https://doi.org/10.31891/2219-9365-2024-79-8
УДК 004.78

БЕРНАТОВИЧ Анатолій
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
https://orcid.org/0000-0002-8403-6363
e-mail: iracotnar1972@gmail.com

АНАЛІЗ ВИМОГ ДО РОЗРОБКИ УНІВЕРСАЛЬНОГО ФІЗИЧНОГО РУШІЯ ІЗ ПІДТРИМКОЮ ПЛАГІНІВ, КОНФІГУРАЦІЄЮ ПАЙПЛАЙНУ ТА АПАРАТНИМ ПРИСКОРЕННЯМ

Ця стаття присвячена створенню вимог до гнучкого та розширюваного фізичного рушія, здатного підтримувати модулі (плагіни), конвеєр обробки, що конфігурується, та інтуїтивно зрозумілий редактор для створення моделей, використовуючи при цьому апаратне прискорення для підвищення продуктивності. Представлено архітектуру фізичного рушія з чітким розмежуванням між основними компонентами та модулями-плагінами. Запропоновано структуру проєкту, що включає чіткий розподіл завдань, з основними компонентами, модулями та графічним інтерфейсом, що зберігаються в різних директоріях. Обговорюється важливість інструментів тестування та налагодження, таких як CMake, для створення та тестування рушія на різних платформах.

Ключові слова: фізична симуляція, фізичний рушій, модульність, симуляція, C++.

BERNATOVYCH Anatolii
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

ANALYSIS OF REQUIREMENTS FOR THE DEVELOPMENT OF A UNIVERSAL PHYSICS ENGINE WITH PLUG-IN SUPPORT, PIPELINE CONFIGURATION, AND HARDWARE ACCELERATION

The aim of the article is the explanation of the physics engine that allows for extensions through modules (plug-ins), has a pipeline of processing that can be modified, has an editor for defining the shapes of the models, and can use the hardware acceleration for computing. This will allow the researchers to apply it in different contexts like computer graphics, animation, scientific modelling, or engineering. According to the article, the extensibility of a physics engine means that the developers can include one or more new physical states or alter an existing one without altering the engine.

It is proved that the modular approach has several advantages: 1. It is easy to add new physical laws and phenomena to the engine without changing the structure of the engine. Special modules for various research or industrial requirements can be built, thus tailoring the engine to the respective domain. Pre-existing modules can therefore be reused from one project and utilized in other projects thereby enhancing the integration process and shortening the time for development. This improves scalability since the engine can expand by adding or replacing existing modules to accommodate new technologies or new requirements. The authors also explain visualization and interactive editing as factors that are essential in enhancing the effectiveness of physics engines. They also suggest a project structure with a core physics engine, modules, editor, user interface, test system, and compatibility with different physical models. Designing a configurable physics engine that can be effectively used for various games and situations is a task that can only be accomplished by a dedicated programmer with profound knowledge of the physics mechanisms. Including distributed computing features, annotated editing, hardware enhancing features, and utilizing techniques like MPI, OpenGL, Vulkan, SFML to improve this flexibility. Therefore, it is possible to list the following beneficial aspects of a physics engine that must be productive, flexible, unalike, and competitive in different researching and entertaining usages.

As for the further developments of the topic for the research, the improvements towards the integration with other systems, development of the new physical models and the usage of the various resource optimization methods like parallel computations for the better performance of the engine have to be mentioned. This makes new opportunities in the sphere of physical modelling, which is critical for science, education, entertainment, and development.

Keywords: physical simulation, physical engine, modularity, simulation, C++.

ПОСТАНОВКА ПРОБЛЕМИ У ЗАГАЛЬНОМУ ВИГЛЯДІ ТА ЇЇ ЗВ'ЯЗОК ІЗ ВАЖЛИВИМИ НАУКОВИМИ ЧИ ПРАКТИЧНИМИ ЗАВДАННЯМИ

У контексті фізичного моделювання основною потребою є можливість зібрати модель і провести ряд симуляцій для дослідників або розробників. Будучи частиною додатків, фізичні рушії забезпечують галузі від комп'ютерної графіки до наукового моделювання та ігор. Багато зусиль докладається для того, щоб зробити моделі реалістичними, коли мова йде, зокрема, про фізичні процеси, і тому акцент робиться на графічних моделях. Однак фізика, яка включає в себе різні форми фізичного моделювання, повинна бути продуктивною у своїй роботі, гнучкою і розширюваною за необхідності, що є дещо складнішим завданням. На практиці, створення фізичного рушія є нетривіальною і, насправді, досить цікавою роботою для програмістів з точки зору програмної інженерії. Наприклад, сучасний світ не може обійтися без фізичного рушія для численних додатків в індустрії розваг, анімації та моделювання, симуляції в науці та техніці, а також численних обчислювальних алгоритмів в інженерії. Оскільки інструмент буде використовуватися для

роботи з різними типами вимог і специфікацій, вкрай важливо розробити структуру, що дозволяє вносити зміни «на льоту». Такий функціонал, як розширюваність або здатність використовувати апаратне забезпечення для досягнення вищої швидкості моделювання, може кардинально змінити уявлення про те, на що мають бути здатні фізичні рушії в будь-якій галузі, якщо вони є актуальними для всієї галузі.

АНАЛІЗ ДОСЛІДЖЕНЬ ТА ПУБЛІКАЦІЙ

В роботі [1] наведено дані про важливість розробки власного фізичного рушія з огляду на недоліки наявних. Авторами вперше поставлено це питання тому інших досліджень з цього питання немає.

ФОРМУЛЮВАННЯ ЦІЛЕЙ СТАТТІ

Метою роботи є: аналіз та моделювання вимог для створення фізичного рушія, який підтримує розширення через модулі (плагіни), має налаштований пайплійн обробки, інтуїтивний редактор для створення моделей і підтримує апаратне прискорення обчислень.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Часто надається великого значення розширюваність сучасного фізичного рушія. Рушій повинен мати можливість включати «модулі» (плагіни), які вводять більш реальні фізичні стани або змінюють існуючі. Це дає можливість розробникам додавати нові функції та компоненти, наприклад, модель демпфера пружинної маси, що використовується в багатьох симуляціях деформівних тіл, без необхідності модифікувати саме ядро рушія. Особи, які беруть участь у процесі розробки програмного забезпечення, можуть використовувати такі мови, як C++ [2, 3], які надають більше контролю та універсальності у написанні коду. API рушія має включати елементи для створення, реєстрації та використання плагінів. Кожен плагін реалізується як збірка, яка може включати набір компонентів, що розширюють можливості плагіна. Наприклад, розробник, який працює над моделлю автомобіля, може створити плагін, який забезпечує фізичні характеристики, такі як тертя, демпфування та інші, необхідні при моделюванні автомобіля. До речі, інший розробник, який моделює літак, може розробити інший плагін, який враховує аеродинаміку, наприклад. Це було видно в архітектурі фізичного рушія, де Pipeline є ядром. Через нього програма регулює кроки, які виконуються в процесі симуляції. Конфігурація конвеєра обробки фактично пропонує варіант, в якому будуть використані реальні фізичні моделі. Це дуже корисно, оскільки деякі моделі можуть бути працездатними, незважаючи на виключення деяких фізичних ефектів. Наприклад, візьмемо створення програми, яка навчатиме пілота навичкам польоту; можна вирішити «вимкнути» тертя для програми.

Як і у випадку з моделюванням, визначено місце візуалізації в повному комплексі фізичної симуляції. Редактор моделі та конвеєр процесів будуть інтерактивними, тобто дослідники зможуть виконувати роботу швидше, не звертаючи уваги на те, як працює програмування. Там, де це можливо, редактор повинен бути спроектований таким чином, щоб створити новий екземпляр фізичного об'єкта, задати його властивості та зрозуміти результати цієї дії.

Конвеєр фізичної обробки дозволить нам досліджувати різні фізичні моделі, виключаючи або включаючи окремі фізичні закони. Ось чому ця функція зручна в сценаріях, де характеристики, висвітлені вище, не є життєво важливими для оцінки змодельованого результату. Це означає, що вона допомагає дослідникам зосередитися на реальній фізиці своїх моделей, а не на аспекті симуляції, що не є критичним і не займає багато часу.

Конвеєр відносно чітко сформульований у своїх фізичних моделях і факторах, які можуть бути налаштовані відповідно до потреб користувача для декількох версій одночасно, що дає платформу для альтернативних розробок. Наприклад, якщо вчений-експериментатор, який досліджує рух небесного тіла, хоче виключити фактор тертя, то так само буде виключена статистика контакту у вакуумі або порожнечі. Фізичний рушій тепер фактично є набором елементів, і одним з таких початкових елементів є інтегрований редактор моделей і конвеєрів, який є основою для побудови конфігурації симуляції.

Редактор повинен дозволяти користувачеві через веб-інтерфейс описувати тіла, їхні характеристики та взаємодії в фізичній симуляції. Користувач повинен вибрати фізичні властивості кожного об'єкта, такі як маса, пружність, тертя тощо. За допомогою інтерактивної демонстрації слід показати користувачам, як обрані ними параметри впливають на результат фізичної симуляції. Після запуску симуляції користувач отримає детальний звіт про результати, що дозволить оцінити правильність обраних параметрів. Редактор повинен відображати всі параметри, що беруть участь у симуляції, такі як сила, швидкість, напруження тощо, а також діаграми, такі як діаграми сили, вектор швидкості, тенденції деформації тощо. Редактор повинен бути більш-менш технічно орієнтованим - основною цільовою групою є науковці та інженери, які не обов'язково є гуру комп'ютерної графіки або програмування високого рівня.

Отже, для того, щоб досягти найбільшої фактичної продуктивності обчислень, необхідно інтегрувати в систему апаратне забезпечення, яке має кращу швидкість. Серед них - «командні» операції центрального процесора, який скорочено називається CPU, та графічного процесора, або скорочено GPU.

Графічний процесор є кращим для розпаралелення обчислень, пов'язаних з фізичним рушієм, наприклад, при пошуку колізій або при відображенні фізичних атрибутів для візуалізації і т.д. Це може бути записано як прискорення апаратного забезпечення для використання OpenGL, Vulkan, DirectX означає, що для підвищення швидкості роботи програм, які використовують ці графічні API, залучаються додаткові обчислювальні ресурси відеокарти. Тобто, частина навантаження з центрального процесора переноситься на відеокарту, яка спеціалізується на обробці графіки. Це дозволяє програмам працювати швидше і ефективніше. Це дозволить нам робити інтенсивні обчислення на графічних процесорах, оскільки, як було показано раніше, графічні процесори працюють краще, ніж центральні процесори в таких обчисленнях.

Сучасні фізичні рушії повинні працювати на різних платформах: починаючи зі стаціонарних терміналів, таких як ПК, ноутбуки, смарт-телевізори, і переходячи до порівняно недавно класифікованих стаціонарних терміналів, включаючи гарнітури віртуальної/доповненої реальності, ігрові консолі, портативні термінали, такі як смартфони та планшети. Існує кілька підходів до вирішення цього завдання: використання спеціалізованих графічних API (OpenGL), універсальних мультимедійних бібліотек (SFML, SDL) або ж більш абстрактних, незалежних від платформи інструментів. Обробка великих і деталізованих моделей вимагає значних обчислювальних ресурсів і може потребувати спеціальних оптимізацій, подібних до тих, що використовуються при розробці дистрибутивів. Іншим інструментом є інтерфейс передачі повідомлень, часто згадуваний як MPI; він корисний при створенні програм, призначених для паралельних обчислень на різних комп'ютерах. Він корисний тим, що розбиває роботу на частини, які можна розмістити на різних вузлах, щоб підвищити ефективність роботи, і в той же час оптимізувати використання ресурсів.

Зупинимось на виборі інструментів та середовища розробки:

1. Редактор коду: Visual Studio Code - відмінний вибір завдяки широкій підтримці C++ [4], дебагуванню та великій кількості розширень. Компілятор: Microsoft Visual C++ - інтегрований у Visual Studio Code і забезпечує високу продуктивність. Система побудови: CMake - гнучка система, яка дозволяє легко керувати проектом і генерувати файли проекту для різних платформ. Система контролю версій: Git - для ефективного керування кодом і співпраці. Розподілені обчислення: стандарт MPI та бібліотеки на кшталт MPICH або OpenMPI для розподілених обчислень.

2. Структура проєкту:

Спочатку ідея майбутнього проєкту була така:

```
physics_engine/
├── include/
│   ├── physics_engine.h
│   ├── module.h
│   └── deformable_body_module.h
├── src/
│   ├── window.cpp
│   ├── physics_engine.cpp
│   ├── module.cpp
│   └── deformable_body_module.cpp
└── CMakeLists.txt
```

У Physics_engine.h має міститися оголошення класу PhysicsEngine, який буде керувати всією симуляцією, оголошення методів для ініціалізації, оновлення та рендерингу, оголошення приватних членів для зберігання інформації про модулі та інші параметри симуляції.

physics_engine.cpp відповідатиме за реалізацію методів класу PhysicsEngine, створення екземплярів модулів та їх додавання до вектора модулів. Логіка оновлення симуляції (викликає метод update для кожного модуля). Може містити функції для завантаження конфігурацій, збереження стану тощо.

module.h міститиме оголошення абстрактного базового класу Module. Віртуальні методи, які повинні бути реалізовані в конкретних модулях (наприклад, update, render).

module.cpp матиме деякі загальні реалізації методів для всіх модулів або допоміжні функції.

deformable_body_module.h мітитиме оголошення класу DeformableBodyModule, який успадковується від Module, оголошення методів, специфічних для деформації тіл (наприклад, додавання сили, оновлення сітки).

deformable_body_module.cpp сприятиме реалізації методів класу DeformableBodyModule. Використання математичних моделей для симуляції деформації (наприклад, метод скінченних елементів).

CMakeLists.txt - файл для конфігурування проекту за допомогою CMake [5]. Визначення залежностей, компілятора, налаштування лінкера та інших параметрів.

Наступні причини пояснюють, чому було розроблено цю модель: Кожен файл належить до того сегменту системи, для якого він призначений. Базовий клас Module є основою структури модулів та

різноманітних інтерфейсів між ними. Основною концепцією тут є те, що приєднання нових модулів може бути зроблено дуже простим способом, шляхом зміни вихідних даних симуляції. CMake використовується для побудови проекту, і це робиться особливим чином, а саме в автоматичному режимі.

З цієї причини його структура не надто складна і є більш-менш базовою, а тому її легко створювати, а потім адмініструвати. Є ще одна перевага для невеликих систем, де обсяг коду досить малий: така структура проекту вигідна для переміщення системи та навігації.

Однак така структура не гарантує чіткого розподілу завдань щодо ядра рушія, модулів, графічного інтерфейсу користувача і так далі. Нелегко розширювати цю структуру далі, оскільки вона посилає неправильні сигнали групам виконавців. Наприклад, всі вихідні файли за домовленістю розміщуються в одному каталозі під назвою «src», що може стати незручним у випадку з великими об'єктами. Чим більше модулів, функцій або навіть елементів графічного інтерфейсу використовується, тим більше зростає складність системи, що призводить до ускладнення структури управління.

Тому структура була розширена та деталізована для поліпшення організації, модульності та масштабованості фізичного рушія з огляду на обмеження попередньої версії:

```
/PhysicsEngine
  /src
    /core
      PhysicsEngine.cpp
      RigidBody.cpp
      SoftBody.cpp
      Force.cpp
      Collision.cpp
    /modules
      SpringMassPlugin.cpp
    /editor
      EditorGUI.cpp
      PipelineEditor.cpp
      Visualization.cpp
  /include
    PhysicsEngine.h
    RigidBody.h
    SoftBody.h
    Force.h
    Collision.h
    Module.h
  /plugins
    SpringMassPlugin.dll
  /tests
    TestPhysicsEngine.cpp
  /CMakeLists.txt
```

/PhysicsEngine - коренева директорія проєкту, де зберігаються всі основні компоненти фізичного рушія, включаючи вихідні файли, заголовкові файли, модулі, плагіни, тести та конфігураційні файли.

/src - директорія містить всі вихідні коди рушія, розділені на підкатегорії для зручності.

/core містить основні компоненти фізичного рушія.

1. PhysicsEngine.cpp:

- Реалізація основного класу рушія (PhysicsEngine), який керує всіма фізичними об’єктами та обчисленнями. Включає методи для запуску симуляцій, оновлення фізики кожного кадру, керування об'єктами та взаємодії між ними.

2. RigidBody.cpp:

- Реалізація класу RigidBody, який представляє жорстке тіло в симуляції. Включає параметри, такі як маса, швидкість, імпульс, та методи для обчислення взаємодії між твердими тілами.

3. SoftBody.cpp:

- Реалізація класу SoftBody, який представляє деформівне тіло. Включає параметри, такі як еластичність, в’язкість, а також методи для обчислення внутрішніх і зовнішніх сил, які впливають на деформівне тіло.

4. Force.cpp:

- Реалізація класу Force, який представляє різні типи сил (наприклад, гравітація, пружні сили). Включає методи для додавання, видалення та обчислення дії сил на фізичні об’єкти.

5. Collision.cpp:

- Реалізація класу Collision, який керує обчисленням зіткнень між об'єктами. Включає методи для виявлення зіткнень та обробки реакцій на ці зіткнення (наприклад, відскок або злипання).

/modules містить реалізації модулів (плагінів), які можуть розширювати функціональність рушія.

1. SpringMassPlugin.cpp:

- Реалізація модуля для роботи з пружинно-масовою моделлю. Включає клас, що реалізує поведінку пружин і мас, взаємодію між ними та реєструє нові компоненти у фізичному рушії.

/editor містить код для редактора моделей і пайплайну обробки.

1. EditorGUI.cpp:

- Реалізація основного інтерфейсу користувача (GUI) редактора. Включає елементи для створення та налаштування моделей, запуску симуляцій, а також для керування пайплайном обробки.

2. PipelineEditor.cpp:

- Реалізація редактора пайплайну обробки. Включає інструменти для додавання, видалення і конфігурації різних етапів обробки фізичної симуляції (наприклад, обчислення сил, інтеграція руху, зіткнення).

3. Visualization.cpp:

- Реалізація модуля для візуалізації симуляційних результатів. Включає методи для рендерингу моделей, відображення результатів симуляції в реальному часі.

/include містить заголовкові файли для всіх основних класів і компонентів.

1. PhysicsEngine.h:

- Оголошення класу PhysicsEngine та його основних методів. Включає оголошення всіх необхідних змінних і функцій, пов'язаних з управлінням фізичними об'єктами та симуляціями.

2. RigidBody.h:

- Оголошення класу RigidBody, включаючи його властивості та методи. Визначає інтерфейс для роботи з жорсткими тілами.

3. SoftBody.h:

- Оголошення класу SoftBody. Визначає інтерфейс для роботи з деформівними тілами, включаючи методи для розрахунку деформацій.

4. Force.h:

- Оголошення класу Force, включаючи його типи (гравітація, пружність) та методи для їх обчислення.

5. Collision.h:

- Оголошення класу Collision, який містить методи для обробки зіткнень між фізичними об'єктами.

6. Module.h:

- Оголошення базового класу Module, який використовується для створення плагінів. Визначає інтерфейс для розширення функціональності фізичного рушія за допомогою модулів.

/plugins - директорія для збереження зібраних модулів (плагінів).

1. SpringMassPlugin.dll:

- Зібраний плагін, який можна динамічно підключати до рушія для розширення його функціональності.

/tests містить тести для перевірки роботи рушія.

1. TestPhysicsEngine.cpp:

- Реалізація тестів для основних компонентів фізичного рушія. Включає юніт-тести для перевірки правильності роботи ядра рушія, фізичних моделей, модулів та взаємодії між ними.

CMakeLists.txt - конфігураційний файл для системи збірки CMake. CMakeLists.txt містить інструкції для збірки проєкту, включаючи підключення залежностей, компіляцію вихідних файлів, збірку плагінів та створення виконуваних файлів. Включає команди для створення бібліотек, підключення зовнішніх бібліотек, конфігурації тестів і налаштувань для кросплатформенності.

Ця структура забезпечує високу модульність, чітку організацію коду і полегшує розширення проєкту.

Кожен основний компонент рушія (ядро, модулі, редактор) ізольований в окремій директорії, що сприяє кращій організації коду та полегшує його розширення. Логічно пов’язані файли розташовані разом. Наприклад, всі базові класи для фізичних об'єктів знаходяться в папці core, а файли GUI — в editor. Така структура легше масштабується, коли потрібно додавати нові модулі, компоненти або функції. Це також полегшує підтримку та тестування окремих частин проєкту. Наявність окремої папки tests дозволяє чітко організувати тести для різних частин коду, що важливо для підтримки якості проєкту.

Зважаючи на оновлену структуру проєкту, пропонуємо вимоги до реалізації основних модулів і функціональності:

1. physics_engine.h/cpp

Основні компоненти:

- Клас PhysicsEngine для керування симуляцією:

- Основний клас рушія, що відповідає за керування симуляцією, зберігання та управління всіма фізичними об'єктами, а також інтеграцію з різними модулями.

- Функції для додавання та видалення об'єктів:

- Метод AddObject(PhysicalObject* obj) додає фізичний об'єкт до симуляції.

- Метод RemoveObject(PhysicalObject* obj) видаляє фізичний об'єкт із симуляції.

- Функція для виконання кроку симуляції:

- Метод StepSimulation(float deltaTime) виконує один крок симуляції, оновлюючи стан всіх фізичних об'єктів згідно з переданим часом (deltaTime).

- Інтерфейс для завантаження плагінів:

- Метод LoadPlugin(const std::string& pluginPath) завантажує плагін (модуль) із вказаного шляху.

- Метод UnloadPlugin(const std::string& pluginName) вивантажує плагін.

2. module.h/cpp

Основні компоненти:

- Абстрактний базовий клас для модулів:

- Клас Module є абстрактним базовим класом, від якого успадковуються всі модулі. Визначає інтерфейс, який мають реалізувати всі модулі.

- Віртуальні функції для ініціалізації, оновлення та рендерингу:

- virtual void Init(PhysicsEngine* engine) = 0; ініціалізація модуля з доступом до фізичного рушія.

- virtual void Update(float deltaTime) = 0; оновлення стану модуля на кожному кроці симуляції.

- virtual void Render() = 0; Рендеринг візуалізації модуля (якщо це необхідно).

3. deformable_body_module.h/cpp

Основні компоненти:

- Клас, що успадковується від Module:

- Клас DeformableBodyModule успадковується від Module і реалізує поведінку деформованих тіл.

- Реалізує поведінку деформованих тіл (наприклад, за допомогою фізики тканин):

- Включає параметри та методи, які дозволяють симулювати поведінку деформованих тіл, таких як тканини або еластичні матеріали, наприклад, на основі пружинно-масової моделі.

- void Init(PhysicsEngine* engine) override; Ініціалізує деформоване тіло в контексті фізичного рушія.

- void Update(float deltaTime) override; Оновлює стан деформованого тіла на кожному кроці симуляції.

- void Render() override; Відповідає за рендеринг деформованого тіла (якщо потрібно).

4. Реалізація системи плагінів:

- Механізм завантаження:

- Варто використати динамічне завантаження бібліотек (LoadLibrary на Windows) для завантаження плагінів під час виконання програми.

- Інтерфейс плагіна:

- Визначте чіткий інтерфейс для плагінів через базовий клас Module, що дозволить легко інтегрувати нові модулі в рушій.

5. Конфігурація пайплайну:

- Файл конфігурації:

- Використовуйте формат JSON [6] або YAML [7] для збереження налаштувань пайплайну обробки (наприклад, черговість етапів обчислення).

- Завантаження конфігурації:

- Реалізуйте завантаження конфігурації на початку роботи програми, щоб автоматично налаштовувати пайплайн відповідно до завантаженого файлу.

6. Апаратне прискорення:

- Використання GPU:

- Розгляньте використання CUDA або OpenCL для реалізації обчислень на графічному процесорі для прискорення симуляцій.

- Використання SIMD:

- Оптимізуйте обчислення на CPU за допомогою SIMD-інструкцій (наприклад, через використання бібліотеки Eigen або схожих).

7. Інтерфейс користувача:

- Вибір бібліотеки GUI:

- Використовуйте Qt [8], wxWidgets [9] або інші бібліотеки для створення графічного інтерфейсу.

- Функціональність:

- Реалізуйте функції для завантаження сцен, налаштування параметрів симуляції та відображення результатів у реальному часі.

8. Тестування та дебагування:
- Написання тестів:
- Створіть одиничні тести для перевірки основних функцій рушія, включаючи модулі, завантаження плагінів та оновлення фізики.
- Використання дебагера:
- Використовуйте вбудовані інструменти для налагодження (наприклад, Visual Studio Code) для відстеження та виправлення помилок у програмі.
9. Розширення функціональності:
- Додавання нових типів об'єктів:
- Реалізуйте нові класи, які успадковуються від Module, щоб додавати нові фізичні моделі (наприклад, рідини або гази).
- Підтримка різних фізичних моделей:
- Додайте підтримку різних фізичних моделей, які можуть бути інтегровані до рушія за допомогою нових модулів або плагінів.
- Інтеграція з іншими системами:
- Додайте можливість інтеграції з іншими програмами, наприклад, через API або інші інтерфейси, щоб використовувати фізичний рушій у різних проєктах, включаючи ігри або наукові симуляції.

ВИСНОВКИ З ДАНОГО ДОСЛІДЖЕННЯ І ПЕРСПЕКТИВИ ПОДАЛЬШИХ РОЗВІДОК У ДАНОМУ НАПРЯМІ

Коли справа доходить до симуляцій, ідея мати спільний, гнучкий і масштабований рушій обчислювальної фізики разом з можливістю використовувати апаратне прискорення має велике значення. Конвеєри допомагають користувачам працювати над дослідженнями замість того, щоб витрачати час на складнощі моделювання, створення та візуалізацію симуляцій. Кросплатформеність рушія та можливості апаратного прискорення означають, що програма може бути більш корисною в багатьох дослідницьких контекстах. По мірі розвитку рушія, як описано тут, до нього будуть додаватися розподілені обчислення, що дозволить працювати з більшими симуляціями. Розробка фізичного рушія, який буде застосовний до всіх типів ігор і сценаріїв, - це завдання, яке повинен вирішувати програміст, що володіє достатніми знаннями про фізичну симуляцію. Можливість його подальшого розширення та зміни фреймворку, створення графічного інтерактивного редактора, перевага використання апаратного прискорення та розподілу обчислювального навантаження роблять його цінним у багатьох галузях. Таким чином, використовуючи доступні інструменти, такі як MPI, OpenGL, Vulkan або SFML, буде спроектовано та реалізовано впорядкований фізичний рушій для різних платформ, який, перш за все, буде пропонувати розумну продуктивність, але в той же час буде унікальним та конкурентоспроможним.

Література

1. Бернатович А., Стеценко І. Методи та програмні засоби фізичної симуляції / А. Бернатович, І. Стеценко // Адаптивні системи автоматичного управління. – 2023. – Т. 1. – № 42. – С. 130-140. https://doi.org/10.20535/1560-8956.42.2023.279104
2. Офіційна документація C++ : [вебсайт]. – Режим доступу : https://en.cppreference.com/w/
3. Зеленський О.С., Лисенко В.С. Основи програмування на C++ / О.С. Зеленський, В.С. Лисенко // Кривий Ріг: Державний університет економіки і технологій. – 2023. – 269 с. – Режим доступу : https://dspace.duet.edu.ua/jspui/bitstream/123456789/831/1/%D0%9D%D0%9F%20Osnovy_C%2B%2B.pdf
4. Visual Studio Code : [veb-sait]. – Режим доступу : https://code.visualstudio.com/
5. CMake : [veb-sait]. – Режим доступу : https://cmake.org/cmake/help/latest/manual/cmake.1.html
6. JSON : [veb-sait]. – Режим доступу : https://www.json.org/json-en.html
7. YAML : [veb-sait]. – Режим доступу : https://yaml.org/
8. Qt: офіційна документація : [вебсайт]. – Режим доступу : https://doc.qt.io/
9. wxWidgets : [veb-sait]. – Режим доступу : https://docs.wxwidgets.org/3.2.5/

References

1. Bernatovych A., Stetsenko I. Metody ta prohramni zasoby fizychnoi symuliatsii / A. Bernatovych, I. Stetsenko // Adaptyvni systemy avtomatychnoho upravlinnia. – 2023. – T. 1. – № 42. – S. 130-140. DOI: https://doi.org/10.20535/1560-8956.42.2023.279104
2. Ofitsiina dokumentatsiia C++ : [veb-sait]. Rezhym dostupu : https://en.cppreference.com/w/
3. Zelenskyi O.S., Lysenko V.S. Osnovy prohramuvannia na C++. Kryvyi Rih: Derzhavnyi universytet ekonomiky i tekhnolohii, 2023. 269 s. Rezhym dostupu : https://dspace.duet.edu.ua/jspui/bitstream/123456789/831/1/%D0%9D%D0%9F%20Osnovy_C%2B%2B.pdf
4. Visual Studio Code : [veb-sait]. – Rezhym dostupu : https://code.visualstudio.com/
5. CMake : [veb-sait]. – Rezhym dostupu : https://cmake.org/cmake/help/latest/manual/cmake.1.html
6. JSON: [veb-sait]. – Rezhym dostupu : https://www.json.org/json-en.html
7. YAML: [veb-sait]. – Rezhym dostupu : https://yaml.org/
8. Qt: ofitsiina dokumentatsiia: [veb-sait]. – Rezhym dostupu : https://doc.qt.io/
9. wxWidgets: [veb-sait]. – Rezhym dostupu : https://docs.wxwidgets.org/3.2.5