КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (12), 2020
ISSN 2663 - 4023
DOI: 10.28925/2663-4023.2021.12.6984
УДК 004.45

Нестеренко Костянтин Павлович
Студент
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна
ORCID ID: 0000-0003-3921-4324
2000kostia@gmail.com

Жураковський Богдан Юрійович
Доктор технічних наук, професор
Національний технічний університет України
"Київський політехнічний інститут імені Ігоря Сікорського", Київ, Україна
ORCID ID: 0000-0003-3990-5205
zhurakovskiybyu@tk.kpi.ua

КОНВЕРТЕР ЗОБРАЖЕНЬ НА ОСНОВІ АЛГОРИТМІВ БЛОЧНОГО СТИСНЕННЯ ТЕКСТУР DXT1, DXT3 ТА DXT5

Анотація. У даній статті проведено аналіз існуючих додатків, які реалізують алгоритми блочного стиснення текстур. На його основі запропановано найбільш оптимальний варіант технічної реалізації. Обрано та обґрунтовано набір технологій для реалізації прототипу та розроблена його архітектура на основі принципів, що забезпечують максимально розширюваність та чистоту коду. З розвитком технологій та інтеграцією комп'ютеризованих систем в усі можливі галузі діяльності людини, все частіше і частіше починає використовуватися програмне забезпечення з використанням тримірної графіки. Подібні програми вже давно відійшли від використання лише у розважальній сфері для таких задач як розробка комп'ютерних ігор або спеціальних ефектів для кінематографу. Тепер з їх допомогою лікарі можуть планувати найбільш складні операції, архітектори перевіряти розроблені плани споруд а інженери моделювати прототипи без використання жодних матеріалів. З одного боку таке стрімке зростання можна пояснити збільшення потужності компонентів для персональних комп'ютерів. Наприклад сучасні графічні процесори, які відіграють ключову роль у роботі графічного програмного забезпечення, за останні десятиліття стали в рази швидшими та в сотні разів збільшили свій об'єм пам'яті. Проте скільки б система не мала ресурсів, все ще залишається питання їх ефективного використання. Саме для вирішення цієї проблеми були створені алгоритми блочного стиснення текстур. Фактично вони дали можливість створювати ефективне програмне забезпечення тоді, коли ресурси комп'ютерів все ще були досить обмежені. А зі збільшенням ресурсів дозволили розробляти програмне забезпечення з неймовірним рівнем деталізації моделей, що й спричинило його активне провадження у такі вибагливі до точності сфери як медицина, будівництво тощо. Кінцевим результатом даної роботи є розроблений додаток з урахування сучасних потреб користувача. Під час розробки були використані найбільш сучасні технології для найбільшої швидкодії та забезпечення актуальності додатку. Також під час розробки були враховані основні переваги та недоліки вже існуючих рішень. Можливості системи були перевірені за допомогою мануального тестування на локальній машині.

Ключові слова: алгоритми блочного стиснення текстур, текстура, DXTn, DDS, Qt.

ВСТУП

Алгоритми блочного стиснення текстур, також відомі як S3 TextureCompression (далі S3TC), DXTn або BCn - це клас алгоритмів для стиснення зображень з втратами з фіксованим розміром вихідних даних. Основний напрям використання даних алгоритмів - це програмне забезпечення з використанням тривимірної комп'ютерної графіки з апаратним прискоренням.

ISSN 2663 - 4023

Подібне програмне забезпечення використовує текстури, тобто контейнери для одного або більше зображень, які накладаються на поверхню певного геометричного об’єкту, для його деталізації та візуалізації. Текстури можуть бути як двовимірними так і тривимірними.

Фактично предметна область використання DXTn не є обмеженою і залежить лише від того, в яких сферах діяльності можна застосувати програмне забезпечення для тримірної графіки та модулювання. А розвитком технологій та інтеграцією комп’ютеризованих систем у всі можливі галузі діяльності людини, ця сфера постійно розширюється. Навіть зараз вона охоплює усе від створення комп’ютерної графіки для кінематографу, медицини та будівництва.

Постановка проблеми. Усі сучасні персональні комп’ютери в тому чи іншому вигляді оснащені графічним процесором - спеціалізованою електронною схемою, основне призначення якої швидко маніпулювати та змінювати пам’ять для прискорення створення зображень у буфері кадру, призначеному для виведення на дисплей. Графічні процесори поділяються на дискретні та інтегровані. Тобто вони можуть мати власну оперативну пам’ять, RAM (Random Access Memory - пам'ять з довільним доступом), або використовувати частину пам’яті комп’ютера відповідно. В обох випадках об’єм цієї пам’яті є обмеженим, а тому виникає необхідність вирішити проблему того, як використати її найбільш ефективно.

Найбільш очевидним рішенням було б використання алгоритмів з максимальним коефіцієнтом стиснення зображень, для того щоб зменшити об’єм необхідної пам’яті [1]. Для цього можна було б наприклад скористатися форматом стиснення зображень без втрат PNG (Portable Network Graphics - портативна мережева графіка) з коефіцієнтом стиснення 2,7 або JPEG (Joint Photographic Experts Group - назва організації розробника), який надає коефіцієнт стиснення 16, проте є алгоритмом стиснення з втратами [2].

Проте насправді це не вирішує поставлену проблему. Варто пам’ятати що при занадто високому показнику коефіцієнту стиснення або коли алгоритм працює без втрат, все це призводить до того, що декомпресія зображення починає займати досить багато часу. А у випадку з графічним програмним забезпеченням, його швидкодія є досить важливим показником якості [3,4,5].

Саме цю проблему вирішують алгоритми класу DXTn. Вони є алгоритмами стиснення з втратами та надають коефіцієнт стиснення від 4 для DXT3 та DXT5 до 6 для DXT1 [6]. При цьому, артефакти які виникають в даних алгоритмах дуже важко помітити на відміну від JPEG. Тобто за своїми параметрами вони найкраще можуть виконати поставлену задачу. Також ці алгоритми були інтегровані у два найбільш популярних графічних API (Application Programming Interface, API - інтерфейс програмування застосунків) DirectX 6.0 та OpenGL 1.3, що ще збільшило їх популярність.

Аналіз останніх досліджень і публікацій.

Використання різних видів текстур є невід'ємною частиною сучасної комп'ютерної графіки. Це дозволяє істотно поліпшити візуальну деталізацію тривимірних об'єктів без ускладнення геометрії. У найпростішому випадку текстури є двомірне зображення, що накладається на тривимірний об'єкт. Існують різні дослідження щодо особливостей та основних вимог до методів стиснення і алгоритмам апаратної реалізації з урахуванням якості компресії-декомпресії текстур. Цим питанням присвячені експериментальні дослідження вітчизняних та закордонних вчених: Т. Палташева, І. Пермінова, J. McDonald, T. Inada, M.D. McCool, J. Strom T. Akenine-Moller, M. Olano та інші.

Вони одностайні в тому, що подальший розвиток форматів стиснення текстур може бути пов'язаний як з еволюцією існуючих підходів і їх розширенням на блоки більшого

розміру, так і з застосуванням схем зі змінним рівнем стиснення. Останній варіант дозволив би витрачати більшу кількість біт тільки для тих ділянок текстур, де це необхідно.

Мета статті. Метою даної роботи є створення власного прототипу додатку, який реалізує алгоритми блочного стиснення текстур DXT1, DXT3 та DXT5.

ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

На даний момент можна виділити дев’ять алгоритмів блочного стиснення текстур. Проте, через те, що деякі з них мають досить специфічну мету та використовуються доволі рідко, основними можна назвати алгоритми DXT1, DXT3 та DXT5, або інша їх назва під якою їх також можна зустріти у документації різного програмного забезпечення - BC1, BC2 та BC3 відповідно.

Вхідними даними для цих алгоритмів є інтерпретація зображення, будь-якого формату, у вигляді двовимірного масиву даних, у якому кожен елемент відповідає кольору пікселя у форматі RGBA або RGB.

Вихідними даними є текстура у форматі DDS, яка має наступну структуру [7]:

- Константа яка відповідає кодуванню чотирьох символів «DDS » (0x20534444)
- Опис даних файлу у вигляді заголовку. Він містить таку інформацію як геометричні розміри зображення, прапорці які вказують як саме інтерпретувати дані, та формат пікселей у якому було збережене зображення.
- Дані зображення у форматі, зазначеному у заголовку.

Насправді не існує стандартної імплементації даних алгоритмів, проте існує задокументований формат збереження вихідних даних для кожного з існуючих видів кодування.

Алгоритми можуть відрізнятися між собою різним коефіцієнтом стиснення, способом кодування кольорів та альфа каналу, тобто прозорості.

Характеристики алгоритму DXT1:

- Коефіцієнт стиснення - 6.
- Зберігає кольори у форматі RGB565, тобто виділяється п’ять бітів на збереження червоного каналу, шість бітів на збереження зеленого каналу, та п’ять на збереження синього каналу [4].
- Не зберігає значення альфа каналу, натомість кожен кодований піксель може бути повністю видимим, або повністю прозорим, тобто мати значення альфа каналу 255 або 0 відповідно [8].

Характеристики алгоритму DXT3:

- Коефіцієнт стиснення - 4.
- Зберігає кольори у форматі RGB565.
- Зберігає 4 старших біти значення альфа каналу. Тобто, якщо ми маємо значення альфа каналу 234 (11101010 у двійковій формі), то після декомпресії ми отримаємо значення 224 (11100000 у двійковій формі) [9]. Найбільше підходить для зображень з різкими змінами у значені альфа каналу.

Характеристики алгоритму DXT5:

- Коефіцієнт стиснення - 4.
- Зберігає кольори у форматі RGB565.

ISSN 2663 - 4023

– Зберігає значення альфа каналу у вигляді інтерполяції між ключовими її значеннями у блоках зображення. Найбільше підходить для зображень де альфа канал змінюється у вигляді градієнту [10].

Для того щоб на практиці найкраще побачити різницю між вихідними текстурами, найкраще підходить зображення яке створює градієнт чорного кольору за рахунок плавної зміни альфа каналу (рис. 1).

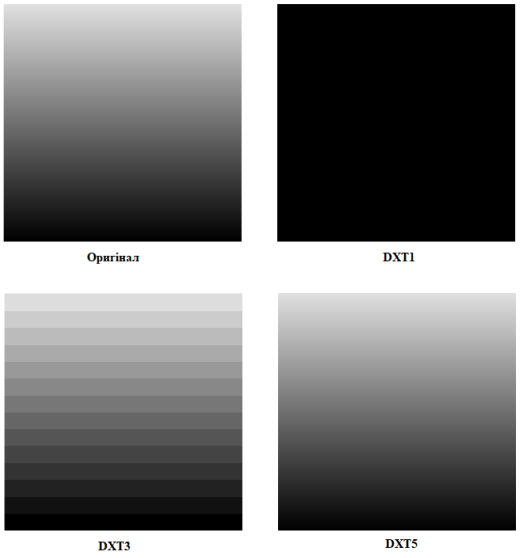

Рис. 1 - Приклад роботи алгоритмів

Як можно побачити з отриманих результатів, для даного зображення найбільш оптимальним буде алгоритм DXT5, проте це не завжди буде так. У випадку коли зображення не мало б альфа каналу (наприклад звичайний градієнт кольору) більш раціонально було б застосувати DXT1 через більший коефіцієнт стиснення, а тому менший розмір вихідної текстури. Аналогічно, якщо б зображення мало більш різкі зміни у альфа каналі, DXT3 створив би текстуру яка була б набагато більше схожа на оригінал аніж DXT5.

Отже усі алгоритми мають певні переваги та недоліки. Проте якщо обрати правильний спосіб кодування, вихідна текстура буде мінімально відрізнятися від оригіналу.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Для написання легко підтримуваного та швидкодіючого додатку для персональних комп'ютерів, була обрана мова програмування C++. Графічний інтерфейс користувача реалізований за допомогою фреймворку Qt [11].

Qt може бути використаний у багатьох сучасних мовах програмування. На даний момент існує підтримка для таких розповсюджених мов як C#, Java, Python та PHP. Проте найбільшу популярність фреймворк має саме для мови програмування C++.

Окрім інструментів для розробки графічного інтерфейсу, Qt містить класи для роботи з мережую, базами даних, графічним API OpenGL, SVG (Scalable Vector Graphics - масштабована векторна графіка) та XML (Extensible Markup Language, скорочено XML).

ISSN 2663 - 4023

Qt має власну IDE (Integrated development environment - Інтегроване середовище розробки) під назвою Qt Creator, який містить у собі вбудоване рішення для створення графічних інтерфейсів користувача Qt Designer.

Qt Designer дозволяє швидко та ефективно створити графічний інтерфейс додатку. Кожен елемент інтерфейсу, які згідно офіційної документації мають назву віджети, має безліч налаштувань, які впливають як на його зовнішній вигляд так і на поведінку. Qt надає можливість користувачу створювати повністю власні віджети або створювати власні версії вже існуючих за допомогою принципу наслідування класів.

Розвинутий механізм сигналів та слотів (рис. 2) дозволяє ще далі модифікувати поведінку віджетів, додаючи реакції на різні події такі як натискання на кнопку, зміну текстових значень тощо. Механізм сигналів та слотів є альтернативою звичному підходу з використанням колбеків, тобто функцій, посилання на які зберігаються у об'єктах та функціях та викликаються за певних обставин. Основним недоліком підходу з використанням колбеків є його контрінтуїтивність у багатьох його реалізаціях в різних мовах програмування та фреймворках, та проблема з перевіркою на коректність типів даних, які передаються у колбек у якості аргументів. Qt віджети мають багато заздалегідь створених слотів, проте користувач завжди може розширити їх за допомогою наслідування класів та опрацьовувати саме ті сигнали, які йому потрібні. Даний механізм гарантує правильність типів даних, які передаються у якості аргументів сигналу. Сигнатура сигналу має повністю відповідати сигнатурі приймаючого слоту. Більше того слоти можуть мати коротшу сигнатуру ніж сигнали, що дозволяє роботи не усі аргументи обов'язковими. Крім того усі проблеми с типами даних аргументів будуть відловлені на етапі компіляції, що дозволить користувачу заздалегідь їх виправити. Сигнали та слоти мають слабку з'язність, тобто класи які створюють сигнал та приймають його фактично нічого одне про одного не знають, та не мають залежності між собою.

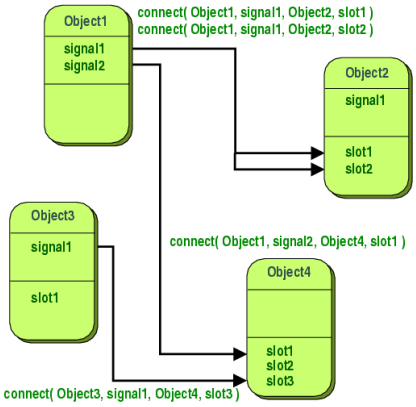

Рис. 2 - Схема роботи механізму слотів та сигналів

Середовище розробки

У якості середовища розробки була обрана IDE Qt Creator (рис. 3). Qt Creator - це крос-платформене середовище розробки з інтегрованими мовами програмування C++, JavaScript та QML (Qt Modeling Language) основна мета якого спростити розробку графічного інтерфейсу користувача. Дане середовище є частиною SDK (Software Development Kit - набір засобів розробки) для фреймворку Qt та використовує Qt API для інкапсуляції викликів функцій операційної системи. Включає у себе візуальний дебагер та редактор форм Qt Designer. IDE підтримує усі стандартні функції для сучасного

ISSN 2663 - 4023

середовища розробки, такі як підсвітка синтаксису, автодоповнення, продвинуті інструменти для рефакторингу коду, швидку навігацію між файлами проекту.

Qt Creator підтримує усі сучасні компілятори для мови програмування C++ такі як GCC, MinGW, MSVC та Clang.

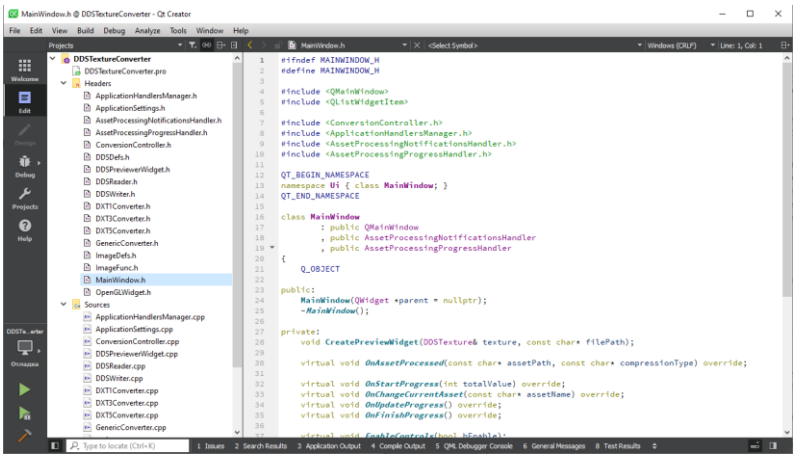

Рис. 3 - Інтерфейс середовища розробки Qt

Задля забезпечення розширюваності додатку та можливості повторного використання написаного коду для інших варіантів реалізації алгоритмів блочного стиснення текстур, виникає необхідність у формуванні ряду вимог до архітектури програмного коду.

Для реалізації графічного інтерфейсу користувача було обрано фреймворк Qt. Проте немає жодних переваг у тому, щоб реалізація алгоритмів мала залежності на даний фреймворк. Більше того, код алгоритмів та програмного інтерфейсу не мають взаємодіяти одне з одним напряму. Має бути створений певний інтерфейс доступу до алгоритмів, яким буде користуватися графічна частина додатку.

Дотримання даної вимоги дозволить у повному обсязі повторно використати раніше створений код реалізації алгоритмів для будь-якого іншого фреймворку для створення інтерфейсу користувача, або навіть у принципово іншому рішенні наприклад у вигляді плагіну для графічного редактора.

Для виконання цієї вимоги під час реалізації додатку необхідно застосувати принципи SOLID (single responsibility, open–closed, Liskov substitution, interface segregation и dependency inversion) [12] та DRY (Don't repeat yourself - Не повторюй себе) [13].

Структура проекту

Згідно вимог до архітектури проекту про незалежність реалізації алгоритмів від графічного інтерфейсу та принципів розробки SOLID та DRY були виділені наступні частини проекту:

- реалізація алгоритмів DXTn;
- графічний інтерфейс додатку;
- інтерфейс доступу до алгоритмів DXTn;
- інтерфейси, структури даних та функції загального користування;

Реалізація алгоритмів DXTn

Дана частина проекту складається з наступних частин:

- абстрактний клас для стиснення текстур
- клас для реалізації алгоритму DXT1

ISSN 2663 - 4023

– клас для реалізації алгоритму DXT3
– клас для реалізації алгоритму DXT5

Абстрактний клас існує для забезпечення принципу DRY та містить у собі код спільний для усіх імплементацій алгоритму, та водночас дозволяє прибрати залежності на конкретні реалізації алгоритмів. Тобто у випадку якщо один з алгоритмів необхідно буде прибрати і замість нього додати інший, код інших частин додатку фактично не зміниться, що робить систему більш розширюваною та гнучкою [14].

Класи для реалізації алгоритмів DXT1, DXT3 та DXT5 містять у собі функції та структури даних специфічні для кожного з них.

Загальний принцип роботи алгоритмів DXTn

В основі усіх алгоритмів блочного стиснення текстур покладена одна ідея, а тому багато етапів роботи у них однакові одне з одним.

Перед початком роботи алгоритмів, вхідне зображення необхідно розбити на текселі. Тексель - це фрагмент зображення розміром 4 на 4 пікселі. З цього визначення виникає умова що для роботи алгоритмів необхідно забезпечити щоб розміри вхідного зображення були кратні чотирьом.

Для вирішення даної проблеми були введені поняття фізичного та віртуального розміру зображення (рис. 4). Фізичний розмір зображення - це його справжній розмір. Уявний розмір - це розмір з урахуванням необхідних підкладок до зображення з метою зробити його розміри кратними чотирьом. Додані таким чином пікселі мають чорний повністю прозорий колір, тобто у форматі RGBA мають значення 0, 0, 0, 0. При збереженні текстури у заголовку вказуються її фізичні розміри, проте дані зберігаються в тому вигляді, в якому вони були отримані після стискання зображення з його уявним розміром. Це необхідно для того, що без повного обсягу даних текстуру неможливо буде відтворити і фактично такий файл можна буде вважати пошкодженим [15].

Далі кожен тексель буде оброблений алгоритмом та збережений у відповідному стиснутому форматі. Сам процес обробки текселю можна розділити на два етапи: стиснення кольорів та стиснення альфа каналу. Алгоритми DXT1, DXT3 та DXT5 однаково кодують кольори, проте принципово по різному кодують альфа канали [16].

Після цього буде сформований заголовок для DDS текстури, який разом зі стиснутими даними буде записаний у файл, що стане завершенням роботи додатку по конвертації одного зображення в DDS текстуру [17].

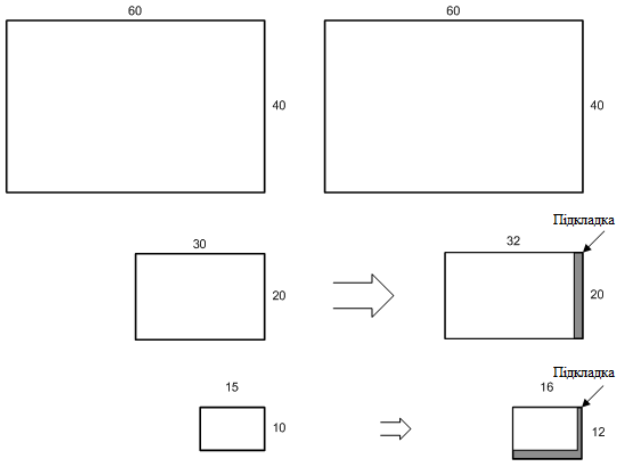

Рис. 4 - Порівняння фізичного та уявного розміру зображень

ISSN 2663 - 4023

Кодування кольорів у алгоритмах DXT1, DXT3 та DXT5

Розглянемо як працює кодування кольорів на прикладі. Вважаємо що зображення з яким ми працюємо вже має розміри кратні чотирьом. Пронумеруємо пікселі одного текселю латинськими літерами від *a* до *p* (рис. 5). Кожен піксель містить у собі колір у форматі RGBA, проте для кодування кольорів нас не цікавить альфа канал, тому для простоти опустимо його та будемо розглядати лише RGB компоненту кольору. Дана структура займає у пам'яті три байти (три канали по одному байту кожен), тобто один тексель займає 48 байтів.

Введемо таке поняття як основні кольори текселю. Для цього уявімо тривимірний простір з Декартовою системою координат *XYZ*, де канали кольору виконують роль координатних осей (R = X, G=Y, B=Z).

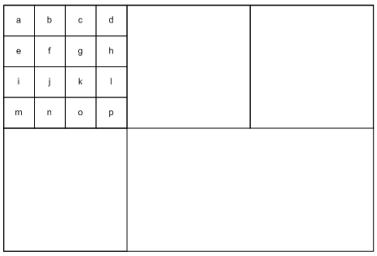

Рис. 5 - Розбиття зображення на текселі

Колір кожного пікселя відповідає точці у даному координатному простору. Основними кольорами текселю вважаються кольори відстань між якими у даному координатному просторі найбільша. У випадку якщо таких пар кольорів декілька, обирається довільна з них. Назвемо ці кольори *color_0* та *color_1* відповідно. Дані кольори зберігаються в даних текстури у форматі RGB565 [19].

Далі обчислюються два додаткові проміжні кольори *color_2* та *color_3* за наступними формулами:

$$color\_2 = 2/3 * color\_0 + 1/3 * color\_1 \quad (1)$$

$$color\_3 = 1/3 * color\_0 + 2/3 * color\_1 \quad (2)$$

Усі математичні операції виконуються покомпонетно. Тобто над кожним каналом кольору окремо. Потім кожному кольору присвоюється індекс від 0 до 3 у двійковій формі, тобто 00, 01, 10 та 11 відповідно.

Тепер користуючись тим самим тривимірним простором для кожного пікселя вирахуємо відстань між його кольором та кольорами *color_0*, *color_1*, *color_2* та *color_3*. Зберігаємо індекс кольору відстань до якого була найменша.

В результаті отримуємо наступну структуру даних (рис. 6):

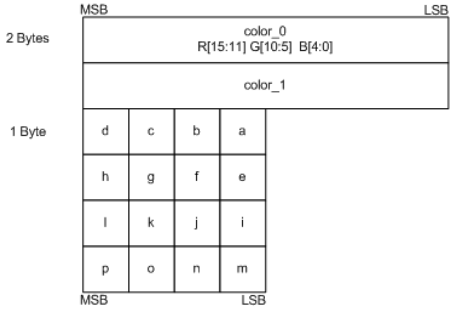

Рис. 6 - Вигляд кодованих кольорів текселю у пам'яті

ISSN 2663 - 4023

В комірках від *a* до *p* зберігаються обрані нами індекси, розміром по 2 біти кожен.

В результаті уся структура яка повністю кодує кольори одного текселя займає у пам'яті 8 байтів (два кольори по 2 байти та 4 байти для збереження індексів). Тобто для збереження кольорів ми використали в 6 разів менший об'єм пам'яті.

Відмінності у форматі текстури для DirectX та OpenGL

DirectX та OpenGL це два найбільш поширені у світі графічні API. Основною перевагою OpenGL можна виділити те що він працює на усіх сучасних операційних системах в той час як DirectX працює лише на операційних системах розроблених корпорацією Microsoft. Не зважаючи на це, вони є абсолютно рівними конкурентами, а більшість сучасних програм навіть дають користувачу вибір, яким API він хоче користуватися (якщо звісно цей вибір можливий з урахуванням операційної системи).

З точки зору текстур DirectX та OpenGL мають одну ключову відмінність. Вони використовують різні системи координат для роботи з текстурами (рис. 7). Більшість додатків які реалізують алгоритми блочного стиснення текстур ігнорують дану відмінність, проте автор роботи вважає що це призводить до значного зменшення потенційних користувачів додатку [20].

Для вирішення цієї проблеми у додатку буде створено окреме налаштування яке буде обирати користувач. Від значення цього налаштування буде залежати у якому порядку зображення буде розбите на текселі. Також варто зазначити що таке «віддзеркалення» стосується також підкладки для зображення у випадку якщо його фізичні розміри не кратні чотирьом.

Окрім цього, додаткових змін у коді робити не потрібно. Ця особливість також врахована під час перегляду текстур. У користувача буде можливість перегорнути текстуру, якщо у цьому буде необхідність.

Варто зауважити що у заголовку текстури немає жодної інформації стосовно того для якого графічного API вона була створена. Тому користувач повинен заздалегідь знати у якому форматі йому необхідно зберегти текстуру.

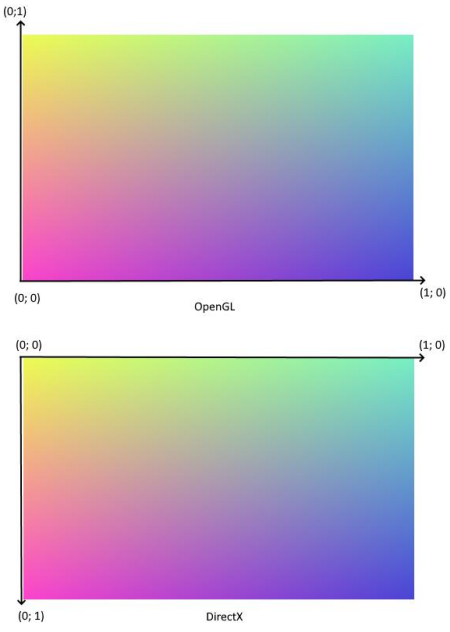

Рис. 7 - Системи координат у DirectX та OpenGL

Реалізація алгоритму DXT1

Як вже було зазначено раніше, алгоритм DXT1 не зберігає значення альфа каналу. Натомість пікселі можуть бути або повністю видимими або повністю прозорими.

Для цього в стандартний алгоритм кодування кольору вносяться наступні зміни. У випадку коли у тексилі присутній прозорий піксель, змінюється принцип за яким обчислюються *color_0*, *color_1*, *color_2* та *color_3*. *color_2* починає обчислюватися за наступною формулою:

$$color\_2 = 1/2 * color\_0 + 1/2 * color\_1 \quad (3)$$

color_3 присвоюється значення повністю прозорого чорного кольору (у форматі RGBA: 0, 0, 0, 0). Далі алгоритм працює без змін, окрім того що коли піксель є прозорим, він автоматично обирає індекс *color_3*.

Те як саме трактувати кодування кольорів для алгоритму DXT1 визначається порівнянням кольорів *color_0* та *color_1*. Якщо колір *color_0* менший за *color_1*, кольори текселю будуть інтерпретуватися вважаючи *color_3* повністю прозорим чорним кольором. Інакше кольори інтерпретуються стандартно.

Варто зауважити що при порівнянні кольорів вони порівнюються не по компонентно, а у вигляді довжини від точки (0; 0; 0) на координатному просторі до точки яка відповідає кольору.

Переваги алгоритму:

- коефіцієнт стиснення 6;
- найшвидший серед алгоритмів DXTn

Недоліки алгоритму:

- сильне спотворення зображень з альфа каналом

Даний алгоритм найбільш доцільно використовувати при роботі з зображеннями у яких не змінюється альфа канал, або його значення буває лише 0 або 255 (повністю прозорий та повністю видимий). В такому випадку він є найбільш ефективним оскільки надає найбільший коефіцієнт стиснення порівняно з алгоритмами DXT3 та DXT5. У випадку роботи з зображеннями у яких присутні альфа канал, рекомендовано скористатися іншими алгоритмами DXTn.

Реалізація алгоритму DXT3

Кодування кольорів у алгоритмі DXT3 відбувається стандартне без змін. Проте на відміну від DXT1 вже відбувається кодування альфа каналу.

Алгоритм DXT3 зберігає альфа канал кожного пікселя текселю проте у спрощеній формі. Зберігається лише чотири старших біти значення альфа каналу. Наприклад, якщо початкове значення альфа каналу було рівне 234 (11101010 у двійковій формі), то після декомпресії ми отримаємо значення 224 (11100000 у двійковій формі). Таке «обрізання» альфа каналу створює візуальні артефакти на зображеннях, де він змінюється плавно.

Дані про альфа канал записуються перед інформацією про кодування кольорів. Результуюча структура (рис. 8) має розмір 16 байтів (8 байтів на кодування кольорів та 8 байтів на збереження альфа каналу). Без використання алгоритму один тексель з форматом пікселя RGBA займає у пам'яті 64 байти (16 пікселів по 4 байти кожен). Тобто алгоритм DXT3 надає коефіцієнт стиснення 4.

ISSN 2663 - 4023

Рис. 8 - Структура кодованого теселю алгоритму DXT3

Переваги алгоритму:

- зберігає альфа канал;
- артефакти майже відсутні для зображень з різкими переходами альфа каналу

Недоліки алгоритму:

- значні артефакти для зображень з плавним переходом альфа каналу

Даний алгоритм найбільш доцільно використовувати при роботі з зображеннями у яких присутні різкі переходи альфа каналу. У випадку коли альфа канал відсутній на зображення, використання алгоритму не є доцільним, оскільки його коефіцієнт стиснення є меншим ніж у DXT1. Якщо у вихідної текстури надто помітні артефакти, рекомендовано скористатися алгоритмом DXT5 та обрати найкращий з результатів.

Реалізація алгоритму DXT5

Кодування кольорів у алгоритмі DXT5 відбувається стандартне без змін.

Основна відмінність від інших алгоритмів полягає у способі кодування альфа каналу.

Алгоритм DXT5 кодує альфа канал за принципом дуже схожим до того як у даних алгоритмах кодується колір.

Для початку алгоритм знаходить найбільше та найменше значення альфа каналу у текселі та зберігає їх у *alpha_0* та *alpha_1* відповідно. Далі обчислюються значення *alpha_2*, *alpha_3*, *alpha_4*, *alpha_5*, *alpha_6* та *alpha_7*. Аналогічно кодуванню кольорів у алгоритмі DXT1, проміжні значення альфа каналу залежать від того як *alpha_0* та *alpha_1* порівнюються між собою.

У випадку коли *alpha_0* більше за *alpha_1*:

$$alpha\_2 = 6/7 * alpha\_0 + 1/7 * alpha\_1 \quad (4)$$

$$alpha\_3 = 5/7 * alpha\_0 + 2/7 * alpha\_1 \quad (5)$$

$$alpha\_4 = 4/7 * alpha\_0 + 3/7 * alpha\_1 \quad (6)$$

$$alpha\_5 = 3/7 * alpha\_0 + 4/7 * alpha\_1 \quad (7)$$

ISSN 2663 - 4023

$$alpha\_6 = 2/7 * alpha\_0 + 5/7 * alpha\_1 \quad (8)$$

$$alpha\_7 = 1/7 * alpha\_0 + 6/7 * alpha\_1 \quad (9)$$

У випадку коли *alpha_0* менша або рівна *alpha_1*:

$$alpha\_2 = 4/5 * alpha\_0 + 1/5 * alpha\_1 \quad (10)$$

$$alpha\_3 = 3/5 * alpha\_0 + 2/5 * alpha\_1 \quad (11)$$

$$alpha\_4 = 2/5 * alpha\_0 + 3/5 * alpha\_1 \quad (12)$$

$$alpha\_5 = 1/5 * alpha\_0 + 4/5 * alpha\_1 \quad (13)$$

$$alpha\_6 = 0 \quad (14)$$

$$alpha\_7 = 255 \quad (15)$$

Далі кожному обчисленому значенню альфа каналу присвоюється індекс від 0 до 7 у двійковій формі, тобто 000, 001, 010, 011, 100, 101, 110 та 111 відповідно.

Значення альфа каналу кожного пікселя порівнюється з усіма обчисленими значеннями, та обирається той, різниця з яким буде найменшою. У випадку якщо таких варіантів декілька, обирається довільний. Отриманий індекс зберігається.

Так само як і в алгоритмі DXT3, дані про кодування альфа каналу зберігаються перед інформацією про кольори. Вихідна структура даних (рис. 9) займає 16 байтів (2 байти на збереження *alpha_0* та *alpha_1*, 6 байтів на збереження індексів для відтворення альфа каналу, та 8 байтів на збереження кольорів). Отримуємо той самий коефіцієнт стиснення 4, той самий що і в алгоритму DXT3.

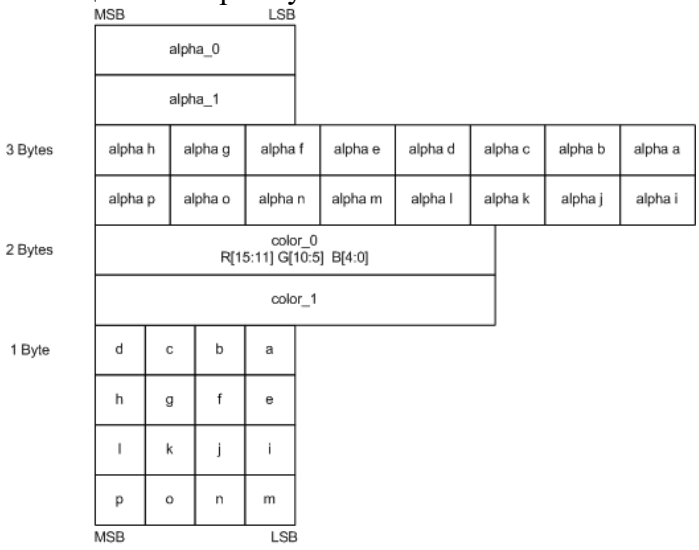

Рис. 9 - Структура кодованого теселю алгоритму DXT5

Переваги алгоритму:

- зберігає альфа канал;
- артефакти майже відсутні для зображень з плавними переходами альфа каналу

ISSN 2663 - 4023

Недоліки алгоритму:

– значні артефакти для зображень з різким переходом альфа каналу

Даний алгоритм найбільш доцільно використовувати при роботі з зображеннями у яких присутній плавний перехід альфа каналу. У випадку коли альфа канал відсутній на зображення, використання алгоритму не є доцільним, оскільки його коефіцієнт стиснення є меншим ніж у DXT1. Якщо у вихідної текстури надто помітні артефакти, рекомендовано скористатися алгоритмом DXT3 та обрати найкращий з результатів.

Структура запропонованого рішення

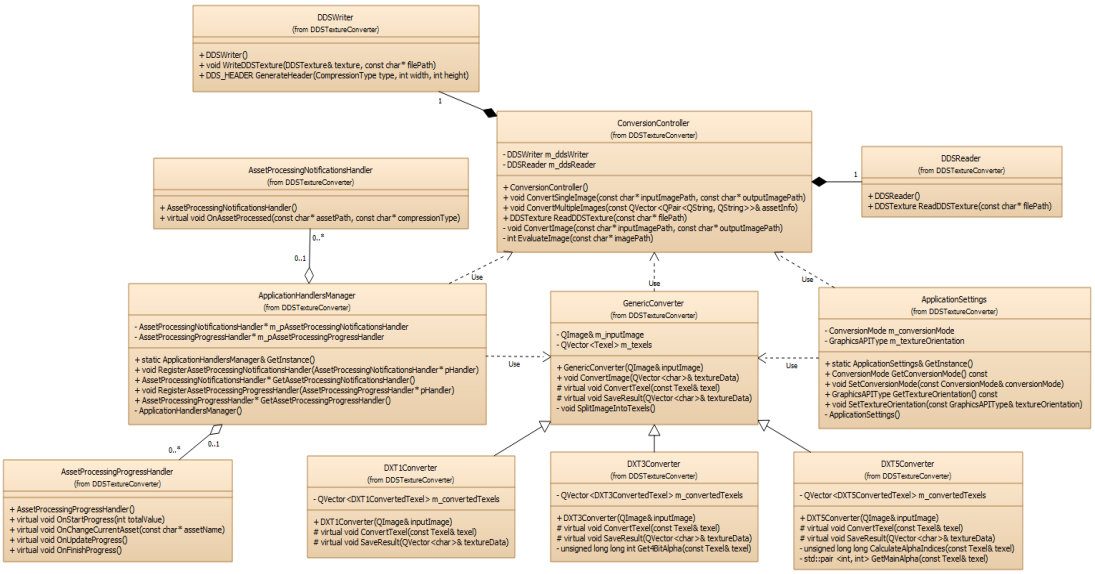

Рис. 10 - UML-діаграма реалізації алгоритмів

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Результатом виконаної роботи є детальний аналіз області застосування алгоритмів блочного стиснення текстур та варіантів їх технічноїреалізації. На основі аналізу було обрано найбільш доцільний варіант реалізації та основі нього розроблено прототип.

В роботі було розглянуто проблему, яку вирішують алгоритми блочного стиснення текстур та їх переваги над альтернативними алгоритмами.

Проаналізовано різні варіанти реалізації алгоритмів та виділено їх основні переваги та недоліки. На основі аналізу обрано варіант технічної реалізації прототипу та сформовано функціональні вимоги до нього.

Розроблено програмну архітектуру прототипу, що дозволить забезпечити його розширюваність та можливість повторного використання напрацювань для інших способів реалізації.

Детально розглянуто принцип роботи алгоритмів блочного стиснення текстур та їх структуру вихідних даних. На основі переваг та недоліків алгоритмів були сформовано рекомендації щодо їх використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жураковський, Б.Ю. (2013). Матричні та комбіновані способи стиснення даних при передачі. *Наукові записки Українського науково-дослідного інституту зв'язку*, (2), 23 – 26.
2. Жураковський, Б.Ю., Жураковський, Я.Ю. (2001). Каскадне стиснення інформації під час обробки в автоматизованих системах управління. *Зв'язок*, (2), 44-46.
3. Жураковський, Б.Ю. (2013). Способи стиснення даних при архівації. *Сучасний захист інформації*, (2), 65-68.
4. Жураковський, Б.Ю. (2015). Аналіз кількості алгоритмів стиснення у каскаді при використанні каскадних методів. *Сучасний захист інформації*, (1), 56–61. http://journals.dut.edu.ua/index.php/dataprotect/issue/view/17
5. Жураковський, Б.Ю. (2014). Аналіз ефективності каскадних методів стиснення інформації. *Сучасний захист інформації*, (3), 84-89. http://journals.dut.edu.ua/index.php/dataprotect/issue/view/12
6. Block Compression–Microsoft. https://docs.microsoft.com/en-us/windows/win32/direct3d10/d3d10-graphics-programming-guide-resources-block-compression.
7. Zhurakovskiy, B., & Tsopa, N. (2019b). Assessment technique and selection of interconnecting line of information networks. У *2019 3rd international conference on advanced information and communications technologies (AICT)*. IEEE. https://doi.org/10.1109/aiact.2019.8847726
8. Programming Guide for DDS – Microsoft. https://docs.microsoft.com/en-us/windows/win32/direct3ddds/dx-graphics-dds-pguide
9. Документація до OpenGL – Khronos. https://www.khronos.org/opengl/wiki
10. Специфікація алгоритмів блочного стиснення текстур – Khronos. https://www.khronos.org/registry/DataFormat/specs/1.1/dataformat.1.1.html
11. Веб-сторінка графічного редактору Paint.NET. https://www.getpaint.net/
12. Веб-додаток для конвертації між різними форматами файлів. https://convertio.co/
13. Документація Amazon Lumberyard – AmazonWebServices. https://docs.aws.amazon.com/lumberyard/?id=docs_gateway
14. Веб-сторінка додатку для конвертації PNGзображень у DDSтекстури. http://www.easy2convert.com/png2dds/
15. History of C. https://www.cplusplus.com/info/history/
16. Документація фреймворку Qt - Qt Group. https://doc.qt.io/qt-5/
17. SOLID: The First 5 Principles of Object Oriented Design. https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
18. Zhurakovskyi, B., Boiko, J., Druzhynin, V., Zeniv, I., & Eromenko, O. (2020). Increasing the efficiency of information transmission in communication channels. *Indonesian Journal of Electrical Engineering and Computer Science*, *19*(3), 1306-1315. https://doi.org/10.11591/ijeecs.v19.i3.pp1306-1315
19. Документація мови програмування C++. https://en.cppreference.com/w/
20. Навчальний ресурс з використання OpenGL. https://learnopengl.com/

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (12), 2021
ISSN 2663 - 4023

Konstantin Nesterenko
Student
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
ORCID ID: 0000-0003-3921-4324
2000kostia@gmail.com

Bohdan Zhurakovskyi
Doctor of Technical Sciences, Professor
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
ORCID ID: 0000-0003-3990-5205
zhurakovskiybyu@tk.kpi.ua

IMAGE CONVERTER BASED ON BLOCK COMPRESSION ALGORITHMS OF DXT1, DXT3 AND DXT5 TEXTURES

Abstract. This article analyzes the existing applications that implement block texture compression algorithms. Based on it, the most optimal variant of technical implementation is introduced. A set of technologies for the implementation of the prototype is selected and substantiated and its architecture is developed on the basis of the principles that ensure the maximum extensibility and purity of the code. With the development of technology and the integration of computerized systems into all possible areas of human activity, more and more software with three-dimensional graphics is being used. Such programs have long since ceased to be used only in the entertainment field for tasks such as computer game development or special effects for cinema. Now with their help doctors can plan the most complex operations, architects check the developed plans of constructions and engineers to model prototypes without use of any materials. On the one hand, such a rapid increase can be explained by the increase in the power of components for personal computers. For example, modern graphics processors, which play a key role in the operation of graphics software, have become much faster in recent decades and have increased their memory hundreds of times. However, no matter how many resources the system has, the question of their efficient use still remains. It is to solve this problem that block texture compression algorithms have been created. In fact, they made it possible to create effective software when computer resources were still quite limited. And with increasing resources allowed to develop software with an incredible level of detail of the models, which led to its active implementation in such demanding areas as medicine, construction and more. The end result of this work is a developed application that takes into account the modern needs of the user. During the development, the most modern technologies were used for the highest speed and relevance of the application. The main advantages and disadvantages of existing solutions were also taken into account during the development. The capabilities of the system were tested using manual testing on a local machine.

Keywords: block texture compression algorithms, texture, DXTn, DDS, Qt.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Zhurakovskyi, B.Iu. (2013). Matrychni ta kombinovani sposoby stysnennia danykh pry peredachi. Naukovi zapysky Ukrainskoho naukovo-doslidnoho instytutu zviazku, (2), 23 – 26.
2. Zhurakovskyi, B.Iu., Zhurakovskyi, Ya.Iu. (2001). Kaskadne stysnennia informatsii pid chas obrobky v avtomatyzovanykh systemakh upravlinnia. Zviazok, (2), 44-46.
3. Zhurakovskyi, B.Iu. (2013). Sposoby stysnennia danykh pry arkhivatsii. Suchasnyi zakhyst informatsii, (2), 65-68.
4. Zhurakovskyi, B.Iu. (2015). Analiz kilkosti alhorytmiv stysnennia u kaskadi pry vykorystanni kaskadnykh metodiv. Suchasnyi zakhyst informatsii, (1), 56–61. http://journals.dut.edu.ua/index.php/dataprotect/issue/view/17
5. Zhurakovskyi, B.Iu. (2014). Analiz efektyvnosti kaskadnykh metodiv stysnennia informatsii. Suchasnyi zakhyst informatsii, (3), 84-89. http://journals.dut.edu.ua/index.php/dataprotect/issue/view/12
6. Block Compression–Microsoft. https://docs.microsoft.com/en-us/windows/win32/direct3d10/d3d10-graphics-programming-guide-resources-block-compression.

7. Zhurakovskiy, B., & Tsopa, N. (2019b). Assessment technique and selection of interconnecting line of information networks. U 2019 3rd international conference on advanced information and communications technologies (AICT). IEEE. https://doi.org/10.1109/aiact.2019.8847726
8. Programming Guide for DDS – Microsoft. https://docs.microsoft.com/en-us/windows/win32/direct3ddds/dx-graphics-dds-pguide
9. Dokumentatsiia do OpenGL – Khronos. https://www.khronos.org/opengl/wiki
10. Spetsyfikatsiia alhorytmiv blochnoho stysnennia tekstur – Khronos. https://www.khronos.org/registry/DataFormat/specs/1.1/dataformat.1.1.html
11. Veb-storinka hrafichnoho redaktoru Paint.NET. https://www.getpaint.net/
12. Veb-dodatok dlia konvertatsii mizh riznymy formatamy failiv. https://convertio.co/
13. Dokumentatsiia Amazon Lumberyard – AmazonWebServices. https://docs.aws.amazon.com/lumberyard/?id=docs_gateway
14. Veb-storinka dodatku dlia konvertatsii PNGzobrazhen u DDStekstury. http://www.easy2convert.com/png2dds/
15. History of C. https://www.cplusplus.com/info/history/
16. Dokumentatsiia freimvorku Qt - Qt Group. https://doc.qt.io/qt-5/
17. SOLID: The First 5 Principles of Object Oriented Design. https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
18. Zhurakovskyi, B., Boiko, J., Druzhynin, V., Zeniv, I., & Eromenko, O. (2020). Increasing the efficiency of information transmission in communication channels. Indonesian Journal of Electrical Engineering and Computer Science, 19(3), 1306-1315. https://doi.org/10.11591/ijeecs.v19.i3.pp1306-1315
19. Dokumentatsiia movy prohramuvannia C++. https://en.cppreference.com/w/
20. Navchalnyi resurs z vykorystannia OpenGL. https://learnopengl.com/