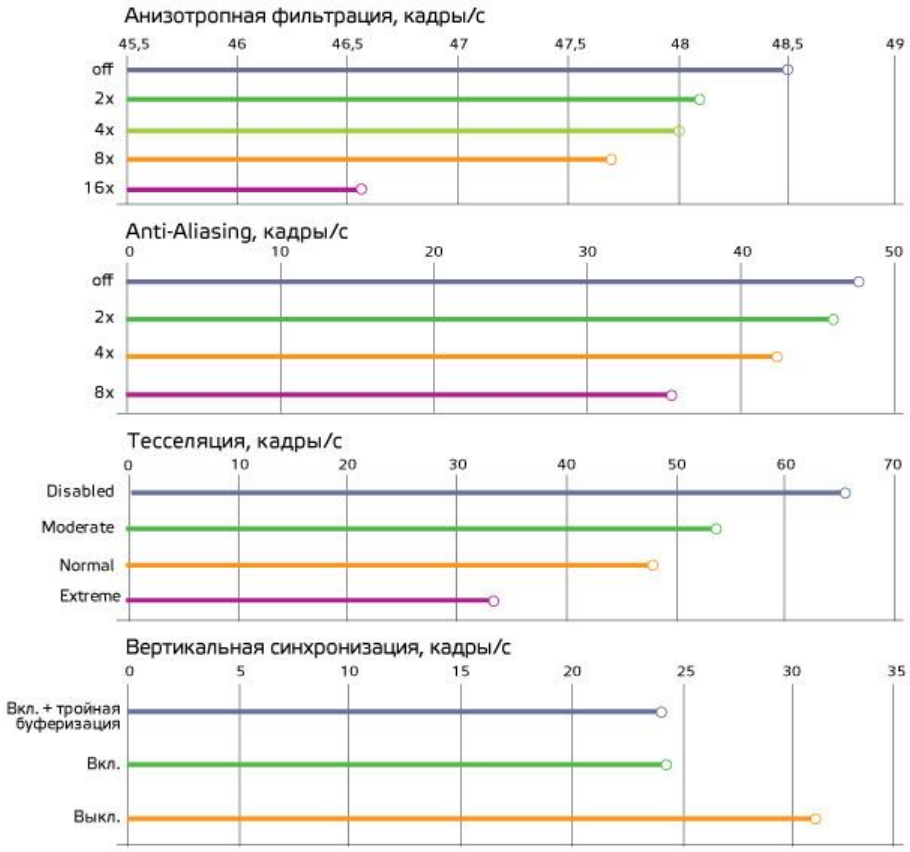

Рисунок 2.12 – Таблиця технологій та їх вплив на продуктивність

2.5 Розробка комплексного метода для прискорення та зменшення споживання пам’яті

Оперативна пам'ять (Random Access Memory, RAM — пам'ять з довільним доступом) — у більшості випадків енергозалежна частина системи комп'ютерної пам'яті, в якій під час роботи комп'ютера зберігається машинний код (програми), а також вхідні, вихідні та проміжні дані, що обробляються процесором. Оперативний пристрій (ОЗУ) — технічне пристрій, що реалізує функції оперативної пам'яті. ОЗУ може виготовлятися як окремий зовнішній модуль або розташовуватися на одному кристалі з процесором, наприклад, однокристальних ЕОМ або однокристальних мікроконтролерах.

Обмін даними між процесором і оперативної пам'яттю проводиться як безпосередньо, і через надшвидку пам'ять нульового рівня чи, за наявності апаратного кеша процесора, — через кеш.

Дані, що містяться в напівпровідниковій оперативній пам'яті, доступні і зберігаються тільки тоді, коли на модулі пам'яті подається напруга. Вимкнення живлення оперативної пам'яті, навіть короткочасне, призводить до втрати інформації, що зберігається.

Енергозберігаючі режими роботи материнської плати комп'ютера дозволяють переводити їх у режим сну, що значно скорочує рівень споживання комп'ютером електроенергії. У режимі глибокого сну живлення ОЗУ вимикається. У цьому випадку для збереження вмісту ОЗП операційна система перед вимкненням живлення записує вміст ОЗП на пристрій постійного зберігання даних (на жорсткий диск або твердотільний накопичувач). У загальному випадку ОЗУ містить програми та дані операційної системи та запущені прикладні програми користувача та дані цих програм, тому від обсягу оперативної пам'яті залежить кількість завдань, які одночасно може виконувати комп'ютер під керуванням операційної системи.[41]

ОЗУ відеокарти ми змінити не можемо, і якщо в неї не влазить вся графіка, яка потрібна для завантаження - залишок поміщається в повільнішу пам'ять - в оперативну пам'ять ПК. І звідти переміщається на запит гри у відеокарту.

Наприклад давня ОЗУ DDR3 в 1600 mhz може переміщати дані зі швидкістю 17 066 MB/s

Для порівняння – сучасний M.2 ssd робить це зі швидкістю 1 700MB/s

А вже HDD з 7200 оборотів – 150 МБ/с

Висновки — ОЗУ — в 10 разів швидше за SSD і в 100+ разів швидше за HDD.

Так що варіанти - або так само займаємося розгоном оперативної пам'яті, або переміщуємо гру з повільних накопичувачів на більш швидкі.

Якщо ви зробите файл підкачки ОЗУ - з HDD - то по суті вам глобально нічого не зміниться - дані як читалися з диска повільно - так і будуть, зміниться тільки організаційний момент.

Якщо у вас є SSD - то можна зробити псевдо збільшення ОЗУ через файл підкачки.

Додаємо підкачку за розрахунком що грі потрібно 16 гігабайт ОЗУ: з 16 гігабайт віднімаємо скільки ОЗП у вас у відеокарті, віднімаємо скільки у вас оперативної пам'яті ПК (але залиште 2 гіга під операційну систему) і залишається те число на яке було б не погано додати файл підкачки або навіть трохи більше.

Організація ОЗУ

Так вже виходить, що місце в озу обмежене, але питання що в ній зберігається. Так наприклад ми можемо звільнити частину місця — від даних, що не використовуються, щоб туди помістити дані гри і уникнути додаткових повільних підвантажень з диска.[42] На рисунку 2.13 показано тест на швидкість читання та запису 3 типів пам’яті.

Рисунок 2.13 – Тест швидкості читання та запису на різних комплектуючих.

Шейдери – це невеликі, так би мовити, "скрипти для відеокарти". Дозволяють досить легко реалізувати такі різні спецефекти та ефекти. Бувають піксельними (працюють із зображеннями - тобто або з екраном повністю, або з текстурами) і вершинні (працюють з 3Д об'єктами).

Наприклад, за допомогою піксельних шейдерів реалізуються такі ефекти, як 3Д-текстури (бамп), паралакс-тектури, промені сонця (sunshafts) а-ля Криза, розмиття по дальності, просто розмиття при русі, анімовані текстури (вода, лава,...), HDR, згладжування, тіні (за технологією ShadowMaps) та дофіга всього такого.. .

Вершинними шейдерами роблять анімацію трави, персонажів, дерев, створюють хвилі на воді (типу об'ємні) ну і т.д.

Чим складніший (якісніший, сучасніший) ефект - тим більше на нього потрібно команд у коді шейдера. Але шейдери різних версій (1.1 – 5.0) підтримують різну кількість команд: чим вища версія – тим більше команд можна використовувати. Тому деякі технології НЕМОЖЛИВО реалізувати на молодших шейдерах.[43] На рисунку 2.14 приведено приклад шейдеру в Unity 3D.

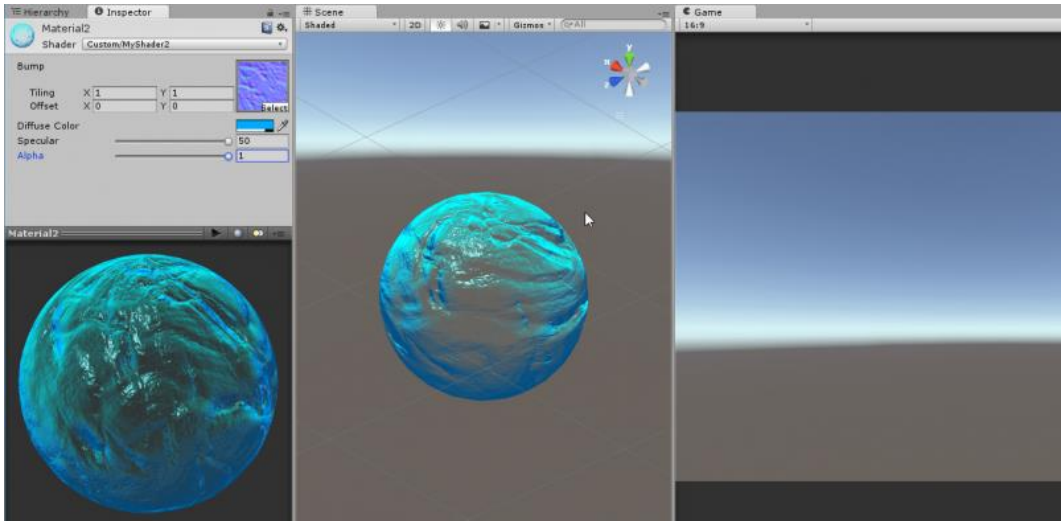

Рисунок 2.14 – Шейдери в Unity3D

Параметр у деяких додатках знижує навантаження ЦП шляхом збереження скомпільованих шейдерів на диску. Використовується лише при запуску гри на першій побудові шейдерів. Раніше зібраний шейдер буде просто вилучений з дискового кеша.

Кешування може зменшити або зовсім прибрати гальма в іграх за необхідності наступної побудови шейдерів. Кеш записуватиметься лише кілька разів, у подальшому використанні він лише зчитується.[44]

Дальність промальовування – це програмна функція, яка встановлює обмеження промальовування карти та текстур у бік яких була направлена камера. Із збільшенням цього параметру автоматично збільшується навантаження на всі комплектуючі комп’ютера. На рисунку 2.15 приведено приклад порівняння дальності та деталізації промалювання при різних налаштуваннях.

Рисунок 2.15 – Порівняння дальності та деталізації промалювання

Оптимізація коду

Оптимізацію можна виконувати на будь-якій стадії генерації коду, починаючи від завершення синтаксичного аналізу і до останнього етапу, коли породжується код результуючої програми. Якщо компілятор використовує кілька різних форм внутрішнього подання програми, кожна з них може бути піддана оптимізації, причому різні форми внутрішнього подання орієнтовані різні методи оптимізації. Таким чином, оптимізація компілятора може виконуватися кілька разів на етапі генерації коду.

Принципово розрізняються два основні види оптимізуючих перетворень:

- перетворення вихідної програми (у формі її внутрішнього подання в компіляторі), що не залежать від результуючої об'єктної мови;

- Перетворення результуючої об'єктної програми.

Перший вид перетворень залежить від архітектури цільової обчислювальної системи, де виконуватиметься результуюча програма. Зазвичай він заснований на виконанні добре відомих та обґрунтованих математичних та логічних перетворень, що виробляються над внутрішнім уявленням програми.

Другий вид перетворень може залежати як від властивостей об'єктної мови, а й від архітектури обчислювальної системи, де виконуватиметься результуюча програма. Так, наприклад, при оптимізації може враховуватися обсяг кеш-пам'яті та методи організації конвеєрних операцій центрального процесора. Найчастіше ці перетворення сильно залежить від реалізації компілятора і є «ноу-хау» виробників компілятора. Саме цей тип оптимізуючих перетворень дозволяє суттєво підвищити ефективність результуючого коду.

У сучасних компіляторів існують можливості вибору як загального критерію оптимізації, а й окремих методів, які використовуватимуться під час виконання оптимізації.

Методи перетворення програми залежить від типів синтаксичних конструкцій вихідної мови. Теоретично розроблено методи оптимізації для багатьох типових конструкцій мов програмування.

Оптимізація може виконуватися для таких типових синтаксичних конструкцій:

лінійних ділянок програми;

логічних виразів;

викликів процедур та функцій;

інших конструкцій вхідної мови

У всіх випадках можуть використовуватись як машинно-залежні, так і машинно-незалежні методи оптимізації.[45]

2.6 Висновок

Для покращення праці двигуна використовується багато методів по оптимізації двигуна та самої гри. Головними цілями при покращені праці є не погіршити, те що є, бо це дуже легко зробити. Багато методів існує для оптимізації, які відрізняються один від одного на різних рівнях розробки гри, або є методи, які дуже схожі між собою. Оптимізація на різних рівнях відрізняється так, як на рівні графіки можливо зменшити деталізацію об’єктів або зменшення навантаження на відеокарту для більшого та швидшого промалювання об’єктів на карті або виділеної області. Процесор відповідає за відправку команд на відеокарту та обчислення фізики ігрового двигуна. Щоб його розвантажити потрібно, або зменшити потужність на обчислення, або переправити деякі операції на інші комплектуючі комп’ютера.

Щоб проаналізувати комплектуючі комп’ютера потрібна використовувати спеціальні програми, у яких головна функція – це збір даних з комплектуючих(температуру, використання пам’яті, використання частот і тд). Ці програми дуже допомагають розробникам зрозуміти, де в них є критичні моменти, які потрібно буде в майбутнім покращити, або зменшити навантаження.

Швидкість завантаження текстур на карту грає велику роль у сприйняті гри та швидкодію. Для збільшення швидкості роботи гри використовується кешування шейдерів. Ця функція допомагає трохи розвантажити відеокарту та передати частину операцій на жорсткий диск та оперативну пам’ять.

Розробка комплексного методу по оптимізації допоможе зменшити навантаження на усі комплектуючи з мінімальними втратами у апаратних ресурсах та невеликими правками у коді. Основною цілю цього метода зменшення використовування оперативної пам’яті та зменшення навантаження на відеокарту за допомогою алгоритмів двигуна та шаблонів у програмуванні.

3. ТЕСТУВАННЯ КОМПЛЕКСНОГО МЕТОДУ

3.1 Побудова різних конфігурацій для тестування ігрових двигунів

Для тестування даного методу було прийнято рішення зібрати 3 різних зборки з різним рівнем потужності комплектуючих, які будуть вимірюватися в терафлопсах.

Розрахунок пікової продуктивності комп'ютера виконується за такою формулою:

1 терафлопс = F x N x I x 106. Де:

F - тактова частота процесора в MHz,

N – число процесорів (ядер),

I – кількість оброблюваних інструкцій за такт,

106 = 1000000.

Реальна продуктивність комп'ютера буде, знаходиться в районі 70-80% від пікової Терафлопс.[46]

В терафлопсах ми будемо вимірювати процесор та відеокарту, для носіїв інформації буде взята лише швидкість запису та читання. Усі технічні характеристики буду приведені для кожної збірки в окремому розділі с показником графіка без оптимізаційного метода. Також буде приведені мінімальні технічні характеристики для ігрового двигуна та рекомендовані характеристики. Також для тестування у перші рази будуть приведені середні або автоматичні налаштування графіки, щоб користувач не задумувався при перших запусках гри, але для тестових збірок та для тестів це буде найкращою можливістю перевірити оптимізацію від розробників, яка була на початковому етапі розробки та на наступних етапах.

Тестування усіх збірок буде проведено на версії гри 0.12.11.5.14896, яка була доступна для оновлення від 21.10.2021. Це останнє оновлення гри на момент тестування збірок для більш реальних цифр.

Мінімальні технічні характеристики:

Операційна система: Windows 7/8/10 64-bit

Процесор: Intel Core 2 Duo, i3 2.4 GHz ил AMD Athlon, Phenom II 2.6 GHz

Відеокарта: DX9 с 1 GB VRAM

Оперативна пам’ять: 6 GB

Місце на жорсткому диску: 8 GB

Рекомендовані технічні характеристик:

Операційна система: Windows 7/8/10 64-bit

Процесор: Intel Core i5, i7 3.2 GHz или AMD FX, Athlon 3.6 GHz

Відеокарта: DX11 с 2 GB VRAM чи більше VRAM

Оперативна пам’ять: 8 GB чи більше

Місце на жорсткому диску: 12 GB

Мінімальні технічні характеристики не дають гарантії, що гра працюватиме у стабільних 30 кадрах на секунду, а гарантовано запуститись на такій збірці. Рекомендовані вже дають гарантія, що гра піде на мінімальних налаштуваннях і видаватиме стабільні 30 кадрів і більше.

Так само для стабільної частоти кадрів потрібно домогтися балансу в комплектуючих, але в збірках, які ми тестуватимемо, буде не збалансована в якій відеокарта буде потужнішою за процесор і він не зможе розкрити всю потужність відеокарти. Для того щоб максимально протестувати метод і отримати приблизні данні для майбутнього корегування та можливостей у майбутньому.

У тестуванні будуть використовуватися 1 ноутбук та 2 персональних комп’ютера, які були зібрані у різні рокі, щоб протестувати метод, як на нових комплектуючих та на комплектуючих минулого покоління. У всіх в них будуть однакові налаштування, а саме:

Розширення екрану: 1920x1080

Співвідношення сторін: 16x9

Частота оновлення екрану: 60 гц

Режим екрану: Повно розмірний

Вертикальна синхронізація: Вимкнена

Ліміт кадрів: 120 к/с

Використання тільки фізичних ядр процесора: Увімкнена

Автоматична очистка оперативної пам’яті: Увімкнена

Файл підкачки: 3024 мб

Усі наступні графічні налаштування будуть підбиратися під збірки, щоб отримати найкращий результати при тестуванні збірок.

3.2 Тестування першої конфігурації

Перша конфігурація буде зібрана на основі процесора від компанії AMD та серії процесорів FX, було обрано цей процесор у 2015 році. Основною проблемою цієї збірки – це незбалансовані компоненти. Також процесор потребую потужний блок живлення для повної потужності процесора, було обрано блок живлення на 500 ват.

Технічні характеристики процесора AMD FX-8320:

Число ядер ЦП: 8

Число потоків: 8

Макс. Частота: До 4.0GHz

Базова частота: 3.5GHz

Об'єм кеш-пам'яті першого рівня: 384КБ

Об'єм кеш-пам'яті другого рівня: 8MB

Об'єм кеш-пам'яті третього рівня: 8MB

Величина відведення теплової потужності за промовчанням величина відведення теплової потужності: 125W

Техпроцес: 32nm SOI

Потужність процесора дорівнює приблизно 128 Гфлоп/с

Відеокарта була обрана від компанії NVIDIA із 10 серії GTX, а саме GTX 1060 6GB, технічні характеристики:

Ядро: GP160

Техпроцес: 16нм

Транзисторів: 4400 млн

Частота роботи ядра: 1506-1708 МГц

Частота роботи шейдерних блоків: 1506-1708 МГц

Шейдерних блоків: 1280

Частота роботи пам’яті: 8000 МГц

Шина пам’яті: 192-bit GDDR5

Об’єм пам’яті: 6144

Потужність відеокарти дорівнює приблизно 4,4 Тфлоп/с.

Оперативна пам’ять була обрана від компанії Kingston DDR3-1333 4096MB

Об’єм пам’яті: 8 гб

Частота пам’яті: 1333 МГц

Ефективна пропускна спосібність: 10600 МБ/с

Жорсткий диск було обрано від компанії Western Digital серії Blue на 1 ТБ, технічні характеристики:

Об’єм пам’яті: 1 ТБ

Швидкість читання: 163 МБ/с

Швидкість запису: 158 МБ/с

Ігровий двигун запустився та обробив більшу частину графічних даних на налаштуванні мінімальних характеристик. 40 кадрів на секунду з падінням до 24 є некомфортними для гри. Процесор не може передати більшість інструкцій із-за малої частоти оперативної пам’яті та малого об’єму кеш пам’яті у процесорі. На рисунку 3.1 показані результати тестування першої конфігурації.

Рисунок 3.1 – Показники частоти кадрів та завантаженості процесора

3.3 Тестування другої конфігурації

Друга конфігурація – це ноутбук від компанії Lenovo серії Legion. Компоненти вже більше збалансовані між собою та

потужність більша, але майже у всіх ноутбуків є проблема с охолодженням основних деталей, тому при великих навантаженнях і при великих температурах більше 90 градусов по Цельсію відбувається троттлінг. Під час троттлінгу процесор та відеокарта зменшують навантаження або працюють на великих температура із-за чого йде перегрівання основного кристалу та зменшення потужності.

AMD Ryzen 5 4600H технічні характеристики:

Кількість ядер CPU: 6

Кількість потоків: 12

Макс. Частота: До 4.0GHz

Базова частота: 3.0GHz

Об'єм кеш-пам'яті L2: 3MB

Об'єм кеш-пам'яті третього рівня: 8MB

TDP номінальний / TDP: 45W

AMD Configurable TDP (cTDP): 35-54W

Техпроцес: TSMC 7nm FinFET

Термопакет: FP6

Макс. Температура: 105°C

Потужність процесора дорівнює приблизно 392,3 Гфлоп/с

Відеокарта NVIDIA RTX 2060 на 6 GB, технічні характеристики:

Ядро: TU106

Техпроцес: 12нм FinFET

Транзисторів: 10.8 млрд

Частота роботи ядра: 1365-1680 МГц

Частота роботи шейдерних блоків: 1365-1680 МГц

Кількість універсальних процесорів: 1920

Кількість текстурних блоків: 120

Кількість блоків блендинга: 48

Частота роботи пам’яті: 14 ГГц

Шина пам’яті: 192-bit GDDR5

Об’єм пам’яті: 6144

Потужність відеокарти дорівнює приблизно 12,9 Тфлоп/с

Samsung SO-DIMM DRR-4 3200 МГц, технічні характеристики

Об’єм пам’яті: 16 ГБ

Частота пам’яті: 3200 МГц

Ефективна пропускна спосібність: 25600 МБ/с

Kingston A2000 1ТБ NVMe M2 2280, технічні характеристики:

Швидкість читання: 2200 МБ/с

Швидкість запису: 2000 МБ/с

Така конфігурація вже є більш комфортною для гри на високих налаштуваннях графіки, частота кадрів приблизна від 80 до 95, так як процесор і відеокарта збалансовані, також пропуска спосібність на запис та читання збільшились у декілька разів. Але остається проблема з троттлінгом, при довгій загрузці, показники потужності зменшуються на 15-20%. На рисунку 3.2 показані результати тестування другої конфігурації.

Рисунок 3.2 – Показники потужності

3.4 Тестування третьої конфігурації

Третя конфігурація була зібрана з рекомендацій розробника на базі процесора від компанії Intel. Розробник гарантую, що при цій зборці ігровий двигун буде видавати стабільні 60 кадрів на секунду при середньо- високих налаштуваннях графіки.

Intel Core I5-9400F, технічні характеристики:

Кількість ядер CPU: 6

Кількість потоків: 6

Макс. Частота: До 4.1GHz

Базова частота: 2.9GHz

Об'єм кеш-пам'яті L2: 3MB

Об'єм кеш-пам'яті третього рівня: 9MB

TDP номінальний / TDP: 65W

Техпроцес: 14nm

Макс. Температура: 100°C

Потужність процесора приблизно дорівнює 460,8 Гфлоп/с.

Відеокарта GTX 1660 на 6 Гб, технічні характеристики:

Ядро: TU116-300

Техпроцес: 12нм FinFET

Транзисторів: 6,6 млрд

Частота роботи ядра: 1530-1785 МГц

Частота роботи шейдерних блоків: 1530-1785 МГц

Частота роботи пам’яті: 4000 МГц

Кількість універсальних процесорів: 1408

Кількість текстурних блоків: 88

Кількість блоків блендингу: 48

Шина пам’яті: 192-bit GDDR5

Об’єм пам’яті: 6144

Потужність відеокарти приблизно дорівнює 10 Тфлоп/с

Samsung SO-DIMM DRR-4 3200 МГц, технічні характеристики

Об’єм пам’яті: 16 ГБ

Частота пам’яті: 3200 МГц

Ефективна пропускна спосібність: 25600 МБ/с

Жорсткий диск було обрано від компанії Western Digital серії Blue на 1 ТБ, технічні характеристики:

Об’єм пам’яті: 1 ТБ

Швидкість читання: 163 МБ/с

Швидкість запису: 158 МБ/с

На рисунку 3.3 показані результати тестування третьої конфігурації.

Рисунок 3.3 – Показники потужності

Третю конфігурацію можливо назвати золотою серединою серед інших тестових. Процесор з відеокартою збалансовані, але жорсткий диск дуже повільний із-за цього завантаження на локацію відбувається приблизно від 5-6 хвилин, що є жахливим результатом. На самій локації при середніх графічних налаштуваннях ігровий двигун видає оброблює інформацію в стабільні 60 кадрів, інколи в 70 кадрів.

3.5 Комплексне порівняння різних конфігурацій

Усі тестові збірки тестувалися на протязі 1 місяця, тому цифри які будуть приведені далі є середніми за місяць. Тестування також проводилося на протязі від 3 до 8 годин, щоб точно знати в який період конфігурації втрачають свою потужність або навпаки набирають оберти для більшої частоти кадрів. У конфігурацій майже не було однакових компонентів або характеристик по інтернету або напрузі, тому тестування можливо вважати незалежним.

На першому тижні тестування збірок було потрібно підготувати та налаштувати їх для обробки даних та тестування на максимальній потужності та на мінімальній. Після проведення технічного обслуговування були встановлені програми для отримання частоти кадрів, графіків навантаження та цифр по частоті процесора, відеокарти, оперативної пам’яті для цього було обрана програма FPS Monitor. На рисунку 3.4 показані результати тестування процесорів в період першого тижня.

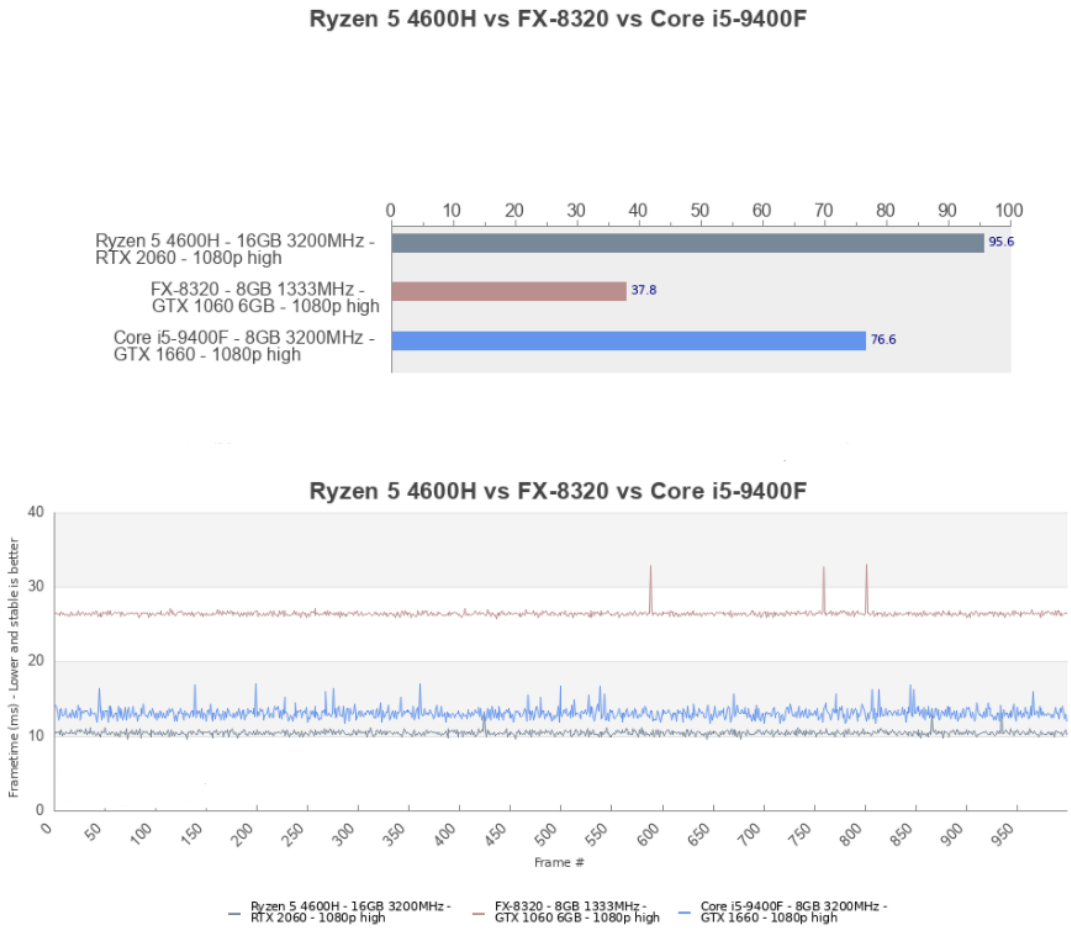

Рисунок 3.4 – Показники потужності конфігурацій

Тестування на першому тижні проводилось у 3 етапи(Налаштування графіки, тестування усіх локацій, аналіз). На

першому етапі на всі три збірки були установлені мінімальна графіка, щоб точно перевірити на запуск та комфортність усіх збірок на користування. На другому етапі кожна збірка тестувалася окрема та на всіх локаціях, які були в грі. В грі були локації закритого типу, гібридного типу, відкритого типу. На кожній локації також є різна кількість об’єктів для обробки та взаємодії з ними. Перша збірка показала результат від 60 до 80 кадрів на карті закритого типу, на карті відкритого типу були просідання кадрів до 20, що було не дуже комфортним для тестувальника. Друга збірка мала кращі результати на карті закритого типу було від 100 до 120 кадрів, на карті відкритого типу були стабільні 90 кадрів. Третя збірка на карті заритого типу показала від 80 до 95 кадрів, на карту відкритого типу завантажувалася більше 6 хвилин, як і перша збірка, але частота кадрів була вища від 65 до 80 кадрів. Третій етап тестування був найважливішим так, як було потрібно зібрати інформацію за тиждень та проаналізувати її за наступними параметрами(температура, частота, стабільність та навантаження на компоненти). Тільки у першої збірки була трохи вища температура стабільно 80 градусів, але завантаженість відеокарти максимальною була приблизно до 40% в той момент, як процесор був навантажений на 80-83%. Інші збірки показали кращий результат ніж перша, в них максимальна температура була до 75 градусів та завантаженість процесора і відеокарти відрізнялися на 5-8 %. Після тестування першого тижня було прийнято рішення, що на великих локаціях потрібно зменшити дальність промалюваня у першої збірки в половину, щоб зменшити навантаження на відеокарту з процесором для тих об’єктів, які гравець не бачить перед собою та підготувати приблизну зону, яка потрібна гравцеві для комфортної гри. На рисунку 3.5 показаний

діапазон точки зору, який допоможе гравцеві комфортно грати та зменшити навантаження на апаратні ресурси завдяки зменшеному діапазону провалювання. В теорії ця частина методу повинна підняти продуктивність в середньому від 3 до 7%. На рисунку 3.5 показано приклад точки зору для більшої оптимізації.

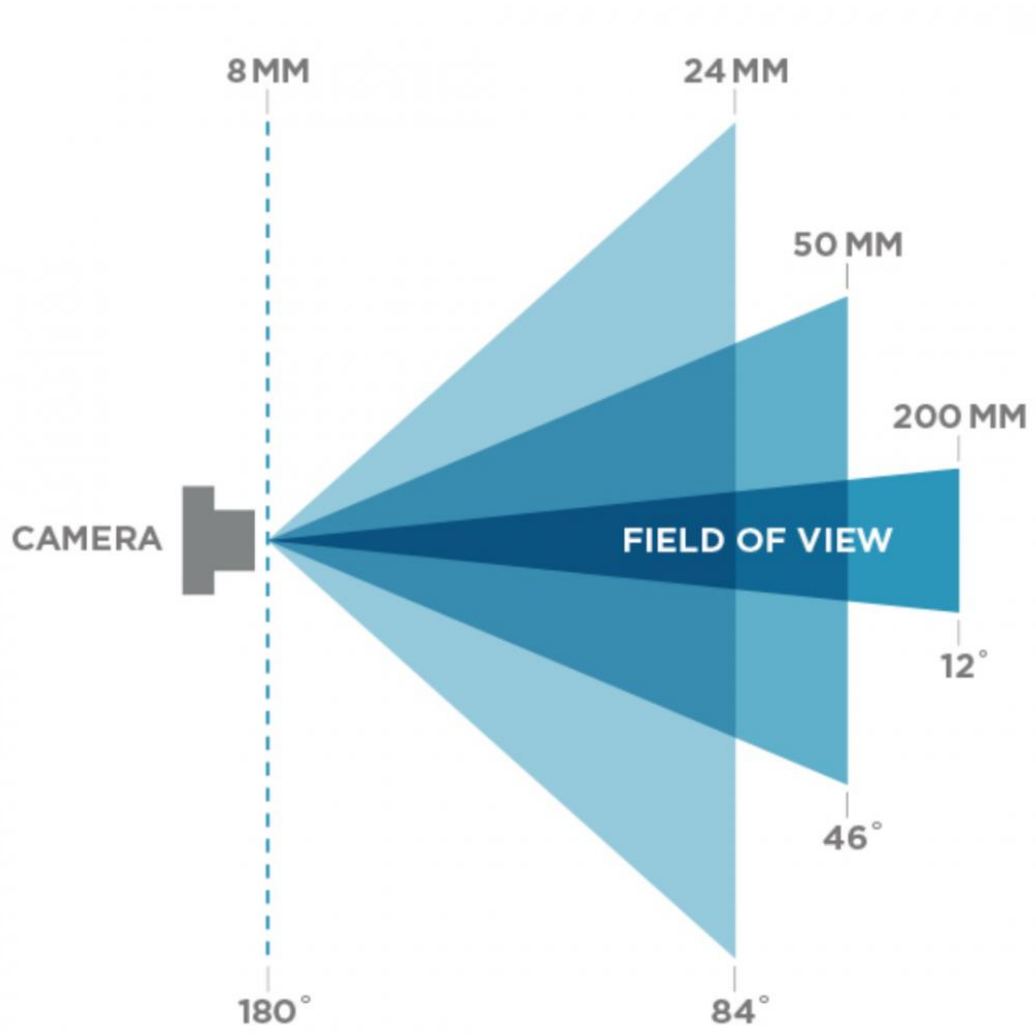

Рисунок 3.5 – Точка зору для оптимізації

На другому тижні тестування графіку підняли до середніх в середньому частота кадрів зменшилися на 20-30% та час для

завантаження на локацію збільшилось приблизно до 10%. Температура збільшилась на 15-17%, але у другої збірки температура піднялася на 20-25% із-за системи охолодження у ноутбуках, тротлінгу під час тестування не було помічено. На рисунку 3.6 показані середні показники потужності за другий тиждень. Під час тестування було виявлено, що ігровий двигун після 3 – 4 годин починає переповнювати кеш комп'ютера і оперативна пам'ять починає переповнюватися і були помічені мікрофризи, які дуже сильно починають заважати. Потрібно було вимикати гру та ігровий двигун, щоб очистити кеш, було запропоновано щоб ігровий двигун автоматично чистив кеш. У оновлені 0.12 була така функція, які допомагала ігровому двигуну. На рисунку 3.6 показані результати тестування конфігурації на другому тижні.

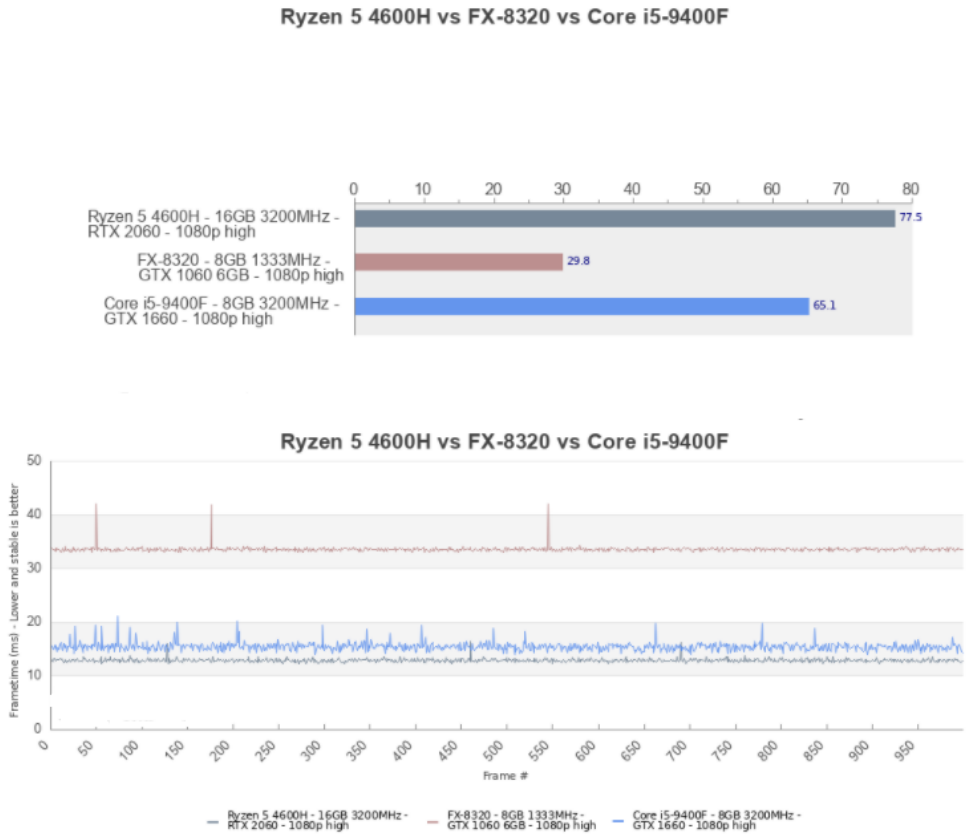

Рисунок 3.6 – Показники потужності на другому тижні

На третьому тижні тестування графіку було підвищено до високих. Перша збірка майже не запускала ігровий двигун на локації, у другої та третьої збірках частота кадрів зменшилась ще на 20-30%, температура піднялась на 15-20% у третій конфігурації. Друга конфігурація мала проблеми з тротлінгом ця проблема проявлялась після 4 годин безперервного тестування, температура збільшилась приблизно до 30% і потужність процесора з відеокартой зменшилась приблизно на 15%, частота кадрів зменшилась на 10-15 кадрів. На рисунку 3.7 із графіка частоти кадрів було прибрано першу збірку із-за того, що майже не можливо було зняти показники із-за високого навантаження на процесор і система не могла оброблювати дані із ігрового двигуна та показувати ресурси, які використовувалися під час гри. На рисунку 3.7 показані результати тестування на третьому тижні.

Рисунок 3.7 – Показники потужності на третій тиждень

За звичай такі проблеми бувають із-за неправильних конфігурацій комп'ютера в нашому випадку із-за малого об'єму оперативної пам'яті у першій збірці та малої частоти 1333 МГц не вистачає для обробки даних. Мінімальні системні вимогу не мають таку детальну інформації, але під час тестування було визначено, що рекомендована частота для пам'яті є 2000 МГц і об'єм пам'яті повинен бути не менше 12 гігабайтів.

Для того щоб зменшити об'єм використаної пам'яті потрібно у коді знайти місця в яких двигун використовую більше ніж потрібна, та змінити код під шаблон легковик, знайти схожі характеристики та відокремити їх в єдиний об'єкт або в декілька об'єктів. Параметри, які на протязі довго часу або зовсім не змінюються їх краще визначити, як константа. Після змін у коді

використання оперативної пам’яті приблизно зменшиться на 10-15%.

Під час тестування було замічені об’єкти, які знаходяться на великій відстані від гравця мають дуже детальне провалювання, яке гравець не зможе замітити приближення. Якщо використати алгоритм для ігрового двигуна під назвою Level of Detail, якщо зменшити рівень деталізації об’єктів, які находяться на відстані 100 ігрових метрів, то завантаженість на відеокарту зменшиться приблизно на 1% на 100 об’єктах. Для зменшення навантаження на процесор можливо передати частину об’єктів, які потрібно промалювати та присвоїти параметри відеокарті, або об’єкти, які статичними записати у пам’яті при увімкнені двигуна. Можливо також для кращої продуктивності ввести функція завантаження шрейдерів при першому запуску двигуна, який за пише параметри об’єктів на фізичний носій і буди звертатися, коли потрібні об’єкти. Краще збільшити об’єм на фізичному носії на 10-15 гігабайтів, а ніж раз за разом промальовувати та присвоювати данні статичним об’єктам.

Також основною проблему при завантаженні локації є під вантаження усіх об’єктів на локацію та присвоєння ним параметрів. Щоб зменшити час завантаження на локацію було прийнято рішення об додаванні функції поступового промальовування в області, якій знаходиться камера.

Для уникнення ефекту пляшкова шийка потрібно розмістити більш детальні вимоги до апаратних ресурсів при запуску ігрового двигуна та прикласти від 3-5 конфігурацій різні за ціною та характеристиками. Ці конфігурації повинні бути протестовані при максимальних навантаження та довгий час, як проводилося тестування у магістерській роботі.

При тестуванні конфігурацій були виявлені проблеми, що тестування у великій період часу не дає точну інформацію, а лише середню інформацію та розбіжність в результатах тестування на 5-8% із-за зовнішніх факторів та людського фактору. Було запропоновано протестувати усі 3 конфігурацію на протязі 24 годин, але це виявилось майже не можливо із-за людського фактору. Також тестування показало, як довго ігровий двигун може без зупинки при навантаженні компонентів на 80-90% працювати без мікрофризів та інших технічних багів. Саме під час тестування було замічена проблема тротлінгу та через який період часу вона з’являться і скільки часу потрібно на те щоб відновити потужність, точніше скільки часу потрібно на зниження температури для відновлення потужності на початок тестування.

ВИСНОВКИ

В результаті дослідження в роботі було розроблено комплексний метод оптимізації програмно-апаратних ресурсів та зменшення використання оперативної пам’яті за допомогою алгоритмів двигуна та шаблонів проектування програмного коду. Новизна та розширення у сфері оптимізації ігрового двигуна Unity 3D, котрий був модифікований та налаштований під роботу з грою Escape from Tarkov. Було переглянуто багато алгоритмів, які підходили під головне завдання та вибір найбільш вдалих.

У першому розділі було проаналізовано роботу ігрових двигунів та види ігрових двигунів. Головні відзнаки двигунів один від одного, налаштування їх під окремий проект та найбільші плюси та мінуси використання такого двигуна. Основні принципи праці двигуна та функції як вони взаємодіють з комплектуючими. Можливість оптимізувати ігровий двигун за допомогою алгоритмів та шаблонів проектування програмного коду.

Другий розділ потрібен для розуміння головної думки, що до оптимізації та взаємодію компонентів між собою. У першій частині більш детально розглядаються обрані алгоритми для оптимізації та аналіз їх використання у різних проектах для отримання найкращого результату при використані алгоритму. Розглядається також взаємодія компонентів, щоб зрозуміти критичні місця при оптимізації та використані алгоритмів. Багато проблем було із-за незбалансованих комплектуючих, які дуже часто створювали ситуації пляшкової шийки. У цьому випадку дуже складно відновити баланс за допомогою алгоритмів та програмного коду. В кінці розділу було розроблено комплексний метод, який повинен підвищити потужність комплектуючих на 15-25% та зменшити використання оперативної пам’яті на 22-26%.

Останній розділ був присвячений тестування 3 збірок, які відповідали заданим характеристикам. Перша збірка повинна була бути незбалансованою для кращого дослідження та тестування метода. Друга збірка була збалансована але мала проблеми з температурою на комплектуючих та зменшенням потужності. Третя збірка вже мала проблему довгого завантаження на локацію та потрібна була для тестування в сфері зменшення завантаження об’єктів та графіки на певну територію. Після тестування 3 збірок був проведений аналіз потужності в якому виявлялись основні проблеми та недоліки використання комплексного методу. Також були запропоновані ідеї для майбутньої оптимізації методу та покращення тестування завдяки знаходженню основних проблем.

СПИСОК ЛІТЕРАТУРИ

1. Ігровий рушій [Електронний ресурс]. - Режим доступу:

http://uk.wikipedia.org/wiki/%D0%86%D0%B3%D1%80%D0%BE%D0%B2%D0%B8%D0%B9_%D1%80%D1%83%D1%88%D1%96%D0%B9

2. Что такое игровой движок? [Електронний ресурс]. - Режим доступу:

https://funduk.ua/technoblog/gaming-raznoe/chto-takoe-igrovoy-dvizhok/

3. Хокінг, Джозеф. Unity - у дії. Мультиплатформна розробка на C#

4. Memory in WebGL [Електронний ресурс]. - Режим доступу:

https://docs.unity3d.com/Manual/webgl-memory.html

5. Verge3D versus Unity WebGL [Електронний ресурс]. - Режим доступу:

https://www.soft8soft.com/verge3d-versus-unity-webgl/

6. Unity (игровой движок) [Електронний ресурс]. - Режим доступу:

https://ru.wikipedia.org/wiki/Unity_(%D0%B8%D0%B3%D1%80%D0%BE%D0%B2%D0%BE%D0%B9_%D0%B4%D0%B2%D0%B8%D0%B6%D0%BE%D0%BA)

7. Сергій Бабаєв. Розробники Contract Wars з Петербурга анонсували новий шутер Escape from Tarkov з квестами та глобальним сюжетом. VC.ru (7 листопада 2015 року).

8. Escape from Tarkov є Unity до наступного рівня. 80 рівень.

9. Анастасія Бунчук. Escape from Tarkov - хардкорна MMO російською мовою. Stopgame.ru.

10. Денис Князєв. ЗБТ російського шутера Escape from Tarkov розпочнеться у 2016 році. Ігроманія.

11. Йоханан Привалов. Попередні вимоги до системи Escape from Tarkov. GameGuru.

12. У Мережі з'явилося відео нового шутера про війну ООН та Росії. Bigmir.ru

13. Bill Lavoy. Escape from Tarkov: Strategic Preview. USGamer

14. Олена Мила. У грі Escape from Tarkov показали динамічну зміну дня та ночі. Game bomb.

15. Escape from Tarkov – Эксклюзивное интервью с разработчиками из Battlestate Games [Електронний ресурс]. - Режим доступу:

https://mmo13.ru/news/post-739

16. Escape from Tarkov – русский The Division от разработчиков Contract Wars [Електронний ресурс]. - Режим доступу:

https://coop-land.ru/news/9400-escape-from-tarkov-russkiy-the-division-ot-razrabotchikov-contract-wars.html

17. Escape from Tarkov – Official Announcement Trailer [Електронний ресурс]. - Режим доступу:

https://www.youtube.com/watch?v=IPlnHe9l34E&ab_channel=Battlestate

18. Геймплей Escape from Tarkov показали в новом трейлере [Електронний ресурс]. - Режим доступу:

https://www.igromania.ru/news/57618/Geympley_Escape_from_Tarkov_pokazali_v_novom_treylere.html

19. Escape from Tarkov – Action Gameplay Trailer [Електронний ресурс]. - Режим доступу:

https://www.youtube.com/watch?v=a70_QmnjhCU&ab_channel=Battlestate

20. Закрытое альфа-тестирование Escape from Tarkov пройдет в августе [Електронний ресурс]. - Режим доступу:

https://www.igromania.ru/news/61059/Zakrytoe_alfa-testirovanie_Escape_from_Tarkov_proydet_v_avguste.html

21. Escape from Tarkov готовится к расширенной альфа-версии [Електронний ресурс]. - Режим доступу:

https://www.igromania.ru/news/64660/Escape_from_Tarkov_gotovitsya_k_rasshirennoy_alfa-versii.html

22. Development Diary: Preparing for the CBT [Електронний ресурс]. - Режим доступу:

https://www.escapefromtarkov.com/news/id/65

23. Закрытое бета-тестирование Escape from Tarkov пройдет в июле [Електронний ресурс]. - Режим доступу:

https://www.igromania.ru/news/68530/Zakrytoe_beta-testirovanie_Escape_from_Tarkov_nachnetsya_v_iyule.html

24. Стартовало ЗБТ Escape from Tarkov – только для тех, кто предзаказал игру [Електронний ресурс]. - Режим доступу:

https://coop-land.ru/news/12578-startovalo-zbt-escape-from-tarkov-tolko-dlya-teh-kto-predzakazal-igru.html

25. Вышла первая книга по Escape from Tarkov [Електронний ресурс]. - Режим доступу:

https://www.igromania.ru/news/73403/Vyshla_pervaya_kniga_po_Escape_from_Tarkov.html

26. Escape from Tarkov готовится покорить и консоли [Електронний ресурс]. - Режим доступу:

https://www.igromania.ru/news/94729/Escape_from_Tarkov_gotovitsya_pokorit_i_konsoli.html

27. Escape from Tarkov [Електронний ресурс]. - Режим доступу:

https://ru.wikipedia.org/wiki/Escape_from_Tarkov#cite_note-PCGamingShow2020-33

28. Escape from Tarkov: система кастомизации оружия [Електронний ресурс]. - Режим доступу:

https://dtf.ru/gamedev/2667-escape-from-tarkov-sistema-kastomizacii-oruzhiya

29. Механики работы оружия Escape from Tarkov [Електронний ресурс]. - Режим доступу:

https://www.youtube.com/watch?v=EdxxxWF0Yf8&ab_channel=NoX-GAME

30. 10 проблем Escape from Tarkov, которые никто не хочет признавать [Електронний ресурс]. - Режим доступу:

https://gameinonline.com/article/10_problem_escape_tarkov_kotorye_nikto_ne_hochet_priznavat.html

31. Советы: как оптимизировать 3D- игры [Електронний ресурс]. - Режим доступу:

https://vc.ru/playgendary/120585-optimization

32. FPS Monitor [Електронний ресурс]. - Режим доступу:

http://radeon.ru/downloads/utils/fps_monitor/

33. Оптимизация кода [Електронний ресурс]. - Режим доступу:

https://pvs-studio.com/ru/blog/terms/0084/

34. «Бутылочное горлышко» и компьютер: что за термин? [Електронний ресурс]. - Режим доступу:

https://ip-calculator.ru/blog/ask/butylochnoe-gorlyshko-i-kompyuter-chto-za-termin/

35. Оптимизация 3-D моделей для игровой сцены [Електронний ресурс]. - Режим доступу:

https://habr.com/ru/company/plarium/blog/484792/

36. Уровень деталей (LOD) (Level of Detail) оптимизация [Електронний ресурс]. - Режим доступу:

http://jmonkeyengine.ru/wiki/jme3/advanced/level_of_detail

37. Батчинг вызовов в отрисовки [Електронний ресурс]. - Режим доступу:

https://sites.google.com/site/rusewyl/grafika/optimizacia-proizvoditelnosti-grafiki/batcing-vyzovov-otrisovki-draw-call-batching

38. Основы оптимизации кода игр [Електронний ресурс]. - Режим доступу:

https://habr.com/ru/post/358176/

39. За что отвечает видеокарта в играх [Електронний ресурс]. - Режим доступу:

https://rg-gaming.ru/kompjutery/za-chto-otvechaet-videokarta-v-igrah

40. Мотор! или Что такое игровая физика [Електронний ресурс]. - Режим доступу: https://habr.com/ru/company/playgendary/blog/490720/

41. Оперативная память [Електронний ресурс]. - Режим доступу:

https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D0%BC%D1%8F%D1%82%D1%8C

42. Оптимизация Escape from Tarkov [Електронний ресурс]. - Режим доступу:

https://retarkov.ru/optimizatsiya-escape-from-tarkov/

43. Шейдеры. Что такое шейдеры в играх? [Електронний ресурс]. - Режим доступу:

https://otvet.mail.ru/question/52748019

44. Кеширование шейдеров NVIDIA. ВКЛ или ВЫКЛ [Електронний ресурс]. - Режим доступу:

https://windd.ru/keshirovanie-shejderov-nvidia-vkl-ili-vykl/

45. Методы оптимизации кода [Електронний ресурс]. - Режим доступу:

https://studfile.net/preview/4599410/page:38/

46. Терафлопс [Електронний ресурс]. - Режим доступу:

https://yoursputnik.ru/teraflops/