Issue 11, 2022

https://doi.org/10.23939/sisn2022.11.110

УДК 004.94

АНАЛІЗ ПРОЦЕСІВ ФОРМУВАННЯ СИМУЛЯЦІЙ З ВИКОРИСТАННЯМ ГРАФІЧНОГО ПРОЦЕСОРА

Юліанна Калинич[1], Юрій Білак[2], Руслан Небесний[3], Павло Федорка[4]

[1, 2, 4] Ужгородський національний університет,
[3] Тернопільський національний технічний університет імені Івана Пулюя,
[1] e-mail: kalynych.yulianna@stu dent.uzhnu.edu.ua, ORCID: 0000-0001-7102-4868,
[2] e-mail: yuriy.bilak@uzhnu.edu.ua, ORCID: 0000-0001-5989-1643,
[3] e-mail: nebesnyi@gmail.com, ORCID: 0000-0001-8886-8346,
[4] e-mail: fedorkapavlo@gmail.com, ORCID: 0000-0002-9242-5588

Досліджено цінність процесів симуляцій для дослідницької діяльності та визначено основні причини доцільності проведення експериментів у віртуальному просторі.

За допомогою рушія гри Unity розроблено процеси симуляції в двовимірному та тривимірному просторах. Використано технології симуляції в двовимірному просторі із застосуванням реакційно-дифузійної моделі Грея – Скотта. Розглянута модель передбачає дослідження системи, в якій відбувається реакція дифузії двох речовин. Побудований на основі цієї моделі програмний продукт дає змогу моделювати візерунок дифузії у режимі реального часу або пришвидшити плин часу в процесах симуляції. Програмний продукт дає змогу конфігурувати основні параметри реакції, що уможливить побудову симуляції процесів у будь-яких необхідних речовинах чи системах. Результат візуалізації процесів симуляції можна переглядати в кількох режимах, що дає змогу оцінити різні аспекти досліджуваної реакції в будь-який момент часу, переглядати концентрацію речовин, досліджувати зміни показників концентрації речовини на одиницю часу в кожній точці площини реакції.

Під час створення візуалізації процесів симуляції за допомогою реакційно-дифузійної моделі Грея – Скотта було досліджено можливість застосування до них методу оптимізації за допомогою обчислень з виконанням графічного ядра. Дослідження показало доцільність розпаралелювання обчислень задля виконання їх на багатьох потоках графічного адаптера. Кожному пікселю вхідного зображення, на якому подано початковий візерунок нанесення речовин на площину, відведено окремий потік, що сприяє виконанню обчислень показників концентрації, та проаналізовано їх зміни в матеріальній точці площини реакції. Доведено, що кількість запущених обчислювальних потоків повинна дорівнювати кількості пікселів зображення.

Тож реалізована модель візуалізації дифузійних процесів сприяє кращому розумінню і глибшому дослідженню хімічних реакцій синтезу кровотворення, ферментації та бродіння тощо.

Процеси симуляції в тривимірному просторі розглянуто на прикладі поведінки зграї однотипних об'єктів. Комп'ютерно реалізовано процеси уникання зіткнень особин зграї між собою, підтримки спільного напрямку руху, обминання перешкод на шляху. Модуль моніторингу дій зграї управляє не лише поведінкою всього гурту, а й кожної окремої особини. Тож змодельована поведінка зграї відображає її реальні дії у природних умовах.

Запропонований підхід до візуалізації процесів симуляції апробовано щодо можливості їх оптимізації за допомогою обчислень на графічному ядрі. Поведінка кожної особини зграї управляється окремим незалежним модулем прийняття рішень і потребує взаємодії з усіма іншими

елементами зграї та визначення її впливу на рішення щодо вибору траєкторії руху досліджуваним елементом зграї. Для кожної особини зграї графічне ядро виділяє окремий потік для визначення рішення щодо подальшого руху.

Ключові слова: симуляція; Unity; C#; HLSL; шейдер; комп'ют шейдер; реакційно-дифузійна модель Грея – Скотта; зграї; птахоподібні об'єкти.

Вступ

Переваги і перспективність візуалізації процесів симуляцій діяльності для розвитку науки очевидні. Такий інструментарій дослідницької діяльності допомагає науковцям багатьох галузей вирішувати актуальні завдання. Процеси віртуальної симуляції стали важливою частиною моделювання роботи багатьох природних систем у фізиці, хімії та біології, в економіці та соціальних науках (наприклад, в обчислювальній соціології), а також в інженерній справі. Комп'ютерне моделювання, наприклад, дає змогу дізнатися стан системи в будь-який момент часу, якщо для неї описано зміни кожного її об'єкта впродовж деякого проміжку часу; створити модель системи ще задовго до того, як за нею почали наглядати; прогнозувати, якою система буде через проміжок часу, упродовж якого неможливо спостерігати за нею у режимі реального часу; або змоделювати процес, систему чи явище, які ніколи в реальних умовах не існували й існувати не могли.

Правильно створена візуалізація процесів симуляції має безліч переваг. Якщо інженер-програміст, що створив візуалізацію процесів, симуляцію, подбав про її гнучкість, тобто можливість якомога точніше налаштовувати параметри системи, то один і той самий застосунок із візуалізації процесів симуляції можна використати, запустивши її з різними вхідними параметрами, що дуже зручно під час здійснення досліджень у різних галузях знань. Це дає змогу відтворювати той самий експеримент декілька разів, досліджуючи кореляцію між введеними параметрами і наслідками спостережень.

Постановка проблеми і її вирішення шляхом оптимізації

Побудова точної моделі симуляції потребує значної кількості обчислень, щоб врахувати всі фактори, які впливають на поведінку складових системи. Цей факт спонукає до пошуку методів прискорення процесів симуляції, щоб отримати змогу відтворювати деталізованіші процеси, для обчислення яких знадобиться велика кількість ітерацій або підтримка якомога більшої кількості простих об'єктів. Такий підхід сприяє отриманню об'єктивних результатів дослідження, оскільки уможливлює одержання результату такої симуляції та спостереження за змінами в реальному масштабі часу.

Рішення було знайдено на підставі аналізу ситуації на ринку комп'ютерних комплектуючих, завдяки швидкому розвитку геймерського "заліза". Розробники ігор з найбільшим ентузіазмом нарощували саме графічну їх складову, а виробники комплектуючих до персональних комп'ютерів хотіли підтримувати високий fps, задовольняючи запити геймерів до конфігурації комп'ютерної техніки. Такі тенденції призвели до того, що завдання виконання обчислень було поділено між двома пристроями: процесором і графічним ядром. Таке рішення пов'язане із відмінностями в принципах роботи процесора і відеокарти, які полягають в наявності у процесора невеликої кількості багатофункціональних ядер, а у відеокарті – значної кількості простих ядер. Тому виконання алгоритмів на графічному ядрі є надзвичайно швидким, проте має недоліки, серед яких можна назвати створення великої кількості потоків, що ускладнює виконання операцій. Тобто на кожному потоці, що виконується на відеокарті, можливо здійснювати лише прості математичні операції. Якщо неможливо розпаралелити вирішення завдання на багато ізольованих один від одного процесів, то його недоцільно запускати на графічному ядрі.

Для забезпечення ефективного розподілу завдань між компонентами системи розробники застосунку використали інструментарій управління обчисленнями, який дає змогу спрямовувати їх для виконання на конкретний обчислювальний пристрій. Розробки компанії Nvidia надали можливість писати код для графічної системи, щоб здійснювати обчислення в графічному ядрі. Таким інструментом комунікації розробника і низькорівневого програмного забезпечення графічного процесора є шейдер.

Шейдер (англ. *shader* – затінювач) – це програма для відеокарти, яка використовується в тривимірній графіці для визначення остаточних параметрів об'єкта або зображення, може охоплювати технологію поглинання і розсіювання світла, накладення текстури, відображення і заломлення променів, затінення, зміщення поверхні та зміни безлічі інших параметрів [1]. Ці спеціальні скрипти підтримуються в середовищі розробки Unity, що дає змогу використовувати їх для дослідження різних об'єктів, створюючи сцени-симуляції. Шейдери розподіляють на підвиди залежно від завдань, які покладаються на конвеєр візуалізації. Наприклад, шейдер фрагмента – це програма для відеокарти, яка відповідає за вихідний стан кожного окремого пікселя, тобто за застосування таких ефектів, як освітлення, відображення, рефракція, карти нормалей, текстури.

У кінці цього процесу надаються кольори пікселів, кожен із яких записується в один з буферів кадрів. Ці буфери кадрів після заповнення міняються місцями і новозаповнений буфер виводиться на екран. В цій розробці використано такий підвид рейдерів, як обчислювальні (комп'ют) шейдери.

Обчислювальні шейдери (*compute shaders*) – це ще один особливий підвид шейдерів, які виконуються поза звичайним конвеєром візуалізації (концептуальна модель, що описує кроки, які потрібно виконати графічній системі для рендерингу тривимірної сцени на двовимірний екран) [2]. Їх можна використовувати для багатопотокових алгоритмів GPGPU (метод виконання обчислень, який характеризується використанням графічного процесора, що зазвичай забезпечує опрацювання комп'ютерної графіки, для того щоб виконувати обчислення в додатках) або для прискорення частин ігрової візуалізації. Для ефективного їх використання потрібні глибокі знання архітектури GPU та паралельних алгоритмів [3].

На відміну від звичайного шейдера, обчислювальний шейдер може повертати будь-які дані. Тобто він зможе повернути текстуру з проєкцією зрендереної сцени, яку потім можна виводити на екран.

Для написання шейдерів використовується спеціальна мова програмування, яка отримала назву HLSL та розроблена компанією Microsoft для надання доступу до функцій графічних бібліотек DirectX та Direct3D. Її синтаксис дуже подібний до мови C. Існують й інші мови написання шейдерів, наприклад, GLSL (для OpenGL та його похідних), Cg (вийшла із вжитку), PSSL (застосовується для розробок під платформу PlayStation), MSL (застосовується під час роботи з графічною бібліотекою Metal), але автори вибрали мову HLSL, адже лише вона підтримується в Unity.

Незважаючи на те, що Unity використовується для кросплатформних розробок, шейдери для неї повинні бути написані на HLSL, а сам рушій пізніше перекладе їх на відповідну платформі мову шейдерів. Це дасть змогу використати результати дослідження у розробках для будь-яких платформ.

Аналіз принципів розпаралелювання задач для виконання на графічному ядрі, який реалізує мова HLSL, дає змогу окреслити коло завдань, для вирішення яких він підходить найкраще.

Щоб виконати рендеринг, потрібно спочатку створити цільовий рендер, тобто текстуру, в яку буде рендеритися кадр і який буде записаний в destination. Необхідно задати їй відповідний розмір і повідомити про це комп'ют шейдер, передавши йому це зображення як параметр, в який потоки комп'ют шейдера записуватимуть результати обчислень кольору.

Тепер можна перейти до запуску шейдера, що виконується за допомогою функції Dispatch(int kernelIndex, int threadGroupsX, int threadGroupsY, int threadGroupsZ). Ця функція "запускає" комп'ют шейдер на GPU. Перший параметр цієї функції – це індекс функції ядра комп'ют шейдера. Наступні три параметри визначають кількість груп потоків, які вказують у трьох вимірах, завдяки чому можна просто застосувати комп'ют шейдери до завдань будь-якої розмірності. Розмір групи потоків задають в самому обчислювальному шейдері (використовуючи атрибут HLSL "numthreads"), і загальна кількість викликів обчислювального шейдера (і відповідно потоків) в групі потоків дорівнює кількості груп потоків, помножених на розмір групи потоків. Розмір групи потоків можна запитувати за допомогою функції GetKernelThreadGroupSizes. Ці групи вибиратимуться випадково та виконуватимуться паралельно.

У такому випадку доцільно створити по одному потоку на піксель цільового рендеру. Розмір групи потоків за замовчуванням заданий в шаблоні обчислювального шейдера Unity, дорівнює

[numthreads (8,8,1)], тому було вирішено дотримуватися його і створити по одній групі потоків на кожні 8×8 пікселів. В кінці результат потрібно записати на екран за допомогою Graphics.Blit().

На рис. 1 подано процес обчислень, що поступово розділяється на потоки, починаючи від основного потоку виконання Unity скрипта (який виконується на CPU), доки на обчислення кожного пікселя не буде виділено по потоку GPU.

Такий спосіб розпаралелювання і нумерації потоків дуже зручний, адже кожен піксель, на який виділено потік у групі потоків, дізнається координати потоку й завдяки цьому отримує свої координати на текстурі, яка рендериться. Зазвичай ці координати називають UV. Після закінчення обчислень на потоці результат, який є кольором, записується в призначене місце в текстурі відповідно до координат свого потоку.

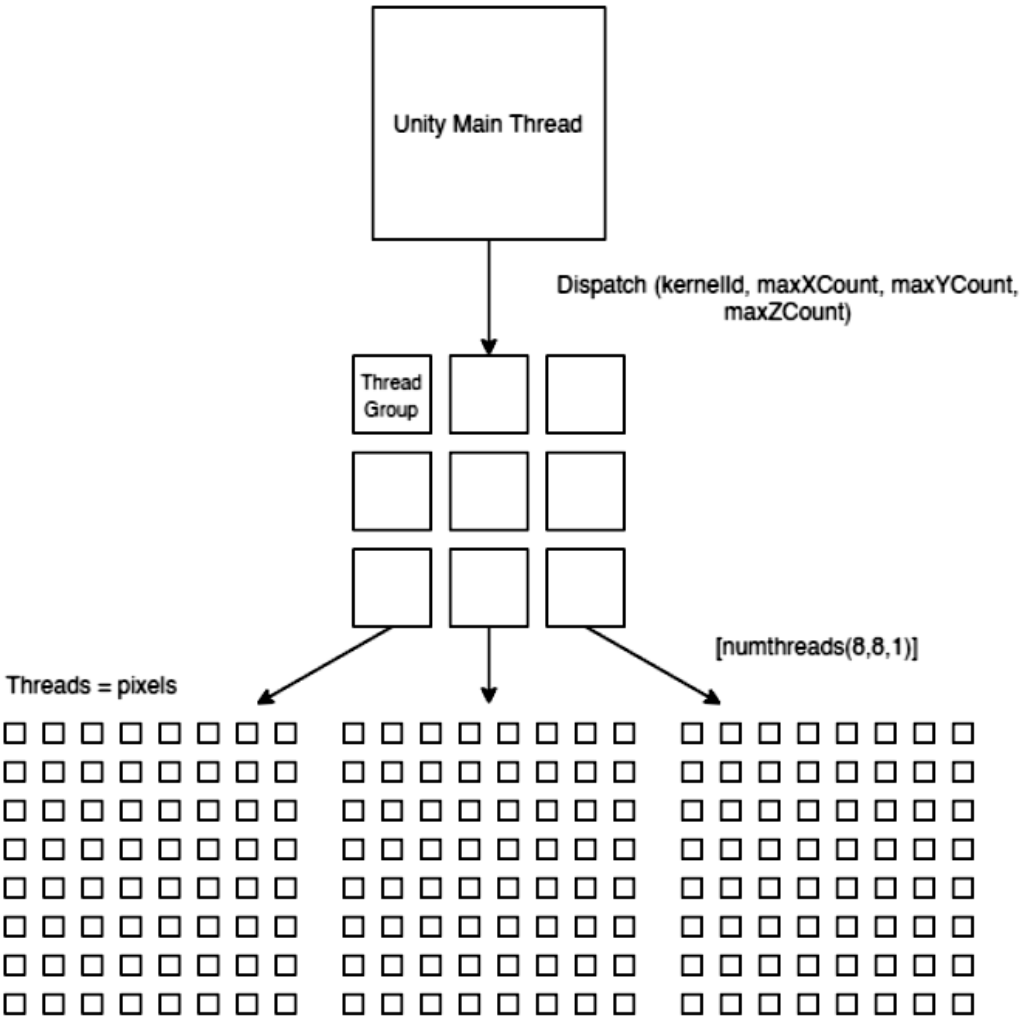

Рис. 1. Схема розпаралелювання процесів

Аналіз останніх досліджень та публікацій

Йохен Хунц [4] проаналізував доцільність використання обчислювальних шейдерів для імплементації складних алгоритмів. Він розглядає структуру мови, яка використовується для їх написання, будову шейдера та потенціал його програмування. Висновки автора ґрунтуються на імплементації обчислювальних шейдерів для OpenGL, яка використовує мову GLSL. Для демонстрації дослідник імплементував декілька симуляцій фізичних процесів. Автор акцентує на використанні спільної пам’яті шейдера та її ефективності в імплементації алгоритмів. У висновках дослідник зазначив, що обчислювальні шейдери добре підходять для фізичних симуляцій.

У статті [5] Космін-Константина Міхая розглянуто доцільність розподілення процесів між процесором та графічним ядром, зокрема, через використання обчислювальних шейдерів. Головною метою дослідника був аналіз можливості використання графічного ядра для імплементації мультимодельної адаптивної системи надійного контролю в реальному часі. Автор дійшов висновку, що використання графічного процесора доцільне за наявності великої бази даних, яку можна опрацьовувати паралельно.

Формулювання цілі статті

Мета статті – аналіз способів побудови процесів віртуальної симуляції та оптимізації за допомогою розпаралелення задач для виконання на численних потоках відеоядра, що дає змогу моделювати стан системи в будь-який момент у режимі реального часу.

Реакційно-дифузійна модель Грея – Скотта

Система Грея – Скотта є реакційно-дифузійною системою [6]. Це означає, що вона моделює процес, який складається із реакції та дифузії, зокрема хімічну реакцію між двома речовинами A та B, обидві з часом дифундують. Під час реакції A витрачається, а B утворюється. Щільності речовин A і B відіграють важливу роль у моделюванні.

Система характеризується двома параметрами: F – швидкість, з якою A поповнюється, і k – швидкість, з якою B видаляється з системи. Зміна цих параметрів призводить до широкого спектра цікавих візерунків, деякі з яких видаються цілком знайомими.

Система Грея – Скотта моделює хімічну реакцію:

$$A + 2B \rightarrow 3B. \quad (1)$$

Ця реакція споживає A і виробляє B. Тому для підтримки реакції необхідно контролювати кількість обох речовин. Це здійснюють, додаючи A за параметром F, або "швидкістю подавання" (*feed rate*), і видаляючи B за k – "швидкістю знищення" (*kill rate*). Видалення B також можна описати іншою хімічною реакцією:

$$B \rightarrow P, \quad (2)$$

де P – інертний продукт, тобто не реагує. У цьому випадку параметр k контролює швидкість реакції.

Обидві речовини з часом дифундують зі швидкістю дифузії D_A (швидкість дифузії речовини A) та D_B (швидкість дифузії речовини B). У цьому моделюванні D_A є подвоєним D_B.

Контролюючи швидкість подавання F (*feed rate*) та швидкість знищення k (*kill rate*), можна симулювати різні реакційно-дифузійні процеси для будь-яких речовин.

Система Грея – Скотта визначається двома рівняннями, які описують поведінку двох речовин, які реагують:

$$a' = -ab^2 + F(1 - a) + D_a \Delta a, \quad (3)$$
$$b' = ab^2 + (F + k)\, b + D_b \Delta b.$$

Змінними a і b у цих рівняннях є концентрації двох реагентів A і B. У лівій частині кожного рівняння є похідна за часом однієї з цих концентрацій, що описує швидкість, з якою вона змінюється. Обидві праві частини рівнянь містять три окремі доданки. Перший описує реакцію між двома речовинами. Оскільки один A і два B реагують, у відповідний доданок входить a в першому степені й b у другому степені: ab^2. У зв'язку зі споживанням речовини A реакцією терм має від'ємний знак у першому рівнянні. У другому рівнянні його знак додатний, оскільки B утворюється у результаті реакції.

Другий член першого рівняння описує швидкість, з якою поповнюється A ззовні системи. Це необхідно, оскільки в іншому випадку A просто була б використана повністю і реакція б припинилася. Швидкість подавання задано параметром F. F множать на $(1 - a)$, щоб гарантувати, що a поповнюється зі швидкістю, яка залежить від поточної концентрації, що ніколи не перевищує 1. B не потребує поповнення, оскільки утворюється в реакції. Натомість її потрібно видалити, щоб підтримувати реакцію. Швидкість видалення ("швидкість знищення") контролюється параметром k. Щоб видалити B швидше, ніж додається A, k додають до F і множать на b, оскільки видалення B також має залежати від його концентрації [7].

Останні члени в обох рівняннях Δa та Δb описують дифузію A і B відповідно. Ця дифузія означена різницею між середньою зваженою сумою концентрацій сусідніх комірок та самою коміркою. Під “середньою зваженою сумою” необхідно розуміти суму концентрацій сусідів, помножену на маску, яка, своєю чергою, слугує ваговим коефіцієнтом. Форма цієї маски може бути різною, від неї залежить перебіг дифузії. У нашому випадку використано маску оберненої пропорційності відстані від центра.

Імплементація. Головним завданням імплементації моделі Грея – Скотта є створення наочного зображення рівня концентрації обох речовин на площині. Оскільки симуляція здійснюється на обчислювальному пристрої, то показати це поле концентрації в аналоговому вигляді неможливо. Тому хорошою альтернативою є створення поля із комірок з набором даних про концентрацію речовин, а саме: концентрація першої речовини, концентрація другої речовини та дельти концентрацій до відповідних речовин. Оскільки для кожної комірки є чотири параметри, то такі комірки можна закодувати у пікселі текстури, кожен з яких для системи є набором з чотирьох чисел з плаваючою комою, які зазвичай репрезентують три компоненти кольору (червоний, зелений, синій) та показник прозорості. Всі компоненти містяться у діапазоні від 0 до 1, що ідеально підходить для задавання концентрацій речовин у кожній точці площини. Нуль означатиме повну відсутність речовини, а одиниця – максимальну концентрацію.

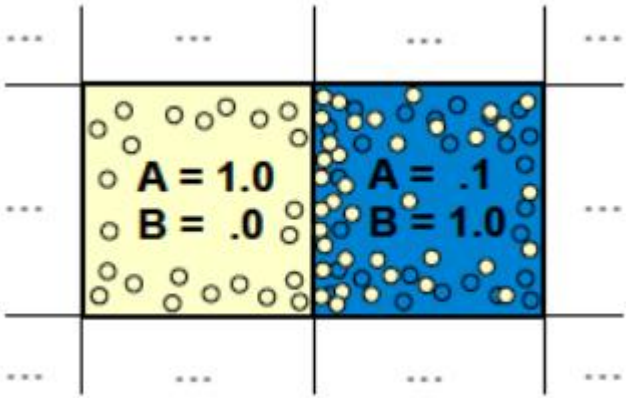

Рис. 2. Схематичне зображення концентрації речовини в комірках даних (пікселях)

Оскільки модель Грея – Скотта передбачає, що площина симуляції початково заповнена першою речовиною, а деякі ділянки заповнені другою речовиною певної концентрації, то такі дані для описання початкового стану системи можна легко задати у вигляді текстури, яку перед першим кроком симуляції завантажено в обчислювальний шейдер.

Далі на кожному кроці симуляції кожна комірка оглядатиме своїх сусідів у певному радіусі, тому необхідно ізолювати результати обчислень від вхідних даних. Для цього потрібно мати два набори комірок: перший для збереження вхідних даних, а другий для збереження результатів. Такий підхід у графіці називається буферизацією, а допоміжна текстура – буфером.

Якщо необхідно прискорити симуляцію, виконується декілька ітерацій симуляції за одиницю часу. Такою одиницею часу в графіці прийнято вважати час рендеру кадра (далі – фрейм). А отже, щоб зекономити ресурси системи, показувати результат симуляції варто тільки один раз за фрейм. Тому для цього знадобиться ще одна допоміжна текстура, в якій буде міститися результат останньої ітерації симуляції цього фрейма, яку і можна вивести на екран. Також ця окрема текстура дасть змогу маніпулювати даними перед виведенням, що дозволить по-різному інтерпретувати результат обчислень.

У ході дослідження було прийнято рішення організовувати дані про стан системи за допомогою текстур. Такий підхід вписується у первинну ідею використання обчислювальних шейдерів для виконання симуляції. По-перше, оскільки шейдери були створені, щоб надати розробникам можливість працювати із текстурами, то вони наділені повним інструментарієм для роботи з текстурами загалом і кожним пікселем окремо. По-друге, оскільки вхідні дані обчислення відділені

від вихідних даних алгоритму в різні текстури, то цей факт автоматично робить процес обчислення нового стану кожного пікселя незалежним від будь-якого іншого пікселя системи. Тобто такий процес може легко виконуватися на паралельному потоці як абсолютно незалежний. А як вказано вище, виконання на графічному ядрі істотно пришвидшує алгоритми, які можна розділити на якомога більше простих однакових процесів.

Основою процесу симуляції стане імплементація формули (3). Розпочати варто з того, щоб отримати поточне значення концентрацій у конкретній точці (пікселі), беручи до уваги, що перший компонент кольору (червоний канал) відповідає концентрації речовини A, а другий компонент (зелений канал) відповідає концентрації речовини B, тобто коефіцієнти a і b.

Наступний крок найтрудомісткіший для обчислення. Це обчислення значень Δa і Δb. Його значення обчислюють як так званий оператор Лапласа [8], який дає різницю між середнім значенням сусідніх комірок та заданою коміркою. Це значення і відповідає за перебіг дифузії, тому що комірка стає все більше схожою на своїх сусідів. Для цього необхідно перевірити всі комірки поряд, у заданому радіусі. Такий радіус було введено як параметр, що можна налаштовувати для кожної реакції. Якщо комірка проходить перевірку на перебування на достатньо маленькій відстані від обчислюваної комірки (відстань між ними у пікселях менша і дорівнює параметру радіуса дифузії), то значення її кольору і вагового коефіцієнта будуть враховані в сумі значень каналів і сумі вагових кофіцієнтів, які, своєю чергою, в результаті ділення і віднімання значення поточного стану комірки по всіх каналах дадуть значення Δa і Δb.

Після цього отриманий результат можна використати для підрахунку значень змін у кожному з каналів, що відповідає за концентрацію. Отримані значення стануть новими значеннями каналів b (канал синього кольору) і a (коефіцієнт прозорості) відповідно.

В обчисленні також беруть участь такі конфігуровані параметри, як diffuseRateA і diffuseRateB. Ці параметри дають змогу імітувати реакційно-дифузійні процеси різних речовин, достатньо лише задати відповідні коефіцієнти швидкості реагування для кожної речовини і спостерігати. Також конфігурованими параметрами є “швидкість подавання” (feed rate) і “швидкість знищення” (*kill rate*), які детально описано вище.

Результатом роботи шейдера стала текстура, в кожному пікселі якої закодована інформація про концентрації речовин у цій точці, а також значення зміни концентрації за останню ітерацію симуляції. Якщо одночасно візуалізувати всі ці дані в такому вигляді, в якому вони зберігаються, а саме в кольорі пікселя, то така інтерпретація даних буде занадто переватаженою і незручною для аналізу. Зробить зручнішим використання цієї симуляції розділення даних під час візуалізації, метою якого є зосередження на одному із аспектів отриманого результату. Тому слід впровадити різні режими виведення. Ці режими є арбітражними, їх імплементація не зумовлена ніякими моделями.

- **AB**. Найпростіший режим виведення, за якого на екрані можна споглядати лише концентрації обох речовин. Значення концентрацій записані в перші два канали кольору, а саме червоний та зелений відповідно.
- **Greyscale.** У цьому режимі показано концентрацію першої речовини порівняно з другою, що визначена різницею між A та B, яка записана у всі три канали кольору, тому така репрезентація виходить чорно-білою. Білий колір означає максимальну концентрацію речовини A і відсутність речовини B. Чорний колір означає або відсутність речовини A, або відсутність обох речовин.
- **Colored.** Найреалістичніший режим, який відображає концентрацію обох речовин, присвоюючи кожній із них деякий колір. У клітинках зі змішаним вмістом колір буде відповідний змішуванню цих кольорів залежно від пропорцій, тобто наближатися до того кольору, концентрація речовини якого є більшою.
- **Delta.** Режим, який комбінує виведення усіх параметрів клітинки, завдяки тому, що концентрація речовини A вважається відсутністю кольору, тобто чорним кольором, а отже, залишає вільними для виведення на екран три канали, які цей режим використовує, для візуалізації концентрації речовини B (червоний), значення зміни речовини A (зелений) та значення зміни речовини B (синій). Проте значення зміни концентрацій обох речовин доволі маленькі, тому потрібно їх привести у той діапазон, коли значення цих каналів дасть достатньо контрастний колір, щоб візуально розпізнавати його.

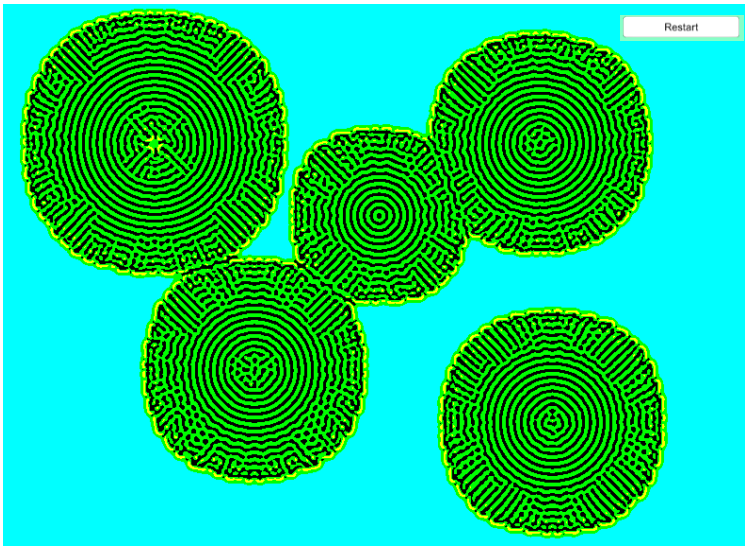

Рис. 3. Результати роботи реакційно-дифузійної моделі у режимі AB

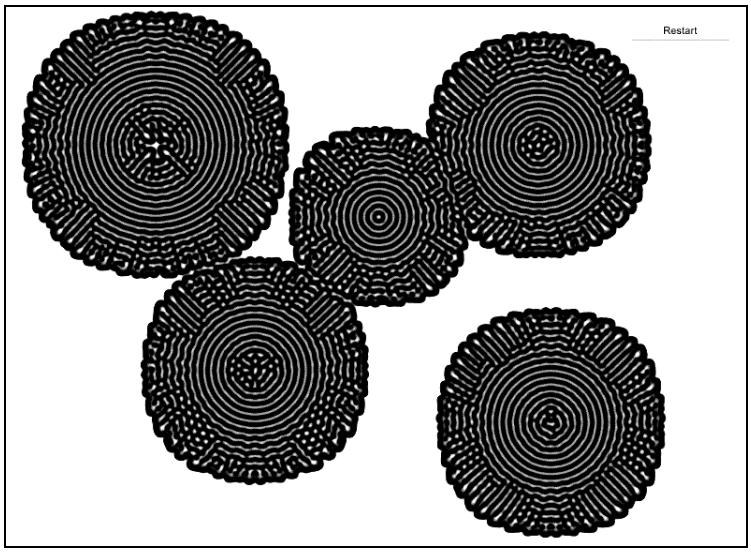

Рис. 4. Результати роботи реакційно-дифузійної моделі у режимі Greyscale

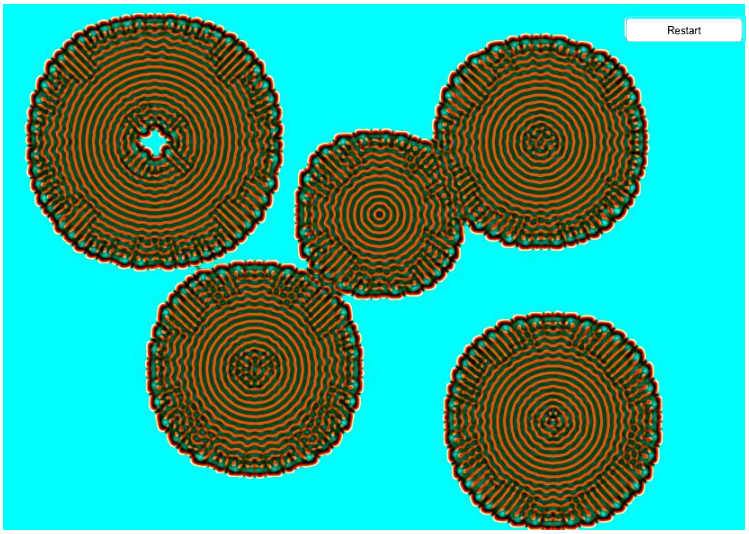

Рис. 5. Результати роботи реакційно-дифузійної моделі у режимі Colored

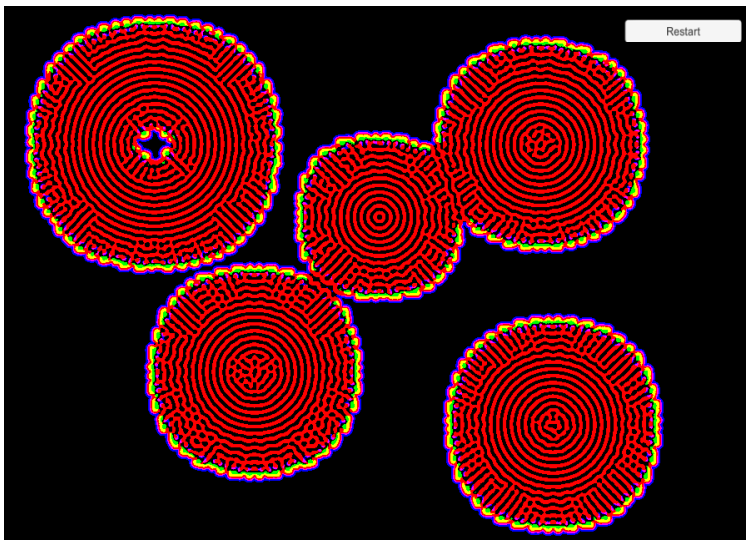

Рис. 6. Результати роботи реакційно-дифузійної моделі у режимі Delta

У демонстраційних матеріалах як вхідні дані використано зображення розміром 1024×768 пікселів, тобто на відеоядро створювали і запускали 786 432 потоків для обчислення стану системи для кожної ітерації симуляції. Таких запусків здійснювалось 60 за секунду. Цей метод оптимізації дав змогу запускати симуляції та в режимі реального часу спостерігати за зміною візерунка реакції. Тобто за відведену одиницю часу 1/60 с всі величини, що описують стан комірок даних, встигали обчислювати.

Модель Грея – Скотта описує реакції, які часто трапляються у реальному світі. Наприклад, деякі етапи в ланцюгу гліколізу, який перетворює глюкозу на енергію в нашому тілі, відбуваються саме за цією моделлю. Іншими прикладами є деякі специфічні реакції ферментації та бродіння, під час яких виділяються такі коензими, як нікотинамід аденіндинуклеоти́д. Також такі реакції відбуваються у процесах згортання крові [9]. Така різноманітність хімічних процесів, що працюють за цим принципом, вказує на високу актуальність досліджень симуляцій у цій галузі, зокрема імплементації, особливо якщо природа формування чудернацьких візерунків, які формує ця симуляція, ще не до кінця досліджена [10]. Досі до кінця не відомо, чому “плями” рухаються по площині й ростуть у напрямку, перпендикулярному до свого руху, потім діляться навпіл, і нові плями починають рухатися у протилежних напрямках.

Тривимірна модель поведінки зграї

Зімітована зграя – це вдосконалення системи частинок, в якій імітовані птахи є частинками. Сукупний рух імітованої зграї створюється розподіленою поведінковою моделлю, яка функціонує за тим самим принципом, за яким діють природні зграї; птахи самі вибирають свій шлях. Кожен змодельований птах реалізується як незалежний актор [11], який орієнтується відповідно до свого локального сприйняття динамічного середовища, законів імітованої фізики, які керують її рухом, і набору поведінок, які запрограмував у неї “аніматор”. Сукупний рух імітованої зграї є результатом щільної взаємодії порівняно простих поведінок окремих імітованих птахів [12].

Тож необхідно дати визначення: зграя як модель – це система із однотипних птахоподібних об’єктів, в якій всі об’єкти рухаються щодо заданих для них законів. Відмінність між зграєю і отарою полягає у тому, що зграя – це система в тривимірному просторі, а отара – в двовимірному. Оскільки у цій статті ми плануємо продемонструвати можливості побудови симуляцій в тривимірному просторі, то надалі йтиметься лише про зграї.

Зграя демонструє багато контрастів. Вона складається з окремих птахів, але загальний рух здається плавним; вона проста за концепцією, але настільки візуально складна, що здається випадковим розташуванням і водночас чудово синхронізованою системою. Можливо, найбільше спантеличує сильне враження навмисного централізованого контролю. Проте всі дані вказують на

те, що рух зграї має бути лише сукупним результатом дій окремих тварин, кожна з яких діє тільки на основі свого локального сприйняття світу [13].

Однією зі сфер зацікавлення в комп'ютерній анімації є описання і контроль усіх типів руху. Комп'ютерні аніматори прагнуть як винайти абсолютно нові типи абстрактного руху, так і дублювати (або вносити зміни) рухи, які трапляються у реальному світі. На перший погляд, створення анімаційного комп'ютерного графічного зображення зграї птахів супроводжується значними труднощами. Скриптування шляху великої кількості окремих об'єктів за допомогою традиційних методів комп'ютерної анімації було б утомливим. Враховуючи складні шляхи, якими рухаються птахи, сумнівно, що цю специфікацію можна було б виконати безпомилково. Навіть якби можна було б описати розумну кількість відповідних шляхів, малоймовірно, що обмеження руху зграї можна було б зберегти (наприклад, запобігти зіткненням між усіма птахами у кожному кадрі). Нарешті, зграю, написану у такий спосіб, буде важко редагувати (наприклад, змінити рух усіх птахів для частини анімації). Неможливо створити сценарій руху зграї, але для ефективної, надійної та правдоподібної анімації зграй та пов'язаних групових рухів потрібен кращий підхід.

Змодельована зграя, описана й імплементована, тісно пов'язана із системами частинок [14], які використовують для подання динамічних "нечітких об'єктів" неправильної та складної форми. Системи частинок використано для моделювання вогню, диму, хмар, а останнім часом – бризу і піни океанських хвиль [15]. Системи частинок – це сукупність великої кількості окремих частинок, кожна з яких має власну поведінку. Частинки створюються, старіють і відмирають. Протягом їхнього життя їм притаманна певна поведінка, яка може змінити власний стан частинки, що складається із кольору, прозорості, розташування та швидкості.

В основі моделі птахоподібних об'єктів – невелике узагальнення систем частинок. У тому, що можна було б назвати "системою підоб'єкта", точкові частинки замінюються цілим геометричним об'єктом, що складається із повної локальної системи координат і посилання на модель геометричної форми. Використання фігур замість точок є візуально значущим, але принциповіша відмінність полягає у тому, що геометричний стан окремих підоб'єктів складніший; вони тепер мають орієнтацію.

Інша відмінність між птахоподібними об'єктами і системами частинок не так чітко визначена. Поведінка перших, як правило, складніша, ніж поведінка частинок, що описано в літературі. Сучасна модель поведінки птахоподібних об'єктів може бути приблизно на один–два порядки складнішою, ніж типова поведінка частинок. Однак це відмінність за ступенем, а не характером. І жодна імітована поведінка не є такою складною, як поведінка справжнього птаха.

Крім того, як показано, частинки в системах частинок не взаємодіють одна з одною, хоча це не є неможливим за визначенням. Але птахи, а отже, і птахоподібні об'єкти повинні тісно комунікувати, щоб правильно взаємодіяти під час польоту. Поведінка птахоподібного об'єкта залежить не тільки від внутрішнього, а й від зовнішнього стану [16].

Фундаментальною частиною моделі об'єкта зграї є геометрична здатність літати. Рух членів змодельованої зграї чи отари можна вважати різновидом "польоту", приховуючи значні тонкощі рухів крил, плавців і ніг (а у випадку отари – обмежуючи свободу руху в третьому вимірі), оскільки ці рухи становлять мало цінності для зграї загалом і не впливають на решту системи. У цій роботі термін "геометричний політ" стосується певного типу руху за траєкторією: динамічне, поетапне, стале геометричне перетворення об'єкта, що рухається вздовж і дотикається до тривимірної кривої. Рух є сталим, тоді як базова геометрична модель об'єкта може вільно формулювати або змінювати форму в цій "системі координат, що літає". На відміну від типовішого анімованого руху вздовж попередньо визначених сплайнових кривих, форма траєкторії польоту не вказується заздалегідь.

Щоб побудувати змодельовану зграю, варто почати з моделі птахоподібного об'єкта, яка підтримує геометричний політ. Додано поведінку, яка відповідає протилежним силам уникнення зіткнення та бажанням приєднатися до зграї. Коротко викладено правила поведінки, яка призводить до імітації зграї, за зменшенням пріоритету:

1. **Уникнення зіткнень.** Потреба в уникненні зіткнень із сусідніми партнерами по зграї.

2. **Узгодження швидкості.** Необхідність порівнювати швидкість із сусідніми партнерами по зграї.

3. **Центрування зграї.** Намагання залишатися поруч із сусідніми партнерами по зграї.

Швидкість – це векторна величина, яка дорівнює відношенню напрямку та швидкості. Спосіб узгодження та поєднання наслідків кожного із цих видів поведінки проаналізуємо детальніше пізніше. Аналогічно, відстані між об'єктами під час обчислення цих правил є ключовими для визначення поведінки зграї. Цей аспект також проаналізуємо детальніше далі, але зазвичай усвідомлення одним птахоподібним об'єктом іншого ґрунтується на відстані та напрямку вектора зміщення кожного з них.

Уникнення статичного зіткнення та динамічне узгодження швидкості є взаємодоповнюваними. Разом вони гарантують, що члени імітованої зграї можуть вільно літати в переповненому небі всередині зграї, не натикаючись один на одного. Уникнення зіткнення – це прагнення ухилитися від неминучої колізії. Уникнення статичних зіткнень основане на відносному положенні партнерів по зграї та ігнорує їх швидкість. І навпаки, узгодження швидкостей ґрунтується лише на швидкості та ігнорує положення. Це прогнозована версія уникнення зіткнення: якщо птахоподібний об'єкт добре порівнює швидкість зі швидкістю сусідів, малоймовірно, що він зіткнеться із будь-яким із них найближчим часом. У разі узгодження швидкостей відстань між птахоподібними об'єктами залишається приблизно незмінною щодо поточного геометричного польоту. Уникнення статичного зіткнення потрібне для встановлення мінімально необхідної дистанції; узгодження швидкості має тенденцію підтримувати його.

Центрування зграї змушує птахоподібний об'єкт бути ближче до центра зграї [17], оскільки кожен елемент зграї має деякий локальний радіус сприйняття світу. "Центр зграї" насправді означає центр сусідніх партнерів по зграї. Центрування зграї змушує птахоподібний об'єкт летіти в напрямку, який переміщує його ближче до центроїда сусідніх птахоподібних об'єктів. Якщо птахоподібний об'єкт глибоко всередині зграї, щільність об'єктів навколо нього приблизно однорідна; потяг щодо переміщення птахоподібного об'єкта приблизно однаковий в усіх напрямках. У цьому випадку центроїд сусідніх птахоподібних об'єктів міститься приблизно в центрі групи, тому потяг до центрування зграї невеликий. Але якщо птахоподібний об'єкт на межі зграї, його сусідній птахоподібний об'єкт перебуває з одного боку. Центр групи птахоподібних об'єктів зміщений від центра околиці до тіла зграї. Тут потяг до центрування зграї сильніший, і траєкторія польоту буде дещо відхилена в бік місцевого центра зграї.

Справжні зграї іноді розділяються, щоб обійти перешкоду. Щоб бути реалістичною, імітована модель зграї також повинна мати цю здатність. Правильне центрування зграї дає змогу змодельованим зграям роздвоюватися. Доки окремий птахоподібний об'єкт може залишатися поруч зі своїми сусідами, він не зважає на те, чи наштовхнеться на перешкоду решта особин зграї. Спрощеніші моделі, запропоновані для імітації організації зграї (наприклад, модель центральної сили або слідування моделі призначеного лідера), не допускають розколів.

Кожна із трьох поведінкових компонент, пов'язаних зі зграюванням, дає окремі пропозиції щодо того, яким шляхом спрямувати об'єкт зграї. Вони виражаються як запити на прискорення. Кожен такий компонент поведінки говорить: "Якби я був головним, я б прискорився в цьому напрямку". Запит на прискорення формує тривимірний вектор, який, за системною конвенцією, обрізається до одиничної величини або менше. Кожна поведінка має декілька параметрів, які контролюють її функцію; один – це "сила", дробове значення від нуля до одиниці, яке може ще більше послабити запит прискорення. Навігаційний модуль головного мозку має збирати всі відповідні запити на прискорення, а потім визначати єдине "бажане" прискорення. Воно повинно поєднувати, розставляти пріоритети та вирішувати потенційно суперечливі потяги. Пілотний модуль приймає рішення про необхідність прискорення, визначеного навігаційним модулем, і передає його льотному модулю, який спрямовує політ у цьому напрямку.

Найпростіший спосіб – об'єднати запити на прискорення і скласти їх разом. Оскільки в цій операції використовуються коефіцієнти сили впливу, послуговуємося зваженою сумою. Так можна відобразити силу впливу поведінки однієї особини на інших, проте цей зв'язок є ненадійним та

його важко регулювати. Аналіз поведінки попередньої версії моделі птахоподібного об'єкта засвідчив, що навігація за допомогою лише зваженої суми запитів на прискорення доволі добре працює.

Найцікавіший рух імітованої зграї відбувається у результаті взаємодії з іншими об'єктами навколишнього середовища. Ізольована поведінка зграї має тенденцію досягати стійкого стану. Проте перешкоди із навколишнього середовища та спроби птахоподібного об'єкта орієнтуватися на рух навколо них посилюють складність поведінки зграї.

Ідея полягає у проєктуванні променів, що беруть початок з голови птахоподібного об'єкта і проєктуються вперед у напрямку зору. Якщо перший такий промінь, який рухається прямо, буде натикатися на перешкоду, то наступний промінь відхилятиметься під деяким кутом і знову перевірятиметься, чи цей напрямок є безперешкодним. І так доти, доки не знайдеться потрібний шлях. На площині це завдання здається простим (рис. 7).

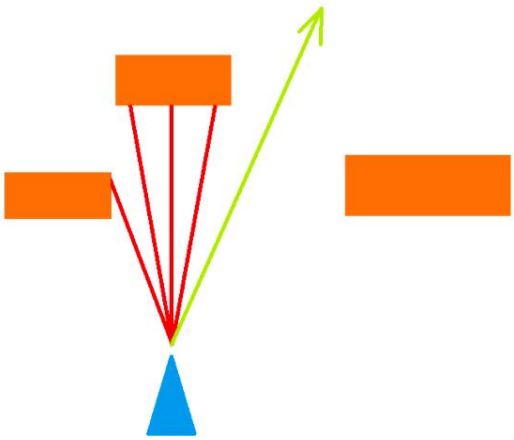

Рис. 7. Схематичне зображення принципу пошуку напрямку для уникнення зіткнень на площині

Проте пошук усіх таких напрямків для тривимірного простору дещо ускладнюється. Спершу варто зробити відступ від елемента зграї на відстань, яка має бути зоною недосяжності об'єкта для колізії, й описати сферу навколо нього. Тоді можна побачити, що завдання зводиться до пошуку рівномірно розподілених точок на сфері. Елегантним вирішенням такого завдання є сфера Фібоначчі.

Накладання решітки Фібоначчі (вона ж "золота спіраль" або "сфера Фібоначчі") на поверхню сфери – надзвичайно швидкий і ефективний наближений метод рівномірного розподілу точок на сфері [18]. Для додаткового прискорення обчислень класичне розв'язання цієї задачі виконане в обчислювальному шейдері, та цей процес відбувається лише один раз – під час ініціалізації сцени. Результат знаходження всіх точок надалі використовується для кожного учасника зграї відносно центра об'єкта. В кожному напрямку по черзі пускають не промінь, а сферу. Радіус цієї сфери повинен бути достатнім, щоб перевірити, чи зможе об'єкт оминути перешкоду, не зачепившись боком.

Навіть якщо вважати, що будь-який елемент зграї не знає нічого про положення, стан, швидкість і напрям всіх інших елементів зграї у будь-який момент часу, алгоритм роботи все-таки передбачає огляд усіх інших об'єктів зграї, щоб оцінити параметри симулювання, що додає обчислювальної роботи [19]. Тобто складність алгоритму становить $O(N^2)$. Щоб симулювати роботу великої кількості елементів системи (кілька тисяч), застосовано алгоритми обчислення з використанням графічного ядра. Така ситуація якнайкраще підходить для симуляції поведінки саме за допомогою відеокарти, адже на момент аналізу поведінка сусідніх елементів не змінюється. Лише після аналізу поведінки всіх учасників зграї дані про їх стан у системі оновлюються і вони самі приймають рішення про подальші дії, із урахуванням даних аналізу оточення зграї. Отже, вдається уникнути ситуації, коли для деяких птахоподібних об'єктів виконано обчислення нового (для цього фрейму) стану і нового напрямку, а для інших – ні. При цьому всі елементи зграї можна розглядати як незалежні ізольовані складові системи.

Отже, кожному птахоподібному об'єкту буде виділено паралельний потік графічного ядра, на якому алгоритм розглядатиме всіх інших птахів у зграї й оцінюватиме, чи вони достатньо близько, щоб потрапляти у поле зору, і відповідно, чи кожен з них впливатиме на поведінку цього птахоподібного об'єкта.

На початковому етапі варто перевірити, чи перебуває порівнюваний елемент у радіусі зору/відчуття поточного елемента. Якщо так, то він вважає його учасником своєї "підзграї". "Підзграєю" вважатимемо сукупність елементів, які містяться у радіусі видимості поточного об'єкта, які надалі впливатимуть на його поведінку (він тяжітиме до центра цієї "підзграї" і визначатиметься із на-

прямком руху в той самий бік, що і його товариші). Крім того, поточний елемент розглядатиме своїх партнерів по “підзграї” й остерігатиметься зіткнення з ними, тож він одночасно і намагатиметься триматись від них на дистанції. Як бачимо на рис. 8, всередині відокремленої групи елементів зграї (які плавають) кожен встановлює “зв’язок підзграї” з одним – п’ятьма найближчими сусідами, які, своєю чергою, вважають своїх найближчих сусідів своєю “підзграєю”, і, зв’язуючи всіх учасників згуртованої групи в мережу, яка прямує в одну сторону, разом оминає перешкоди, якщо на всю групу вистачить місця, і т.д.

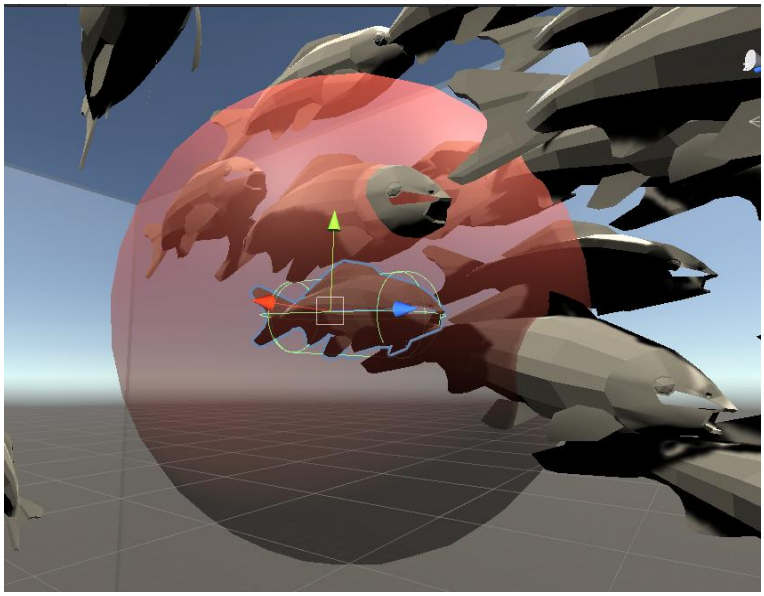

Рис. 8. Візуалізація перевірки відстані до сусідніх елементів зграї

Такий підхід дасть змогу звільнити основний потік виконання від необхідності розраховувати відстані до всіх інших птахоподібних об’єктів та кількість найближчих сусідів, напрям “пізграї”, центр “підзграї” та напрям, за яким потрібно уникати зіткнення з найближчими сусідами. Пізніше всі ці дані, обчислені на кожному фреймі, будуть ініціалізовані кожному елементу зграї, щоб він самостійно приймав рішення про подальшу поведінку відповідно до моделі поведінки актора.

Зважаючи на зазначені вище фактори та наведені переваги виконання алгоритмів на графічному ядрі, може постати питання: чому просто не запускати всі процеси й алгоритми виконуватися на відеокарті? Адже виграш у швидкості обчислень такий значний. Проблема полягає в тому, що графічний процесор не вміє інтерпретувати об’єктно-орієнтовані мови, які можна інтерпретувати до виконання на процесорі. Саме через іншу будову і принципи роботи відеокарта і процесор є різними комплектуючими обчислювальної машини [20].

Відповідно, рушій Unity побудований так, щоб основні фізичні процеси за участю бібліотек Unity та взаємодія з об’єктами повинні відбуватися на основному потоці центрального процесора. Водночас скрипти для графічного процесора виконуються як допоміжні обчислювальні елементи додатка, на які варто покладати лише частину функцій, і ніяк не можуть слугувати каркасом архітектури застосунку.

Тож основою архітектури застосунку став клас FlockManager. Цей клас комунікує з іншими функціональними одиницями, які повертають йому результати виконання покладеної на них роботи. Крім того, цей клас розподіляє дані між об’єктами зграї, які уже самостійно приймають рішення про подальшу поведінку і рух, із урахуванням свого положення у сцені в кожний момент часу.

Першим класом, з яким комунікує FlockManager під час ініціалізації, є Spawner, який створює і розставляє об’єкти сцени в заданій кількості у визначеному регіоні з випадково згенерованим поворотом. Ці елементи у вигляді списку повертаються FlockManager’у для подальшої ініціалізації.

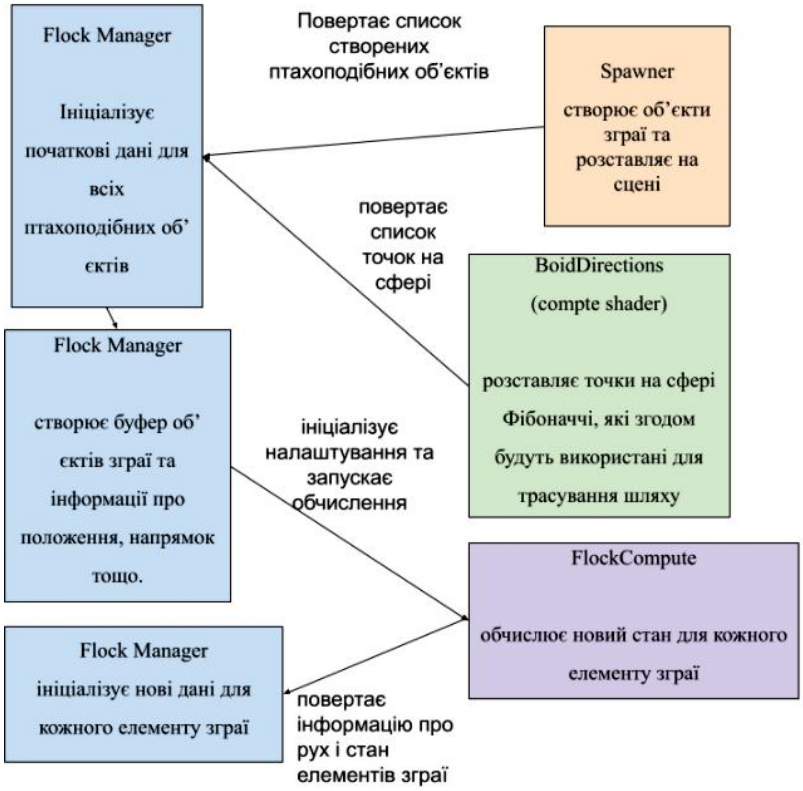

Рис. 9. Схема взаємодії класів та обчислювальних шейдерів

Також для ініціалізації елементів зграї задля їх подальшого орієнтування в просторі та оминання перешкод використовується результат обчислень сфери Фібоначчі. Ці дані, як уже зазначено вище, обчислюються у BoidDirections – обчислювальному шейдері. Разом з точками на сфері Фібоначчі на етапі ініціалізації задають такі початкові параметри птахоподібного об'єкта: початкова швидкість (визначається як середньоарифметична швидкість між мінімальною і максимальною допустимими швидкостями), початковий напрям руху (вперед відповідно до повороту), радіус чуття інших об'єктів зграї, радіус оминання, максимальну силу повороту, ваговий коефіцієнт вирівнювання напрямку, ваговий коефіцієнт гуртування, ваговий коефіцієнт уникнень партнерів по зграї. Разом з цими параметрами ініціалізуються також параметри маневрування між перешкодами: маска шарів об'єктів (список міток на об'єктах сцени, на які реагуватиме особина у разі наближення – проєкція сфери), параметр радіуса сфери, що використовується для оцінювання ймовірності колізії, ваговий коефіцієнт оминання перешкод, та відстань, яку проходить особина, реагуючи на перешкоду.

На початку кожного фрейму відбувається ініціалізація даних обчислювального шейдера FlockCompute. Дані про елементи зграї потрапляють у буфер, який ініціалізується FlockManager`ом в FlockCompute. Після цього запускається аналіз зграї для рекалькуляції інформації про зграю, яка відома кожному птахоподібному об'єкту. Після цьоого FlockManager запитує дані з перерахованого буфера і передає їх елементам зграї для подальшого опрацювання і прийняття рішення про подальшу поведінку. Тепер можна очистити дані буфера. Такий процес відбувається під час кожного обчислення кадру впродовж всього робочого циклу.

Висновки

У результаті дослідження можна зазначити, що використання графічного ядра задля розвантаження центрального процесора є дієвою стратегією, проте деякі завдання складно розпаралелити. Підтвердженням цього є досвід імплементації симуляцій, описаних у цій статті. Беручи до уваги цей досвід, можна стверджувати, що симуляція реакційно-дифузійної моделі є прикладом задачі, яка добре розпаралелюється. Про це свідчить той факт, що майже всю імплементацію симуляції виконано в обчислювальному шейдері. Водночас, симуляція дій зграї виявилась завдан-

ням, менш придатним для паралелізації. В її імплементації на графічний процесор вдалося перекласти лише частину роботи, однак та частина, яку вдалось перемістити на графічний процесор, була доволі трудомісткою, що істотно прискорило виконання обчислень.

Отже, це дослідження засвідчило, що можливість використання відеоядра для виконання завдань, не пов'язаних із графікою, сприяє істотному підвищенню швидкодії застосунків, зокрема наукових фізичних симуляцій, та уможливлює виконання їх у режимі реального часу.

Розроблені в ході досліджень симуляції можуть слугувати інструментами досліджень у низці галузей. Імплементовану модель Грея – Скотта можна застосовувати в дослідженні взаємодій хімічних елементів. Як ми зазначали, цей принцип покладено в основу багатьох процесів, що відбуваються в людському організмі. Не виключено, що використання розробленої симуляції буде корисним не лише в галузі теоретичної хімії, а й у медичній галузі.

Створена візуалізація процесів симуляції дій зграї наділена не меншим потенціалом. Вона може бути корисною біологам, що досліджують поведінку зграйових тварин, які рухаються у тривимірному просторі. Наприклад, риби і птахи. Ввівши деякі спрощення, можна перевести симуляцію у двовимірну площину і симулювати дії отари, наприклад, отари овець, кіз тощо. Модель можна застосовувати також в оборонному секторі, що дуже актуально в сучасних умовах. Процеси симуляції надають можливість спрогнозувати, в якій точці розміщуватиметься кожен літальний пристрій у будь-який момент часу після початку руху, рухаючись за описаними законами.

Розроблені процеси симуляції не є ідеальними повнофункціональними відтвореннями зображених процесів. Тож у подальших дослідженнях задля їх використання у дослідницькій діяльності потрібно буде провести консультації із фахівцями з предметної області щодо рекомендацій стосовно впровадження додаткових факторів впливу на процеси симуляції, щоб повною мірою наблизити їх до природних процесів або додатково дослідити вплив цих факторів на модель. Наприклад, подальшим кроком для покращення процесів симуляції дій зграї буде впровадження розрахунків кута повороту об'єкта в просторі та врахування форми об'єкта задля точнішого уникнення колізій об'єктів всередині зграї. У подальших дослідженнях плануємо використовувати методи симуляції для моделювання освітніх процесів.

Список літератури

1. Lammers, K. (2013). Unity Shaders and Effects Cookbook. Packt Publishing, 268 p.

2. Doppioslash, C. (2017). Physically Based Shader Development for Unity 2017: Develop Custom Lighting Systems. Apress. 255 p.

3. Marschner, S., Shirley, P. (2021). Fundamentals of Computer Graphics. Peters/CRC Press 716 p.

4. Hunz, J. (2013). The Possibilities of Compute Shaders – an Analysis. Koblenz, 60 p.

5. Cosmin-Constantin, M., Ciprian L. (2021). Using Graphics Processing Units and Compute Shaders in Real Time Multimodel Adaptive Robust Control. *Electronics,* 10, 24–62. DOI:10.3390/10202462.

6. Erban, R., Chapman, J. (2020). Stochastic Modelling of Reaction-Diffusion Processes. Cambridge University Press. DOI: 10.1017/9781108628389.

7. Gray P., Scott, S. K. (1983). Autocatalytic reactions in the isothermal, continuous stirred tank reactor: Isolas and other forms of multistability. *Chemical Engineering Science*, 38, 29–43. DOI: 10.1016/0009-2509(83)80132-8.

8. Evans, L. (1998). Partial Differential Equations: Second Edition. *Graduate Studies in Mathematics*, Vol. 19; 2010; 749 p.

9. Newell, Alan C.; Whitehead, J. A. (1969). Finite bandwidth, finite amplitude convection. *Journal of Fluid Mechanics*. Cambridge University Press (CUP), 38, 279–303. DOI: 10.1017/s0022112069000176.

10. Ross, J., A. Arkin, S., Mueller, C. (1995). Experimental evidence of Turing structures. *Journal Phys. Chem*., 99, 10417–10419. DOI: 10.1021/j100025a051.

11. Reynolds. C. W. (1982). Computer Animation with Scripts and Actors, Computer Graphics, 16 (3), (acm SIGGRAPH `82 Proceedings), 289–296. DOI: 10.1145/965145.801293.

12. Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. SIGGRAPH '87: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques. Association for Computing Machinery*, 25–34. DOI: 10.1145/37401.37406.

13. Stephens, R. (2019). Essential Algorithms: A Practical Approach to Computer Algorithms Using Python and C#, John Wiley & Sons. 800 p.

14. Gille, W. (2020). Particle and Particle Systems Characterization. Small-Angle Scattering (SAS) Applications. Published by CRC Press 348 p.

15. Reeves, W. (1983) Particle Systems-A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics*, 2, 91–108. DOI: 10.1145/357318.357320.

16. Shaw, E. (1979) Fish in Schools. *Natural History*, 84 (8), 4046. DOI: 10.1243/0954410041322005.

17. Abelson, H., Sessa, A. (1981). Maneuvering a Three Dimensional Turtle in Turtle Geometry: The Computer as a Medium for Exploring Mathematics. The MIT Press, Cambridge, Massachusetts, 140–159.

18. Madarshahian, R., Hemez, F. (2021). Data Science in Engineering, IX, Springer Nature. https://www.codecademy.com/

19. Hewitt, C., Atkinson. R. (1977). Parallelism and Synchronization in Actor Systems. ACM Symposium on Principles of Programming Languages, 4, Los Angeles, California, 267–280.

20. Yuen, D., Wang, L., Chi X., Johnsson, L., Ge, W., Shi, Y. (2013). GPU Solutions to Multi-scale Problems in Science and Engineering, Springer Science & Business Media. DOI: 10.1007/978-3-642-16405-7_2.

References

1. Lammers, K. (2013). Unity Shaders and Effects Cookbook. Packt Publishing. 268 p.

2. Doppioslash, C. (2017). Physically Based Shader Development for Unity 2017: Develop Custom Lighting Systems. Apress, 255 .

3. Marschner, S., Shirle, P. (2021). Fundamentals of Computer Graphics. V. CRC Press, 716

4. Hunz, J. (2013). The Possibilities of Compute Shaders – an Analysis. Universitat Koblenz. 60

5. Cosmin-Constantin, M., Ciprian L. (2021). Using Graphics Processing Units and Compute Shaders in Real Time Multimodel Adaptive Robust Control. *Electronics,* 10, 24–62. DOI: 10.3390/10202462

6. Erban, R., Chapman, J. (2020). Stochastic Modelling of Reaction-Diffusion Processes. Cambridge University Press. DOI: 10.1017/9781108628389.

7. Gray P., S.K. Scott. (1983). Autocatalytic reactions in the isothermal, continuous stirred tank reactor: Isolas and other forms of multistability. *Chemical Engineering Science*, 38, 29–43. DOI: 10.1016/0009-2509(83)80132-8.

8. Evans, L. (1998). Partial Differential Equations: Second Edition. American Mathematical Society, 749 p.

9. Newell, Alan C.; Whitehead, J. A. (1969). Finite bandwidth, finite amplitude convection. *Journal of Fluid Mechanics*. Cambridge University Press (CUP), 38, 279–303. DOI: 10.1017/s0022112069000176.

10. J Ross, A. Arkin, S. C. Mueller. (1995). Experimental evidence of Turing structures. *J. Phys*. *Chem*., 99, 10417–10419. DOI: 10.1021/j100025a051

11. Reynolds. C. W. (1982). Computer Animation with Scripts and Actors, Computer Graphics, 16 (3), (ACM SIGGRAPH `82 Proceedings), 289–296. DOI: 10.1145/965145.801293

12. Reynolds, C.W. (1987). Flocks, herds and schools: A distributed behavioral model. SIGGRAPH '87: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques. Association for Computing Machinery*, 25–34. DOI: 10.1145/37401.37406.

13. Stephens, R. (2019). Essential Algorithms: A Practical Approach to Computer Algorithms Using Python and C#, John Wiley & Sons, 800

14. Gille, W. (2020). Particle and Particle Systems Characterization. Small-Angle Scattering (SAS) Applications. CRC Press, 348 p.

15. Reeves, W. (1983) Particle Systems-A Technique for Modeling a Class of Fuzzy Objects, ACM Transactions on Graphics, 2, 91–108. DOI:10.1145/357318.357320

16. Shaw, E. (1979) Fish in Schools, Natural History, 84 (8), 4046.

17. Abelson, H., Sessa, A. (1981). Maneuvering a Three Dimensional Turtle in Turtle Geometry: The Computer as a Medium for Exploring Mathematics, The MIT Press, Cambridge, Massachusetts, 140–159.

18. Madarshahian, R., Hemez, F. (2021). Data Science in Engineering, IX, Springer Nature. https://www.codecademy.com/

19. Hewitt, C., Atkinson. R. (1977). Parallelism and Synchronization in Actor Systems, ACM Symposium on Principles of Programming Languages, 4, Los Angeles, California, 267–280.

20. Yuen, D., Wang, L., Chi X., Johnsson, L., Ge, W., Shi, Y. (2013). GPU Solutions to Multi-scale Problems in Science and Engineering, Springer Science & Business Media. DOI: 10.1007/978-3-642-16405-7_2

MATHEMATICAL AND SOFTWARE FORMATION PROCESS OF SIMULATIONS ON A GRAPHICAL PROCESSING UNIT

Yulianna Kalynych[1], Yuriy Bilak[2], Ruslan Nebesnyi[3], Pavlo Fedorka[4]

[1, 2, 4] Uzhhorod National University,
[3] Ivan Pulyuy Ternopil National Technical University,
[1] e-mail: kalynych.yulianna@student.uzhnu.edu.ua, ORCID: 0000-0001-7102-4868,
[2] e-mail: yuriy.bilak@uzhnu.edu.ua, ORCID: 0000-0001-5989-1643,
[3] e-mail: nebesnyi@gmail.com, ORCID: 0000-0001-8886-8346,
[4] e-mail: fedorkapavlo@gmail.com, ORCID: 0000-0002-9242-5588

The paper investigates the value of simulations for research activities and identifies the main reasons for the feasibility of conducting experiments in the virtual space.

The work includes two-dimensional and three-dimensional simulations built using Unity Engine. The technology of simulation in two-dimensional space was used for the simulation of Gray – Scott's reaction-diffusion model. The considered model involves the exploration of the system in which the diffusion reaction of two substances takes place. The resulting software based on this model allows simulating the diffusion pattern in real time or speed up the flow of time in the simulation. The software product allows configuring the basic parameters of the reaction, which gives an opportunity to build simulations of any given substances or systems. It displays the result of the simulation in several modes, which allows user to evaluate different aspects of the studied reaction at any time, e.g. view the concentration of substances, the value of the change in concentration per time unit at each point in the reaction plane.

For the implementation of a simulation of Gray – Scott's reaction-diffusion model, the possibility to apply the optimization method to it by transferring calculations to the graphics core was investigated. Research has shown numerous advantages of the parallelization of calculations by the means of performing them on many threads of the graphics adapter. During the parallelization process for each pixel of the input image, which shows the initial substances application pattern on the plane, a separate stream is allocated, which calculates the values of concentration and changes in concentration at the material point of the reaction plane. From the stated above it can be concluded that the number of running computing streams is equal to the number of pixels in the image.

Therefore the simulated visualization of diffusion helps to better understand real world processes, such as chemical reactions in the synthesis of hematopoiesis, fermentation.

The perspectives of simulation in three-dimensional space were analyzed at the basis of the behavior of living flocks of the same type. A host of simple processes, such as avoiding collisions of elements with each other inside their groups, maintaining a common direction of movement, and bypassing obstacles along the way were implemented. The resulting flock behavior is not governed by one module of behavior of the whole flock, but is formed by the behavior of each individual entity within the flock. Therefore, the behavior of the simulated flock reflects the real behavior of flocks in nature.

The described simulation was also investigated for the possibility of optimizing processes using calculations on the graphics core. Each element of the flock has a separate independent module that requires the awareness of all other entities in the flock to determine the impact on its own decisions about the movement and orientation Therefore, the graphical core allocates a separate stream for each flock entity.

Key words: simulation; Unity; C#; HLSL; shader; compute shader; Gray – Scott reaction-diffusion model; flocks; bird-like objects.