КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

3D ГРАФІКА ТА ПРОЦЕДУРНА АНІМАЦІЯ В ПРИКЛАДНОМУ ЗАСТОСУВАННІ

Виконав студент 4-го курсу

Андрій БОНДАРЕНКО _____ (підпис)

Науковий керівник:

доцент, кандидат фіз.-мат. наук

Лариса КАТЕРИНИЧ _____ (підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____ (підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«29» травня 2023 р.,
Протокол № 11
Завідувач кафедри

Олександр ПРОВОТАР _____ (підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 53 сторінки, 14 рисунків, 25 джерел посилань

3D ГРАФІКА, АНІМАЦІЯ, ПРОЦЕДУРНА АНІМАЦІЯ, ПРОЦЕДУРНА АНІМАЦІЯ РУХУ, ДИФЕРЕНЦІАЛЬНІ РІВНЯННЯ ДРУГОГО ПОРЯДКУ, ПРОЕКТУВАННЯ ІНТЕРФЕЙСУ, ІНТЕГРАЦІЯ В UNITY.

Об'єктом роботи є дослідження 3D графіки та анімації в комп'ютерних іграх. Предметом роботи є створення на основі матеріалу з проведеного дослідження процедурної анімації для руху істот.

Метою роботи є дослідження 3D графіки та анімації та створення процедурної анімації за допомогою математичного підгрунтя та технологій Unity.

Методи розробки: методи розв'язання диференціальних рівнянь, створення алгоритмів для процедурної анімації, дослідження результатів та їх налаштування.

Інструменти розроблення: безкоштовний ігровий рушій для створення 2D та 3D ігор Unity3D, мова програмування C#.

Результат роботи: проведене дослідження основних аспектів 3D графіки та анімації. Проаналізовано диференціальні рівняння другого порядку та як вони описують рух тіла. Також окремо взято до уваги кожен з параметрів уже згаданого рівняння та детальніше описано як кожен з них впливає на окремі аспекти руху тіла. Було проаналізовано існуючі технології та рішення щодо процедурної анімації, які наявні в ігровому рушії Unity. За допомогою описаних вище моментів, була розроблена архітектура системи процедурної анімації та інтерфейс для її контролю. У результаті було створено комп'ютерну гру, що включає в себе всі обговорені вище аспекти.

Новизна роботи: проведені дослідження та розробка описують процедурну анімацію не лише з точки зору створення алгоритмів для анімації руху кінцівок істот, а і беруть до уваги швидкість та прискорення об'єкта гри.

За допомогою цього здобувається більша реалістичність руху та покращуються відчуття від гри.

Рекомендації щодо використання результатів роботи: створене дослідження процедурної анімації руху в комп'ютерних іграх допоможе бажаючим краще розібратись в необхідному математичному апараті, потрібному для її реалізації. Розроблена гра можна застосовувати як приклад використання теоретичних знань на практиці, що є одним з найважливіших моментів у навчальному процесі.

Сфера застосування. Як вже було описано вище, дане дослідження можна використовувати в навчальному процесі або при створенні процедурної анімації в комп'ютерних іграх.

Значимість роботи. На сьогоднішній день комп'ютерні ігри займають значне місце у життях людей. Оскільки процедурна анімація контролюється за допомогою алгоритмів та математичних функцій, то це дозволяє їй бути більш органічною та адаптивною, що привнесе реалістичності в гру, що у свою чергу принесе більше задоволення людині, яка в неї грає. Також, використання математичних знань у створенні анімації, тим паче анімації руху, може зберегти багато часу при розробці, відчутно прискоривши процес створення повторюваних та динамічних анімацій.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта дослідження. 3D графіка та анімація увійшли в наше життя та міцно засіли в ньому. За допомогою віртуальної реальності можна робити будь-що: відвідати фантастичні місця своєї улюбленої гри, поглянути, що відбувається в космосі чи на дні океану, або навіть просто переглянути будь-який куток нашої планети.

Узагалі, якщо обдумати роль 3D графіки та анімації в сучасному житті, то отриманий список із позитивних речей, які вони привносять, допомагає не лише розвиватись в індивідуальному плані, але ще й на рівні цілого суспільства. Розглянемо деякі приклади.

Навчання. Якщо вчитель для того, щоб пояснити нову тему, буде малювати щось на дошці, то навряд чи багато дітей буде із захопленням спостерігати за цим дійством. Інша річ, якщо вчитель, для опрацювання нового матеріалу, покаже деяку 3D анімацію. Це тому, що анімації, зазвичай, носять більш розважливий характер, і учні запам’ятовують нові теми краще. Розглянемо, для прикладу, процес навчання студентів медичного інституту. Читання книжок, безумовно, дає потрібні знання, але ж де взяти потрібний практичний досвід? Переглянувши анімацію, наприклад, з проведення певної процедури чи просто оглянувши тривимірну модель якогось органу студенти зможуть відчути цей досвід. Те саме стосується й старих, нерухомих моделей, що використовувались для навчання в минулому. У сучасному світі за допомогою анімацій студенти можуть набагато ефективніше вивчати людські процеси, такі як, наприклад, справжні, реальні рухи тіла людини.

Наука. 3D графіка та анімація також мають широке застосування в науці. Візуалізація даних з використанням тривимірної графіки дозволяє відображати складні набори даних у вигляді моделей. У свою чергу, це допомогає опрацьовувати дослідникам великі обсяги інформації. Візьмемо згадану вище сферу медицини, 3D моделі органів не лише допомагають студентам краще

вчитись, а й дозволяють науковцям проводити аналіз та вивчення хвороб та різних хірургічних процедур. Моделювання також знайшло своє місце і в галузях фізики, де воно використовується для вивчення поведінки складних систем. У астрономії анімація моделей допомогає вивчати процес руху планет, що було б неможливо зробити в реальному часі.

Бізнес. 3D графіка та анімація також допомагають у сфері бізнеса. Найпопулярнішим їхнім застосуванням у цій області буде реклама та маркетинг. Саме анімація дозволяє створювати рекламні ролики чи презентації, що демонструють функції та переваги рекламованих продуктів. Якщо говорити про звіти, то велику кількість цінної інформації, які вони несуть для певного підприємства, також можна відобразити за допомогою графіки. Така візуалізація допоможе краще відслідкувати динаміку продажів та ключові моменти для бізнесу.

Комп'ютерні ігри. Ну і звичайно, один з найпопулярніших сегментів, в якому знайшли своє застосування 3D графіка та анімація – комп'ютерні ігри. Світ, який досліджуєш; герой, яким керуєш; персонажі, в яких береш завдання; зброя, якою завдаєш поразки ворогам – все це графіка. Але ігровий світ не є статичним. Біг, стрільба, змахи мечем - вдихнути у персонажів життя неможливо було б без анімації. Звичайно не варто забувати про візуальні ефекти, такі як дим, вогонь, освітлення, тіні. Всі ці ефекти надають реалістичності грі і не було б можливими без використання графіки та анімації.

Актуальність роботи та підстави для її виконання. Як вже було описано вище, 3D графіка та анімація знайшли своє місце у багатьох важливих сферах сьогодення, особливо у відеоіграх. У розробці ігор, особливо ігор великих масштабів, одним з найважливіших пунктів є час, який буде витрачено на її виконання. У ігровому світі є багато речей, які треба змусити рухатись, тобто, анімувати. Одним з найпопулярніших видів анімації є анімація з ключовими кадрами, між якими використовується інтерполяція для створення плавного переходу. Проте, на анімування повторюваних чи динамічних дій може витрачатись багато часу. Щоб його зберегти, можна

використати процедурну анімацію, тобто анімацію за допомогою математичних функції та алгоритмів, на якій і буде зроблений акцент у моєму дослідженні.

Мета й завдання роботи. Метою роботи є дослідження 3D графіки та анімація в комп'ютерних іграх з акцентом на аналізі та визначенні особливостей процедурної анімації руху. Завданням буде створення гри, систематизуючи досвід з дослідження вище описаних аспектів. Для досягнення цієї мети було поставлено такі задачі:

- дослідження математичного підгрунтя тривимірної графіки;
- дослідження основ анімації;
- аналіз створення процедурної анімації для руху об'єктів;
- розробка архітектури та інтерфейсу для гри.

Об'єкт, методи й засоби розроблення. Об'єктом розроблення комп'ютерної гри є практичне використання процедурної анімації для руху.

Розробці гри передувало дослідження математичного підгрунтя для створення реалістичної анімації руху моделей. Основу для цього склав аналіз диференційного другого порядку та налаштування відповідних параметрів даного рівняння.

У якості ігрового рушія було обрано Unity3D – безкоштовний потужний двигун, який використовується для створення 2D та 3D ігор. Мова програмування була обрана C#, оскільки вона є сумісною з Unity, а її можливості дозволяють створити необхідний функціонал.

Можливі сфери застосування. Дане дослідження може використовуватись в навчальному процесі з основ комп. графіки та анімації разом з університетською дисципліною диференціальних рівнянь або для саморозвитку людей, які зацікавлені в темі процедурної анімації. Розроблена гра може слугувати практичним прикладом застосування теоретичних знань.

1 ОГЛЯД ОСНОВНИХ ПРИНЦИПІВ 3D ГРАФІКИ ТА АНІМАЦІЇ

1.1 Історія розвитку 3D графіки та анімації

Області 3D графіки та анімації пройшли довгий та важкий шлях з початку їх створення. Зі звичайних каркасних моделей до захоплюючих подих шедеврів, які ми можемо спостерігати сьогодні, ці сфери зазнали неймовірних змін протягом останніх десятиліть. Розглянемо детальніше їхній розвиток, починаючи з другої половини XX століття, а саме такі періоди:

- **період з 1960-1970 років.** Започаткування комп'ютерної графіки було покладене у 60-х роках минулого століття, коли інформатик Айвен Сазерленд, якого ще називають батьком комп'ютерної графіки, розробив інноваційну програму, яка називається Sketchpad. Ця програма представила концепт взаємодії з графікою за допомогою пера. На початку 1970-х років, американський інформатик Едвін Кетмелл представив світу текстурування – техніка надання поверхні тривимірної деталі кольору, блиску або інших властивостей, для, наприклад, імітації певного природного матеріалу. Ці нововведення заклали фундамент для розвитку одних з найважливіших галузей сучасності;
- **каркасне моделювання.** 1970-ті роки в плані розвитку тривимірної графіки стали важливим етапом у зв'язку з появою каркасних моделей. Вже згаданий раніше Айвен Сазерленд та Девід Еванс стали засновниками цього важливого відкриття. Каркасні моделі включали в себе створення тривиімрних об'єктів за допомогою з'єднання ліній, формуючи багатокутники, які стали формою та структурою для віртуальних моделей. Ця проста на перший погляд техніка стала вирішальною, поява якої стала сходинкою для подальшого розвитку сфери комп'ютерної графіки;

- **вихід на сцену рендерингу.** У 1980-х роках відбувся помітний прогрес у тривимірній графіці. У свою чергу, це стало б неможливим без розвитку рендерингу. Визначним моментом в цьому стало представлення методу затемнення за Фонгом, який був розроблений Буйєм Туонгом Фонгом. Затемнення за Фонгом допомогло в більш реалістичному представленні освітлення поверхонь тривимірних об'єктів, що покращило їхній зовнішній вигляд. Не можна не взяти до уваги паралельний розвиток обладнання та алгоритмів растеризації, що прискорили процес рендерингу та дозволили мати більш комплексні та детальні сцени в реальному часі;
- **полігональне моделювання та анімація.** Останнє десятиліття 20 століття стало важливим пунктом у створенні полігонального моделювання та анімації. Також у цей період світ побачив вихід для загального користування 3D Studio та Maya, що стали революційними моментами у розвитку індустрії, надаючи можливості для моделювання, текстурування та анімування об'єктів. Застосування полігонального підходу в побудові тривимірних моделей дозволило максимально апроксимувати їх до реального світу, що є надзвичайно можливим для відчуттів реалістичності користувачів;
- **покращення інтерактивності та реалістичності.** З початком нового тисячоліття в розвитку комп'ютерної графіки відбулись помітні зміни. Все це пов'язано з метою досягти максимальної реалістичності та інтерактивності. З появою технік PBR (Рендеринг на основі фізики), якість освітлення, матеріалів та відбиття значно покращилось. Великого розвитку зазнала також сфера комп'ютерних ігор, яка зіграла надзвичайно важливу роль в покращенні рендерингу в реальному часі, при цьому ігрові рушії ставали все потужніше і дозволяли створювати все кращий досвід

для користувачів. Варто зазначити, що поява на світ віртуальної (VR) та доповненої (AR) реальності допомогла людям взаємодіяти з тривимірними просторами зовсім новими способами. Це дозволило людям не лише отримувати унікальний досвід, перебуваючи в незабутніх локаціях своєї улюбленої гри, а й краще відчути особливості своїх творінь для, наприклад, архітекторів та дизайнерів.

Підсумовуючи, хочеться зазначити, що комп'ютерна графіка та анімація пройшли надзвичайно цікавий та непростий шлях до того виду, який ми можемо побачити сьогодні. Від каркасних моделей до неймовірних віртуальних об'єктів, від комплексних математичних обрахувань до доступності кожному бажаючому стати частиною цієї галузі. З кожною сходинкою розвитку, комп'ютерна графіка отримувала все нові обороти, які здавались неможливими раніше. Ніхто не знає, що очікує нас далі в цій нетривіальній подорожі, але можна з впевненістю сказати, що нас очікують нові захоплюючі етапи в майбутньому.

1.2 Моделювання 3D об'єктів

1.2.1 Методи моделювання

Як вже було описано вище, 3D графіка дозволяє нам створити будь-які об'єкти потрібної нам форми, але це не було б можливим без моделювання. Згідно з ресурсом [7], для моделювання існують декілька методів, кожен з яких має свої особливості.

Почнемо з полігонального моделювання як найпоширенішого та універсального методу. Тривимірні об'єкти створюються за допомогою з'єднання вершин для утворення багатокутників, такі як чотирикутники чи трикутники. Наприклад, три вершини, що об'єднані ребрами утворюють найпростіший полігон в Евклідовому просторі – трикутник. Більш складні

полігони утворюються з декількох трикутників або як один об’єкт з кількістю вершин більше ніж три. Такий метод дозволяє ефективно представляти фігури, забезпечуючи при цьому достатню гнучкість. Полігональні моделі ідеально підходять для зображення органічних та твердих поверхонь, що робить їх підходящим варіантом для створення персонажів, середовищ та архітектурній візуалізації. Однак, варто згадати і про недостаток цього метода моделювання, а саме, полігони не взмозі точно відобразити зігнуті поверхні, тому для приблизної апроксимації знадобиться їхня велика кількість. У свою чергу це призведе до збільшення часу оброблення такої моделі та відповідно зниження швидкості.

Перейдемо до наступного методу моделювання – процедурного моделювання. Процедурне моделювання це лише загальна назва для цілого ряду технік створення тривимірних моделей та текстур у 3D графіці. Даний метод використовує набір правил, щоб генерувати потрібні об’єкти, наприклад фрактали. Набір правил може бути вбудованим в алгоритм або творець може сам налаштовувати параметри потрібним йому чином для отримання бажаного результату. Такий підхід дозволяє зберегти час розробникам при створенні моделей, оскільки позбавляє їх потреби в створенні об’єктів власноруч.

NURBS (Non-uniform rational b-splines) моделювання являє собою математичний підхід до тривимірного моделювання. Зокрема ідеально підходить для генерування та подання гладких та кривих поверхонь. Форма поверхні визначається точками контролю та математичними формулами. Такий метод моделювання найбільш підходить для створення об’єктів, в яких важливий точний контроль над кривизною.

Ще одним з найосновніших методів моделювання є метод моделювання підрозділів, або ж subdivision modeling. Його суть полягає в тому, що на початку опрацьовується звичайна полігональна сітка, яка поступово покращується розділяючи багатокутники на менші частини. Такий метод дозволяє митцю створити комплексні та деталізовані моделі. За допомогою додавання більшої кількості рівнів розділів полігона можна відобразити гладкі

та органічні форми, що робить цю техніку зручною для працювання з персонажами та деталізованими об’єктами.

Останній і досить молодий метод моделювання – моделювання скульптингом. Він, по суті, відображає метод скульптування в реальному житті, тобто дозволяє створити модель розтягуючи, розгладжуючи, щипаючи, відтягуючи цифровий об’єкт, наче він зроблений з глини. Така техніка найкраще проявляє себе, якщо на меті стоїть створення моделі персонажів чи будь-яких інших істот, або інших природніх елементів.

Кожен з наведених вище методів дозволяє якнайкраще виконати задачу, для якої він підходить. Від універсальності полігонального моделювання, до точності NURBS та природності скульптування, кожна техніка має свої переваги та недоліки. Освоївши риси кожного методу, людина може створити будь-що, обмежуючись лише власною уявою.

1.2.2 Процес генерування полігональних сіток та текстурування.

1.2.2.1 Генерування полігональних сіток.

Генерування полігональних сіток це процес створення сіток, тобто ділення нескінченного геометричного простору на скінченні геометричні та топологічні клітини. Сітки створюються за допомогою комп’ютерних алгоритмів, часто за керівництвом людини через графічний інтерфейс користувача, та залежать від складності поставленої задачі.

Полігональні сітки використовують для рендерингу та фізичної симуляції, такої як метод скінченних елементів та обчислювальна гідродинаміка. Рендеринг детальніше буде обговорений в наступних розділах. Щодо фізичної симуляції, в декількох словах, метод скінченних елементів використовується здебільшого в механічній інженерії, наприклад аероіндустрія, автомобільна індустрія, морська тощо. Даний метод дозволив значно покращити стандарти інженерних проектів завдяки можливості

проектування, відлагодження та оптимізації продукції перед її випуском. Що стосується обчислювальної гідродинаміки, то це галузь, що займається аналізом та вирішенням проблем, що пов’язані з рухом рідин. Це у свою чергу доводить, що полігональні сітки є корисними в багатьох сферах діяльності.

Як вже було сказано вище, сітки складаються з найпростіших клітин, такі як трикутники, оскільки, у випадку з комп’ютерною графікою, ми можемо проводити над ними операції трасування променів. З іншого боку, прогрес ще не дійшов до того, щоб проводити такі операції над складними просторами та фігурами.

Перед тим, як говорити про техніки створення сіток, варто зазначити, що багато таких технік побудовано на принципах триангуляції Делоне, разом з правилами додавання вершин, таких як алгоритм Руперта. Для того, щоб краще це зрозуміти розглянемо детальніше поняття триангуляції.

Триангуляція це процес за допомогою якого полігон розбивається на трикутники, що використовують тей самий масив вершин та колективно покривають одну й ту саму зону. Перед тим, як потрапити до графічного обладнання полігони мають пройти процес триангуляції. Нехай кількість вершин полігона дорівнює n, тоді даний багатокутник буде розбитий на $n - 2$ трикутника. Опуклі багатокутники досить легко піддаються такому процесу, варто лише вибрати одну вершину і під’єднати ребра до решти несуміжних вершин, щоб сформувати «віяло» з трикутника, як зображено на рисунку 1.1.

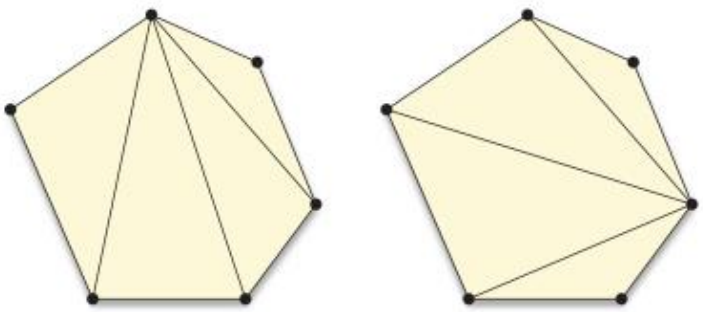

Рисунок 1.1 – Триангуляція та створення «віяла»

Для увігнутих многокутників та многокутників, які мають три або більше колінеарних вершин ситуація інша. Вони не можуть бути триангульовані таким шляхом, доводиться застосувати комплексніший алгоритм, який описано в роботі [1]. Цей алгоритм прийматиме на вхід масив *n* вершин, що знаходяться у протигодинниковому напрямку та видаватиме на виході *n – 2* трикутників. На кожній ітерації ми шукатимемо набір з трьох послідовних вершин, для яких відповідний трикутник не є виродженим. Такий трикуник називають вухом (ear). Щойно вухо було знайдене, трикутник вилучається, а центральна вершина прибирається з наступних ітерацій. Алгоритм повторюється, аж доки не залишиться три вершини. Процес зменшення розмірів триангуляції шляхом вилучання вух називається обрізанням вух.

Повернемось до методів створення сіток. Розглянемо найосновніші техніки:

- **алгебраїчні методи.** Генерація сітки алгебраїчними методами базується на інтерполяції. Ця техніка виконується використовуючи відомі функції в першому, другому та третьому вимірах, що приймають області довільної форми. Головна перевага цих методів полягає в тому, що вони забезпечують явний контроль над формою сітки(grid) та інтервалами в ній;
- наступний на черзі **метод диференціальних рівнянь**. Так само, як і вище описаний алгебраїчний метод, метод диференціальних рівнянь використовується для створення сіток (grid). Плюсом даної техніки є те, що вирішення рівнянь з створення сіток(grids) може бути використано для генерації сіток(meshes). Створення сіток(grid) може виконуватись усіма трьома класами диференціальних рівнянь з частинними похідними: еліптичного типу, гіперболічного типу та параболічного типу;
- **варіаційний метод.** Цей метод включає в себе техніку, що мінімізує гладкість сітки, ортогональність та зміну об’єму. Ця

техніка формує математичну платформу для вирішення проблем з генерацією сітки. Дана техніка є дуже потужною, однак її головним недостатком є те, що витрачається багато ресурсу на на вирішення рівнянь, пов'язаних з сіткою, що у свою чергу навантажує центральний процесор.

1.2.3 Процес текстурування.

Текстурування – процес накладання текстури на комп'ютерно згенеровану графіку, з метою надання їй кольору, блиску та інших властивостей. Розглянемо детальніше основні процеси текстурування.

Почнемо з найосновнішого моменту – **UV-перетворення (mapping)**. UV-перетворення – процес проектування поверхні тривимірної моделі на двовимірне зображення. Він полягає в присвоєнні двовимірних координат поверхні тривимірного об'єкта. Обчислення рендерингу використовує ці координати, щоб визначити, як пофарбувати 3D поверхню. Як приклад, зображено UV-перетворення для куба на рис. 1.2.

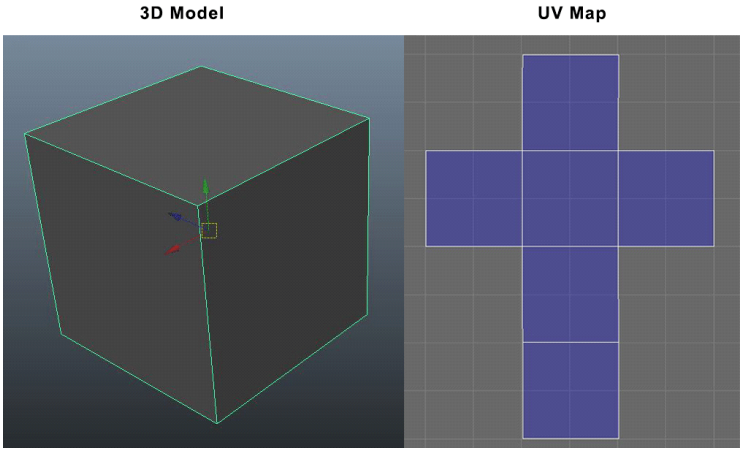

Рисунок 1.2 – UV-перетворення для куба

Перейдемо до типів карт, які існують. **Текстурна карта** – картинка, яка застосовується на поверхню моделі або до її полігону. Текстурні карти можуть

бути в першому, другому або треьому вимірі, щоправда, другий вимір є найбільш вживаним. Вони містять в собі RGB дані, мова про які буде йти пізніше, та деколи окремий канал альфа (A) для позначення прозорості. Кожен з цих типів використовується, щоб покращити вигляд тривимірних об’єктів. Ось деякі з них:

- карта базового кольору. Визначає базовий колір або вигляд поверхні об’єкта. Зазвичай створюється за допомогою програмного забезпечення і представляє первинну візуальну інформацію для об’єкта;
- карта відбиття. Цей тип контролює особливості відбиття поверхні. Визначає, як світло взаємодіє з об’єктом, впливаючи, наприклад, на блиск. Ця карта складається з чорно-білих пікселів. Світліший піксель символізує більшу здатність матеріалу до відбиття світла і більший відблиск поверхні. Темніший піксель, відповідно, з точністю навпаки;
- карта нормалей. Дана карта використовується для надання деталей до поверхні, цим надаючи більше нерівностей до неї, що у свою чергу збільшує реалістичність текстури. Карти нормалей відображають поведінку взаємодії зі світлом поверхні, створюючи ілюзію глибини. Головною її перевагою є деталізація моделі без додавання більшої кількості полігонів.

Нарешті ми підійшли до самого моменту **текстурування**. Як уже було описано вище, це процес накладання текстур на поверхню об’єкта за допомогою UV-координат. Простими словами, його можна описати як нанесення розмальованого паперу на звичайну коробку. Існує багато технік для текстурування, серед яких явне призначення атрибутів вершин, таких як позиція, колір, відбиття, UV-координати, вектори нормалей, які вручну налаштовують у пакетах для тривимірного моделювання за допомогою засобів

розгортки. До цього переліку технік також входять планарне проектування, циліндричне та сферичне відображення.

Фільтрування текстур. Дане фільтрування використовується, щоб покращити якість та гладкість текстур під час рендерингу на поверхні об’єкта. З існуючих методів фільтрування можна виділити інтерполяцію методом найближчого сусіда, білінійну та трилінійну інтерполяцію. Вони використовуються для інтерполяції між текселями (текстурними пікселями) для досягнення кращого вигляду моделі.

Останній пункт з підрозділу текстурування варто приділити **запіканню**. Запікання використовується для покращення продуктивності, особливо під час рендерингу в реальному часі для комп’ютерних ігор. Основною суттю запікання є перенесення інформації з комплексних, високо деталізованих моделей на простіші, нижче деталізовані моделі. Такий процес дозволяє зберігати візуальні особливості деталей, при цьому не втрачаючи в продуктивності.

1.2.4. Порівняння різних форматів збереження 3D об’єктів.

Без сумнівів, найважливішим етапом роботи з тривимірною графікою є збереження, обмін та використання створених моделей. Було винайдено чимало файлових форматів для збереження 3D об‘єктів та їхнього продуктивного подальшого використання. У цьому підрозділі буде досліджено та проаналізовано найпопулярніші формати, а також підкреслено їхні недостатки та переваги.

Формат RAW. RAW – простий базовий файловий формат для зберігання даних тривимірних об’єктів. Він відображає інформацію про геометрію об’єкта, наприклад розташування вершин, нормалей та текстурних координат. Файли такого формату не закодовуються та не стискаються, що робить їх розмір менше, порівняно з деякими іншими. Даний формат не може підтримувати анімації та інформацію про матеріали.

Формат OBJ. Широкозастосований та універсальний формат для збереження тривимірних моделей. Його простота та сумісність роблять його доступним для ручного редагування, оскільки він є читаємим для людини. Також цей формат підтримує зберігання геометричних даних, UV-координат та даних щодо матеріалів. З мінусів варто підкреслити відсутність підтримки зберігання анімацій, скелетної анімації зокрема та інформації щодо сцени.

Формат FBX. Цей формат здобув славу через свою універсальність, підтримуючи при цьому геометричні дані, анімацію та скелетну анімацію. Він зберігає складні ієрархії та дані про сцену, що робить його зручним для використання в комплесних проектах. Варто зазначити, що він також підтримує зберігання даних про текстури та матеріали. Однак, FBX є власним форматом, це означає, що його сумісність з різними програмними пакетами може відрізнятись. Ще одним з мінусів є великі розміри файлів такого формату, оскільки вони містять багато комплексних даних.

Формат PLY. Даний формат є гнучким та, як і інші описані вище формати, підтримує зберігання геометричних даних, кольору, прозорість та інших атрибутів. PLY формат може зберігатись у двох видах: бінарному або ASCII, що дозволяє зберігати ці файли і потім обмінюватись ними ефективно. Серед недостатків даного формату є те, що він не підтримується так загально, як інші, та має досить великі розміри файлів.

Підсумовуючи, варто взяти до уваги, що кожен формат має власні особливості і найкраще застосовується в окремих моментах. Деякі формати не підтримують анімацію, наприклад OBJ, але його висока сумісність та зручність читання компенсують це. Великі розміри FBX файлів можуть відлякувати, але цей формат може зберігати багато комплексної інформації, необхідної для подальшого використання у великих проектах. Формат PLY, у свою чергу, зберігає файли у двох виглядах, бінарному та ASCII. RAW формат малий у розмірах та може використовуватись у низькорівневих обмінах даними.

1.3 Рендеринг

Рендеринг – це процес перетворення тривимірних моделей та сцен у двовимірні зображення або анімації. Картинка на виході називається рендер. Рендеринг включає в себе обробку кольорів, геометрії, текстур, тіней, положення віртуальної камери і інших ефектів та даних, що складають собою файл сцени, який потім передається в рендер програму, щоб на виході отримати цифрове зображення.

Рендеринг вважається одним з головних підрозділів тривимірної комп'ютерної графіки. Це останній і надважливий крок в графічному конвеєрі (pipeline), що дає моделям та анімаціям їхній фінальний вигляд.

Рендеринг використовується в ряді галузей, таких як архітектура, симуляції, візуальні ефекти. На мою думку, найбільший вплив рендеринг має на галузь комп'ютерних ігор.

У випадку 3D графіки, сцени можуть бути відрендерені двома шляхами: завчасно відрендерені або рендер у реальному часі. Перший варіант широко застосовується у фільмах, оскільки він займає багато часу, але на виході дає надзвичайно якісну картинку. З іншого боку, рендеринг у реальному часі генерує не такий красивий результат, але застосовується в іграх через його швидкість.

1.3.1 Основні методи рендерингу

Розглянемо найосновніші техніки рендерингу, що використовуються у комп'ютерних іграх. Згідно з електронним ресурсом [15] та роботою [1], серед них можна виділити наступні.

Растеризація. Растеризацією називається процес перетворення векторного зображення у растрове. У сучасних ігрових рушіях використовується саме такий метод рендерингу через його надзвичайну швидкість. Хоча і результати не досягли таких вершин, як за допомогою

техніки трасування променів, все ж ця техніка вважається однією з найкращих. За допомогою растеризації, об’єкти на екрані створюються з сітки, яка складається з трикутників або полігонів, які є основою тримірної моделі. У кожного трикутника є свої вершини, які містять багато інформації, наприклад розташування в просторі, колір, текстура і так далі. Потім комп’ютер конвертує трикутники відповідних 3D моделей в пікселі на 2D екрані. Кожному пікселю може бути присвоєний початковий колір, який визначається даними, що знаходяться у вершинах трикутника.

Трасування променів. Найпростіший спосіб зрозуміти, що таке трасування променів, це вийти на вулицю і поглянути навколо себе. Об’єкти, які людина може побачити освітлюються за допомогою променів світла. Якщо ми прослідкуємо траєкторію променю з наших очей до об’єкта, який освітлюється, це і буде трасування променів. Комп’ютерне обладнання не достатньо потужне на сьогоднішній день, щоб використовувати цю техніку в реальному часі, тобто в комп’ютерних іграх. Частіше за все цей метод використовується в фільмах, де всі кадри рендеряться завчасно, тобто офлайн. Саме через це у комп’ютерних іграх використовується растеризація.

Взагалі термін трасування променів відноситься до будь-якого алгоритму, який відслідковує промені світла, щоб визначити з якими об’єктами у віртуальному світі він взаємодіє. Прикладне застосовування цього є генерація карти освітлення, визначення видимості, виявлення колізій.

Проблема знайдення точок перетину об’єкта та променя визначається наступним чином.

$$P(t) = S + tV \tag{1.1}$$

Рівняння 1.1 – рівняння прямої, що перетинає поверхню. Треба визначити корені багаточлена *P(t)* степеня n. Для плоских поверхонь степінь полінома дорівнює одиниці. Для таких поверхонь типу сфери чи циліндра степінь полінома дорівнює двійці, а отже рішення полягає в розв’язанні квадратичного рівняння. Для більш комплексних фігур, таких як тор чи сплайн, степінь полінома буде три або чотири. У такому випадку ми все ще

можемо знайти рішення аналітично, проте витративши значно більше ресурсів. Варто згадати про метод Ньютона, який також використовується в трасуванні променів.

1.3.2 Освітлення, тінь, відбиття

Освітлення, або ілюмінація – терміни, що використовуються для опису процесу, під час якого визначається колір та інтенсивність світла, що досягає поверхні. Поняття тіні або затінення, зазвичай описує методи визначення кольору та інтенсивності світла, що відбивається в сторону спостерігача, для кожної точки поверхні.

Взаємодія світла та поверхні – надзвичайно складний фізичний процес і щоб змоделювати дану взаємодію, треба буде витрачати занадто багато обчислювального часу. Натомість, використовуються прості моделі, що лише апроксимують поведінку даної взаємодії.

1.3.2.1 RGB-модель

Людське око, більшою мірою, чутливе до трьох кольорів, а саме: червоний, зелений та блакитний. Таким чином, телевізори та комп'ютерні дсиплеї можуть відображати широкий спектр кольорів змішуючи червоний колір, зелений та блакитний у різних пропорціях. Ця система зазвичай називається RGB колір. Всі моделі світла, які будуть описані далі, використовують RGB-модель.

Кольори виражаються як кортежі довжини три, що складаються з компонентів червоного, зеленого та блактиного кольору, чиї значення знаходяться в інтервалі від нуля до одного. Ці компоненти відображають не лише колір, який сприймає око, а й інтенсивність світла. Отже, колір C ми можемо позначити наступним чином:

$$C = (C_r, C_g, C_b)$$

Колір C може бути помножений на певний скаляр s, щоб отримати новий колір (1.3). Варто зазначити, що додавання та множення кольорів відбувається покомпонентно (1.4, 1.5).

$$sC = (s * C_r, s * C_g, s * C_b) \quad (1.3)$$

$$C + D = (C_r + D_r, C_g + D_g, C_b + D_b) \quad (1.4)$$

$$C * D = (C_r * D_r, C_g * D_g, C_b * D_b) \quad (1.5)$$

Множення кольорів, на скаляр або на інший колір, ще називається модуляцією. Колір пікселя, що належить відрендереному трикутнику, зазвичай визначається через деяку комбінацію кольорів з декількох джерел. Колір пікселя, що лежить на грані трикутника, зазвичай походить від добутку кольору, що знаходиться в текстурній карті та іншого кольору, інтерпольованого поміж вершинами трикутника.

1.3.2.2 Джерела світла

Колір, який ми обраховуємо для будь-якої точки на поверхні – це сума вкладів усіх джерел світла, що освітлюють поверхню. Стандартні типи джерел світла, що підтримуються тривимірними графічними системами, можуть бути чотирьох видів.

Навколишнє світло. Навколишнє світло представляє собою світло, що спрямоване в усі сторони, має встановлену інтенсивність та впливає на всі об’єкти на сцені однаково.

Спрямоване світло. Також відоме, як нескінченне джерело світла. Вид освітлення, що випромінює світло в одному напрямку нескінченно. Зазвичай використовуються для моделювання сонця в сценах.

Точкове світло. Вид освітлення, що випромінює світло в кожному напрямку з однієї точки в просторі. Інтенсивність світла зменшується, якщо об’єкт знаходиться далі від джерела світла.

Нехай точкове світло було розташоване в точці P у просторі. Інтенсивність світла C, що досягає точки Q, буде розраховуватись наступним чином:

$$C = \frac{1}{k_c + k_l d + k_q d^2} C_0,$$

де C_0 – колір світла;

d – дистанція між джерелом світла та Q, тобто Евклідова відстань;

k_c, k_l, k_q – константне, лінійне та квадратичне згасання відповідно.

Прожекторне світло. Прожекторне світло схоже на точкове, але має напрямок випромінювання. Інтенсивність світла згасає в залежності від відстані, так само як і в точкового світла.

Нехай прожектор помістили в точці P та він має напрямок R. Інтенсивність C світла, що досягає точки у просторі Q виглядає наступним чином:

$$C = \frac{\max(-R * L, 0)^p}{k_c + k_l d + k_q d^2} C_0, \quad (1.7)$$

де C_0 – колір світла;

d – дистанція між джерелом світла та Q, тобто Евклідова відстань.

k_c, k_l, k_q – константи згасання світла;

L – напрям одиничної довжини, що вказує з Q до джерела світла.

$$L = \frac{P - Q}{||P - Q||} \quad (1.8)$$

p – експонента, що контролює, наскільки концентроване світло прожектора. Чим більше значення p, тим більш сконцентрований буде прожектор, це можна буде побачити на рисунку 1.3. Очевидно, що освітлення буде найбільш яскравим, якщо $R = -L$ і буде поступово спадати, якщо кут між R та $-L$ зростатиме. Якщо кут між ними більше за 90 градусів, то освітлення від прожектора не буде досягати цілі.

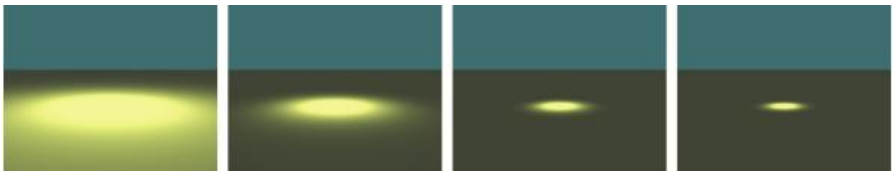

Рисунок 1.3 – Змінення значення експоненти p від меншого до більшого, зліва направо відповідно

1.3.2.3 Відбиття світла

У комп'ютерній графіці найпоширенішими є два типи відбиття: **розсіяне** та **дзеркальне** відбиття. У цьому дослідженні детально описано буде лише розсіяне відбиття світла.

Розсіяне відбиття світла. Розсіяна поверхня — це поверхня, на якій частина світла, що падає на точку поверхні, розсіюється у випадкових напрямках. Зазвичай ефект полягає в тому, що певний колір світла, колір розсіяного відбиття поверхні, відбивається рівномірно в усіх напрямках. Це називається Ламбертовим відбиванням і тому що світло відбивається рівномірно в усіх напрямках, вигляда Ламбертового відбивання не буде залежати від позиції спостерігача.

Розглянемо рис. 1.4. Промінь, що має площину поперечного перерізу A, освітлює цю саму площу A тільки тоді, коли поверхня перпендикулярна напряму, в якому рухається світло. Зі зростанням кута між нормальним вектором поверхні та напрямом світла, збільшується площа освітлення променем світла. Нехай кут між нормальним вектором та напрямом світла дорівнює θ, тоді площа, що освітлюється променем дорівнюватиме $A/cos\ \theta$. Величина cos θ визначається скалярним добутком між нормальним вектором N та одиничним вектором напрямку до джерела світла L. Якщо скалярний добуток менше нуля, тоді це означає, що поверхня не повинна бути освітлена, оскільки вона розвернута від джерела світла. Таким чином варто затиснути цей добуток так, щоб він не приймав негативних значень.

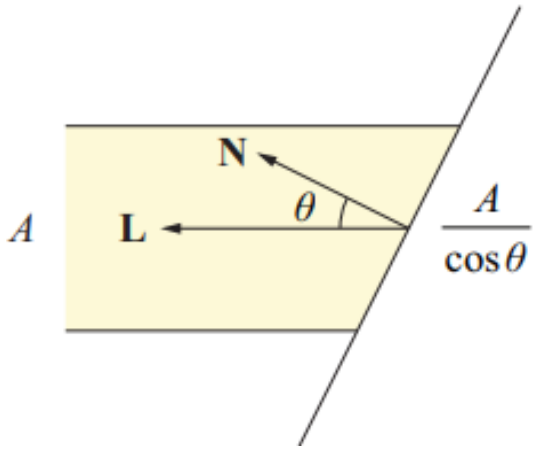

Рис 1.4 – Розсіяне відбиття світла

Нехай маємо точку Q – точка на поверхні, з якої відбивається світло в сторону спостерігача. Для фінальної формули врахуємо інтенсивність C_i кожного з n джерел світла, що освітлюють Q. Формули знаходження інтенсивностей для різних видів освітлення були описані вище, а саме формули 1.7 та 1.8. Відбите світло множиться на колір D розсіяного відбиття поверхні. Додаючи внесок від n джерел світла і враховуючи навколишню яскравість A, можемо виразити дифузний компонент нашої формули:

$$K_{diffuse} = D * A + D \sum_{i=1}^{n} C_i \max\{N * L_i, 0\},$$

де $L_i -$ одиничний вектор, що вказує з точки Q до $i -$ того джерела світла.

1.3.2.4 Затінення

У комп'ютерній графіці затінення стосується зміни кольору об'єкта, поверхні або полігону у тривимірній сцені, основуючись на відстані від світла, куті до камери і інших властивостях, щоб досягти фотореалістичного ефекту. Розглянемо два методи затінення, які використовуються у комп'ютерній графіці.

Затінення за Гуро. Інтерполяційний метод, що використовується для створення неперервного затемнення поверхонь, що відображаються полігональними сітками. Найчастіше використовується для здобуття неперервного освітлення на трикутних сітках(meshes), обраховуючи освітлення на вершинах кожного трикутника і лінійно інтерполюючи отримані кольори для кожного пікселя, що покривається трикутником. Затінення за Гуро вважається кращим, ніж рівномірне затінення та вимагає меншого обчислення, ніж затемнення за Фонгом, проте в нього є і свої мінуси, наприклад гранчастий кінцевий вигляд моделі.

Затінення за Фонгом. Цей метод затінення інтерполює нормальні вектори поверхонь між растеризованими полігонами і обраховує кольорі пікселів, спираючись на інтерпольовані нормалі та модель відбиття. Дана техніка перевершила затінення за Гуро в тому, що вона представляє кращу апроксимацію затінення гладкої поверхні. Також затемнення за Фонгом працює краще, коли застосовується до моделі відображення з невеликими відблисками.

1.4 Основні принципи анімації

Анімація – процес надання життя статичним об’єктам у комп’ютерній графіці. Головною ідеєю комп’ютерної анімації є програвання визначених картинок з більшою швидкістю, щоб обманути глядача, змусивши його інтерпретувати ці картинки як неперервний рух зображень. Комп’ютерна анімація є цифровим наступником технік лялькової анімації, використовуючи при цьому тривимірні об’єкти.

Для тривимірних анімацій, об’єкти спочатку моделюються. Потім в них встановлюється віртуальний скелет. За допомогою нього аніматор рухає потрібні частини тіла моделі, використовуючи при цьому **ключові кадри,** мова про які піде дещо пізніше. Відмінності у вигляді між ключовими кадрами

автоматично обраховується комп’ютером, цей процес також відомий як морфінг або твінінг. Фінальною дією у створенні анімації є рендеринг.

1.4.1 Ключові кадри

Ключові кадри є одним з найважливіших концептів у анімації. Вони визначають конкретні точки у часі, під час яких повинні відбуватись зміни у розташуванні, повороті чи масштабуванні об’єкта. Визначаючи ключові кадри анімації об’єкта, аніматор має змогу створити гладкий та неперервний рух між ними. Тобто, вони діють як ключові пози у певній анімації, за допомогою яких об’єкт може рухатись.

Наприклад, при анімації персонажа, ключові кадри можуть бути виставлені у моменті найвищої точки стрибку, в момент удара по м‘ячу чи при зробленому кроці. Останній варіант можна побачити на рис. 1.5.

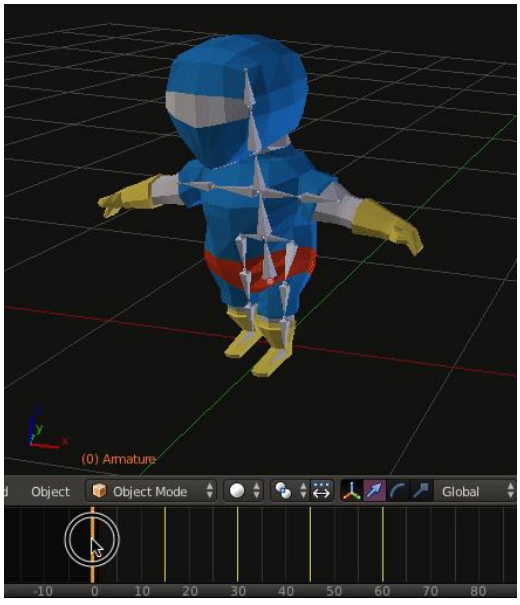

Рис 1.5 – Ключові кадри для анімації руху

1.4.2 Інтерполяція та сплайни

Інтерполяція у комп’ютерній анімації використовується для обрахування проміжних значень між ключовими кадрами. За допомогою

інтерполяції анімація виходить плавною, що надає їй реалістичності. Існують різні види інтерполяції, кожен з яких надає свої особливості в анімуванні об’єктів. Відповідно до електронного ресурсу [17], опишемо деякі з них.

Лінійна інтерполяція створює прямий перехід між двома ключовими кадрами. Такий спосіб є простим та дієвим, хоча йому бракує деяких комплексних рис інших видів інтерполяції.

Криві Безьє. Параметрично задана крива, що використовує набір точок контролю, щоб створювати гладкі та неперервні криві. Криві Безьє є одним з найкращих способів інтерполювати між ключовими фрагментами, оскільки вони дозволяють створювати більш комплексні та природні переходи. Найчастіше використовують квадратичні та кубічні криві Безьє.

Перечислені вище методи інтерполяції лише декілька з багатьох їх видів. Кожен з різних видів інтерполяції дозволяє створити унікальну поведінку в анімації нашого об’єкта, але основною ціллю усіх технік, без якої анімація не була б такою, якою ми її знаємо, є плавний перехід між ключовими кадрами, за допомогою якого ми отримуємо реалістичну анімацію.

Сплайн – це математична крива, що забезпечує гнучкий та продуктивний спосіб створювати шляхи для об’єктів, інтерполювати рух та створювати плавні переходи між ключовими кадрами. Як уже було згадано, сплайни широко використовуються для визначення шляхів, яким будуть слідувати об’єкти чи персонажі під час анімації. Сплайн слугує як крива, що буде направляти об’єкти по необхідній траєкторії, наприклад, рух камери по сцені.

Сплайни контролюються за допомогою контрольних точок. Змінюючи позицію та інші властивості цих точок, аніматор можна змінювати характер та кривизну сплайну. Такий контроль над сплайном дозволяє маніпулювати ним в потрібному для аніматора ключі.

1.4.3 Методи анімації

Існують різні техніки комп'ютерної анімації. Відповідно до електронних ресурсів [18] та [19], розглянемо детальніше кожну з них, звернувши увагу на особливості застосування кожної:

Традиційний метод анімації, або «Кадр за кадром». Один з найдавніших методів анімації, його суть полягає в тому, щоб промальовувати кожний кадр окремо. Після цього, намальовані кадри запускаються послідовно, створюючи при цьому ілюзію руху. Даний метод займає багато часу, проте надає високий рівень контролю для артиста.

Анімація по ключовим кадрам. Даний вид анімації відбувається за допомогою розстановки ключових кадрів у потрібні моменти. Як уже було описано вище, дії між кадрами компенсуються за допомогою інтерполяції, що забезпечує плавність руху. Такий метод дозволяє аніматору контролювати переміщення, обертання та масштабування тривимірних моделей, при цьому створюючи комплексні та реалістичні анімації персонажів або навколишнього середовища.

Запис руху або Motion Capture. Ця техніка полягає у відслідковуванні рухів людини за допомогою спеціальних девайсів, їхнього запису та перетворенню в цифрову анімацію. Саме за допомогою спеціального обладнання, даний вид анімації може набути найреалістичнішого та найприроднішого вигляду, оскільки всі рухи походять напряму від людини.

Процедурна анімація. Остання з технік анімації, на якій буде зосереджено детальну увагу в подальшому – процедурна анімація. Цей метод анімації полягає у створенні анімації за допомогою математичних розрахунків та алгоритмів. Відсутність ключових кадрів у цій техніці анімації компенсується різними параметрами, за допомогою яких можна контролювати та змінювати характер анімації. На відміну від анімації за допомогою запису руху або анімації за ключовими кадрами, які використовують завчасно створені анімації, вона може генерувати анімацію у реальному часі.

2 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ПРОЦЕДУРНОЇ АНІМАЦІЇ В UNITY

Unity – потужний ігровий рушій, що надає рішення для створення ігор та інших аплікацій. Серед них є технології для роботи з анімацією, в тому числі з процедурною анімацією. Розглянемо їх детальніше.

Почнемо з систем, які не повністю пов’язані з процедурною анімацією. Одною з таких систем є **Unity Timeline**. Цей інструмент, з можливістю редагування прямо в інспекторі, дозволяє за допомогою ключових кадрів створювати анімацію та маніпулювати нею. За допомогою нього можна створювати кінематографічні вставки, кат-сцени та інші анімації, додаючи про цьому аудіоефекти. Як вже було зазначено раніше, ця технологія не була створена напряму для роботи з процедурною анімацією, проте використання їх в комбінації може призвести до неймовірних результатів.

Ще одна технологія, яка є найвідомішою та найрозповсюдженішою для роботи з анімацією – Unity Mecanim. Ця анімаційна система дозволяє створювати та налаштовувати необхідні послідовності у формі скінченного автомату, регулюючи при цьому умови переходів між анімаціями. Ще однією особливістю цієї системи є наявність дерев змішування або blend trees. Вони використовуються для плавного зміщування всіх анімацій, управління над якими здійснюється за допомогою параметра змішування, який є одним з числових параметрів анімації в Animator Controller в Unity.

Щодо рішень, які надає Unity і, на мою думку, є невід’ємними у створенні процедурних анімацій, можна виокремити Unity Animation Rigging та можливість створення власних скріптів для керування анімацією. Щодо скелетної анімації, то це вбудований в Unity пакет, що надає можливості гнучкого налаштування процедурних анімацій. Ця технологія дозволяє створювати комплексну та реалістичну поведінку персонажів через систему обмежень і параметрів. Animation Rigging створює поверх скелету, створеного

під час створення персонажу, ще один скелет, за допомогою якого ми перевизначимо частину анімації. Наприклад, якщо персонаж відтворює певну анімацію, але нам потрібно, щоб він головою слідкував за якимось об’єктом, то це легко зробити за допомогою пакету скелетної анімації в Unity, як показано на рис 2.1.

Рисунок 2.1 - Використання скелетної анімації в Unity, щоб змусити персонажа головою слідкувати за ціллю

Оскільки кожен об’єкт в Unity складається з компонентів, то створення власних скриптів на C#, які також є компонентами, дозволяють маніпулювати об’єкти в потрібному для розробника ключі. Це стосується і анімації. За допомогою скриптів можна використати повний потенціал скелетної анімації в Unity, комбінуючи різні види анімацій через код.

3 МАТЕМАТИЧНЕ ПІДГРУНТЯ ДЛЯ СТВОРЕННЯ ПРОЦЕДУРНОЇ АНІМАЦІЇ РУХУ

Метою даної роботи, окрім дослідження, є створення процедурної анімації руху персонажів. Однією з головних переваг процедурної анімації є відсутність ключових кадрів, це означає, що модель не буде рухатись за чітко визначеною анімацією, а буде адаптуватись під навколишнє середовище та інформацію на вході зі сторони гравця. Саме це дозволить отримати у грі найбільшу реалістичність, що у свою чергу принесе більше задоволення від неї.

Оскільки для руху було обрано саме процедурну анімацію, це автоматично означає, що ми відмовились від ідеї ключових кадрів. Одним з головних мінусів такого підходу є те, що зараз ми не можемо використати інтерполяцію, щоб додати відчуття інерції та плавності. Отже, повинен бути інший обхідний шлях, щоб додати реалістичності нашому руху.

Нехай, у нас є певна вхідна інформація x з нашого ігрового світу, що змінюється динамічно. Нашою метою буде створення на виході y, що імітує поведінку x. Як продемонстровано на рисунку 3.1 – найпростішим та найпримітивнішим варіантом для досягнення цього результату буде $y = x$, в такому випадку y буде копіювати інформацію на вході x. Для наочності розглянемо рисунки 3.1 та 3.2, які демонструють описану вище поведінку.

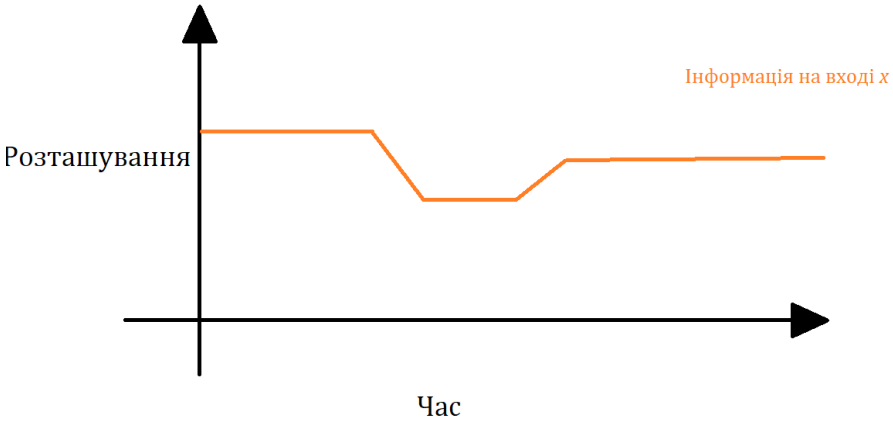

Рисунок 3.1 – Інформація на вході *x*

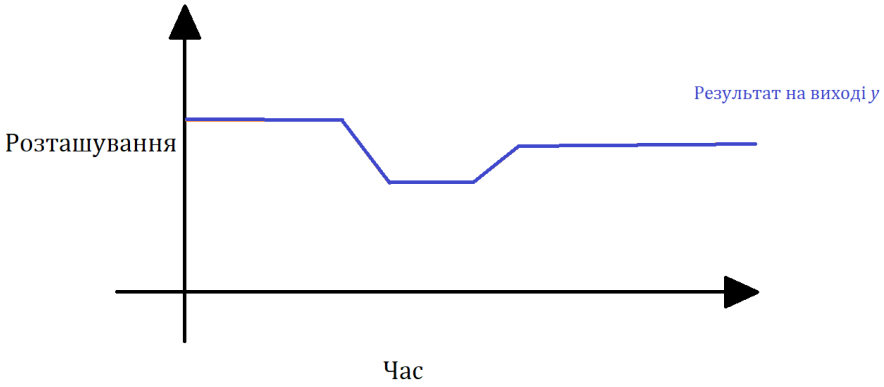

Рисунок 3.2 – Результат на виході *y*

На перший погляд може здатись, що звичайне $y = x$ не несе в собі ніякої цінності, оскільки це звичайне повторення вхідної інформації. Частково це правда, але це важлива сходинка в подальшому розвитку процедурної анімації руху.

Механічні рухи у реальному світі залежать від швидкості та прискорення. Додамо їх до нашого рівняння, та отримаємо нове рівняння:

$$y + k_1 * y' + k_2 * y'' = x + k_3 * x'$$

де y – залежна змінна, що описує відповідь системи.

x – змінна, що характеризує вхідні дані, які діють на систему.

y' – перша похідна y, що представляє швидкість.

y'' – друга похідна y, що представляє прискорення.

k_1, k_2, k_3 – константи, що впливають на динаміку системи.

Детальніше про k_1, k_2, k_3 мова буде йти дещо згодом.

Таке рівняння є диференціальним рівнянням другого порядку та дозволить нам отримати більш комплексну та природню поведінку системи, при цьому система все ще буде імітувати поведінку вхідних даних.

3.1 Коефіцієнт затухання *ζ*, частота *f*, початкова відповідь системи *r*

Як уже було сказано раніше, константи k_1, k_2, k_3 впливають на динаміку системи, проте інтуїтивно не зрозуміло, як кожен з цих параметрів повинен це робити.

Коефіцієнт затухання ζ - це коефіцієнт, що описує, як швидко коливання згасають від одного руху до іншого. Цей параметр позначається літерою ζ *(дзета)*, при ζ більшу, ніж одиниця, система називається **надзгасною**, тобто повертається до рівноваги без коливань, при ζ дорівнює нулю – **незгасна**, тобто коливається в її природній резонансній частоті, при ζ менша, ніж одиниця – **слабкозгасна**, тобто коливається з меншою, ніж незгасна, амплітудою, що прямує до нуля та при ζ дорівнює одиниці – **критично згасна** система, що повертається до рівноваги якнайшвидше, без коливань. У нашій системі він означатиме як саме система буде «сідати» на ціль.

Частота f – величина, що позначає кількість повторів за одиницю часу. Період частоти – інтервал, що символізує час між подіями. Є одним з найважливіших параметрів у згасних коливаннях та обраховується за формулою

$$f = \omega/(2\pi)$$

де ω – кутова частота.

Визначимо ще один параметр, необхідний для налаштування нашої системи, **початкова відповідь *r*.**

Це параметр, який буде позначати те, як швидко реагує система реагує на вхідні дані. Наприклад, при r дорівнює нулю системі знадобиться час, щоб перевести себе зі стану спокою в стан руху; при r більше за нуль, система негайно відповість на вхідні дані, що поступили; при r більше за одиницю система трохи перестарається з імітацією вхідних даних, проте всеодно зрівняється з *x*.

Як ми будемо визначати кожен з цих параметрів? Згадаємо наше рівняння для опису руху:

$$y + k_1 * y' + k_2 * y'' = x + k_3 * x' \quad (3.1)$$

Перетворимо дану систему у наступний вигляд, поділивши кожен терм рівняння на коефіцієнт при члені з похідною найбільшого порядку, тобто на k_2:

$$y'' + \left(\frac{1}{k_2}\right) * y + \left(\frac{k_1}{k_2}\right) * y' = \left(\frac{1}{k_2}\right) * x + \left(\frac{k_3}{k_2}\right) * x' \quad (3.2)$$

Згідно з роботою [21] система рівняння руху має наступний вигляд:

$$m * y'' + \mathrm{c} * y' + k * y = u(t) \quad (3.3)$$

Використовуючи власну частоту гармонічного осцилятора:

$$\omega_n = \sqrt{k/m}$$

і визначення коефіцієнта затухання

$$\zeta = \frac{\mathrm{c}}{\mathrm{c_c}},$$

де $\mathrm{c_c} = 2 * m * \omega_n$ – критичне затухання.

Виведемо c:

$$\mathrm{c} = \mathrm{c_c} * \zeta = 2 * m * \omega_n * \zeta = 2 * m * \zeta * \sqrt{k/m}$$

Перепишемо систему 3.3 наступним чином, поділивши рівняння на m, щоб звести систему до загального виду:

$$m * y'' + 2 * \zeta * m * \sqrt{\frac{k}{m}} * y' + k * y = u(t) \mid : m$$

Отримаємо наступне рівняння:

$$y'' + 2 * \zeta * \omega_n * y' + {\omega_n}^2 * y = u(t)/m \quad (3.4)$$

Порівняємо коефіцієнти рівнянь систем 3.2 та 3.4:

$$2 * \zeta * \omega_n = \left(\frac{k_1}{k_2}\right) \tag{3.5}$$

$$\omega_n{}^2 = \left(\frac{1}{k_2}\right) \tag{3.6}$$

Нам потрібно виразити ζ з рівняння 3.5. Для досягнення цієї цілі спочатку виразимо ω_n з рівняння 3.6.

$$\omega_n = \sqrt{1/k_2} \tag{3.7}$$

Підставимо його у наше рівняння 3.4:

$$\zeta = \frac{k_1}{2*k_2*\sqrt{1/k_2}} = \frac{k_1}{2*\sqrt{k_2}} \tag{3.8}$$

Збережемо це рівняння в такому вигляді поки що, воно знадобиться нам в наступному підрозділі.

Виведемо тепер частоту f, формула якої є $f = \frac{\omega_n}{2\pi}$. Оскільки ω_n нам відоме з рівняння 3.7, то ми можемо його застосувати, отримаємо:

$$f = \frac{\omega_n}{2\pi} = \frac{1}{2*\pi*\sqrt{k_2}} \tag{3.9}$$

Останній з параметрів, які потрібно вивести є початкова відповідь *r.* Нас найбільше цікавить права частина рівняння, а саме терм $k_3 * x'$, оскільки саме коефіцієнт k_3 буде відповідати за початкову відповідь на вхідні дані. Для вимірювання цього впливу, сформуємо співвідношення між k_3 та k_1 – коефіцієнті при похідній першого порядка у відповіді системи. Отримаємо $\frac{k_3}{k_1}$. Для посилення ефекту вхідних даних домножимо співвідношення на двійку. Отриманий вираз:

$$r = \frac{2*k_3}{k_1} \tag{3.10}$$

- відображення параметра, що контролює початкову відповідь системи та дозволить налаштувати систему для необхідних вимог при розробці.

3.2 Параметри k_1, k_2, k_3

Описавши все необхідне підгрунтя, можемо вивести потрібні нам параметри.

Почнемо з найпростішого, а саме параметр k_2, який виводиться з рівняння 3.9:

$$\sqrt{k_2} = \frac{1}{2 * \pi * f} => k_2 = \frac{1}{(2 * \pi * f)^2}$$

За допомогою k_2 зможемо вивести k_1 з рівняння 3.8 наступним чином:

$$k_1 = \zeta * 2 * \sqrt{k_2} => \frac{2 * \zeta}{2 * \pi * f} => \frac{\zeta}{\pi * f}$$

Останній на черзі параметрів - k_3. Скористаємось рівнянням 3.10 та в результаті отримаємо:

$$2 * k_3 = r * k_1 => k_3 = \frac{r * k_1}{2} => \frac{r * \zeta}{2 * \pi * f}$$

Представимо нашу систему другого порядку з параметрами у їхньому новому вигляді:

$$y + \frac{\zeta}{\pi * f} * y' + \frac{1}{(2 * \pi * f)^2} * y'' = x + \frac{r * \zeta}{2 * \pi * f} * x'$$

3.3 Методи розв’язання диференціальних рівнянь

Існує багато методів розв’язання диференціальних рівнянь, в цьому дослідженні та у розробці практичної частини буде використовуватись напівнеявний метод Ейлера.

Напівнеявний метод Ейлера – чисельний метод апроксимації, що використовується для розв’язання звичайних диференціальних рівнянь, тобто диференціальних рівнянь, що залежать від однієї змінної.

Щоб зрозуміти, як він працює, опишемо головні кроки даного алгоритму.

Для початку, нам необхідні початкові значення змінних. У нашому випадку необхідні початкові значення y, позначимо його y_0, що відповідає за розташування; та початкове значення y', позначимо його y'_0.

Під час гри, нам необхідно ітеративно оновлювати ці значення кожного фрейму. Нехай нам треба вирішити звичайне диференціальне рівняння з моменту t_0 до моменту t_1, час, що проходить між фреймами дорівнює T. Тоді матимемо n кроків, де $n = (t_0 - t_1)/T$.

Потім, починаючи з початкових значень, ми ітеративно будемо апроксимувати значення y та y'. Тобто, на першому кроці матимемо $y_0 = x_0$ та $y' = 0$. Для кожного наступного кроку від 1 до n буде робити наступне:

- нове значення $y[i+1] = y[i] + T * y'[i]$;
- нове значення $y'[i+1] = y'[i] + T * y''$;
- y'' обраховується за допомогою нашої системи 3.1.1, тобто виражаємо y'' та отримуємо $y'' = (x + k_3 * x' - y - k_1 * y')/k_2$.
- Підставляємо в y'' значення та отримуємо $y'[i+1] = y'[i] + T * (x[i+1] + k_3 * x'[i+1] - y[i+1] - k_1 * y'[i+1])/k_2$;
- Єдиною проблемою зараз лишається швидкість вхідних даних $x'[i+1]$, яке обраховується за допомогою середньої швидкості як $x'[i+1] = \frac{x[i+1]-x[i]}{T}$;
- Повторюємо даний алгоритм допоки не досягнемо бажаного результату.

Ця модифікація методу Ейлера покращує стабільність та точність результату. Звичайно, не варто забувати, що точність розв’язку залежить від розміру кроку нашого алгоритму. Чим менше крок, тим акуратніше буде результат.

4 ПРОЕКТУВАННЯ АРХІТЕКТУРИ ТА ІНТЕРФЕЙСУ ДЛЯ ПРОЦЕДУРНОЇ АНІМАЦІЇ В UNITY

4.1 Розробка архітектури системи руху за допомогою процедурної анімації

Перед тим, як перекласти описаний вище алгоритм на мову програмування, необхідно продумати архітектуру, необхідну для правильного функціонування цілої системи.

Перед тим, як говорити про анімацію взагалі, треба мати об’єкт, який ми будемо анімувати. Особливістю моєї розробки є саме чотириногі створіння. На щастя, готові моделі можна знайти на офіційному сайті від Unity Technologies, який називається Unity Asset Store. Обраний персонаж зображений на рисунку 4.1.

Рисунок 4.1 – Чотириногий роботичний павук, обраний на роль головного персонажу

Щодо необхідних скриптів, система буде складатись з допоміжного скрипту, який має назву *SecondOrderDynamics*, в якому буде описаний напівнеявний метод Ейлера та обраховані константи k_1, k_2, k_3.

Наступний скрипт, який буде «сидіти» на нашому чотириногому, це *SpiderController*, за допомогою якого ми будемо контролювати рухи павука, та виставляти значення для ζ, f, r, які будуть передані скрипту *SecondOrderDynamics*.

Цікавий момент щодо керування павуком. Оскільки наш рух побудований на принципі, що система імітує поведінку інформації на вході, то рухати ми будемо не самого павука, а створимо поле типу Vector3 та назвемо його, наприклад, *targetMovePosition*. Саме координати цього поля ми будемо змінювати і передавати їх кожного фрейму в функцію, що реалізує напівнеявний метод Ньютона. Хоча ми і не рухаємо самого персонажа, проте складається ілюзія, ніби так і відбувається.

Останній важливий скрипт, який нам буде потрібно буде називатись *LegFix*. Його основна суть в тому, щоб створювати процедурну анімацію пересування ніг павука.

Щодо необхідних технологій, то для згаданого пересування ніг нам знадобиться описана вище технологія анімаційного віртуального скелета (Animation Rigging) та інверсна кінематика.

4.2 Проектування інтерфейсу для контролю персонажа

Персонаж буде контролюватись за допомогою вводу з клавіатури гравцем клавішами WASD.

Камера, у свою чергу, буде переслідувати гравця та дивитись на нього. Це буде зроблено за допомогою пакета *Cinemachine*, який є, буквально, мізками камери.

5 РЕАЛІЗАЦІЯ СКРИПТІВ ТА АЛГОРИТМІВ. ІНТЕГРАЦІЯ В UNITY

5.1 Імплементація напівнеявного методу Ейлера

Як уже було згадано раніше, в Unity кожен об’єкт складається з компонентів, які тим чи іншим чином контролюють його поведінку. Скрипти – це власні компоненти, написані розробником, саме вони дозволяють надати гнучкість діям об’єкту.

Усі класи, що успадковуються від *MonoBehaviour* в Unity можуть бути додані як компонент до об’єкта. У випадку класа *SecondOrderDynamics*, який відповідальний за реалізацію напівнеявного методу Ейлера, то це буде звичайний клас, який не буде додаватись до об’єкта. Код цього класу буде розміщено в додатку А.

Коротко про цей клас, в конструкторі будуть ініціалізуватись константи k_1, k_2, k_3 за допомогою параметрів ζ, r, f, що будуть передаватись як аргументи. У конструкторі також ініціалізуються значення y, y' та попереднє значення *x,* яке використовується для обрахування x'.

Головний метод цього класу називається *UpdateValues.* Він приймає параметри T - відстань між кадрами, координати *targetMovePos*, згадка про які була в розділі 4.1 та опціональний параметр, що символізує похідну від вхідної інформації. Саме тіло методу – це алгоритм, описаний в розділі 3.3, перекладений на мову *C#.*

5.2 Тестування різних значень параметрів ζ, r, f

Кожен з параметрів ζ, r, f додає власну характеристику до поведінки системи. Після застосування теорії, описаної в підрозділі 5.1, імплементуємо систему другого порядку в Unity та проекспериментуємо з декількома різними значеннями параметрів.

Візьмемо, наприклад наступні параметри $\boldsymbol{\zeta = 1, r = 2, f = 0.5}$. Якщо звіритись з теорією, описаною в підрозділі 3.1, то можна відразу припустити, що рух даної системи буде: оскільки r більше, ніж одиниця, то система «перестарається» з відповіддю на вхідні дані і перескочить нашу ціль при її зупинці. ζ дорівнює одиниці, отже система критично згасна та повернеться в рівновагу без коливань, отже частота коливань f не буде грати жодної ролі. Описана вище теорія зображена на рис. 5.1:

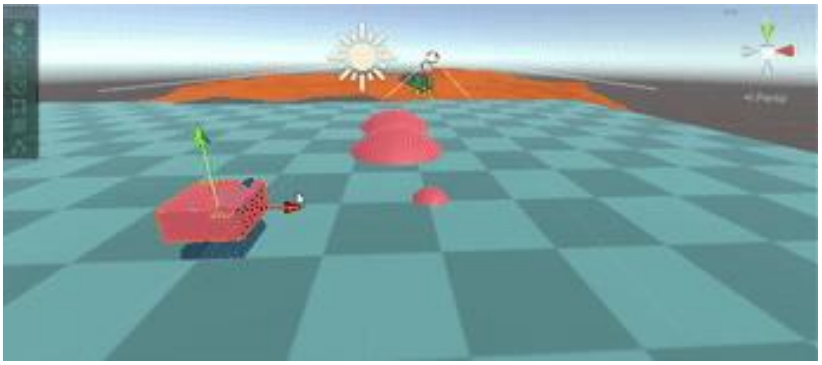

Рисунок 5.1 – Тестування системи з параметрами $\boldsymbol{\zeta = 1, r = 2, f = 0.5}$

Знайдемо ще цікаві форми поведінки нашої системи, виставивши параметри $\boldsymbol{\zeta = 0.15, r = 3, f = 1}$. Отримали наступну поведінку на рис. 5.2:

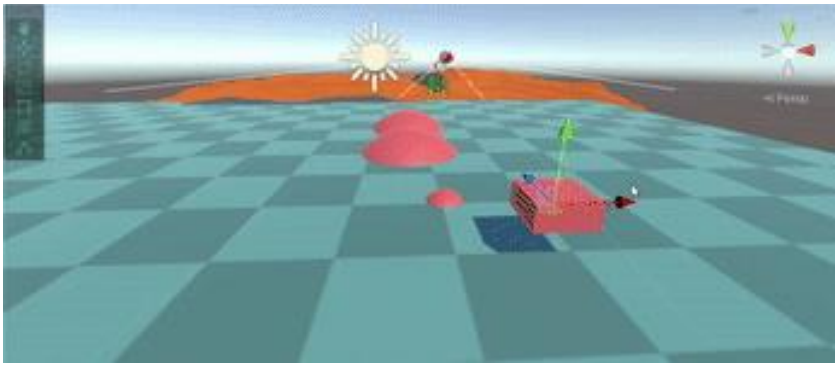

Рисунок 5.2 – Тестування з параметрами $\boldsymbol{\zeta = 0.15, r = 2, f = 1}$

Шляхом проб і помилок, визначимо нейтральну поведінку нашої системи за допомогою параметрів $\boldsymbol{\zeta = 0.5, r = 1, f = 0.5.}$ Використаємо ці параметри для опису руху нашого головного персонажа.

5.3 Реалізація класу, що контролює персонажа

Скрипт, який контролює персонажа називається *SpiderController*. Саме в ньому написана реалізація, що дозволяє рухати павука, при цьому беручи в розрахунок його швидкість та прискорення для імітації реалістичності його руху.

Unity має декілька особливих методів, які отримуються за допомогою успадкування від класу *MonoBehaviour* – методи *Start*, *Update*, *FixedUpdate*. Насправді їх дещо більше, але саме за допомогою цих методів було виконано реалізацію руху персонажа.

Метод *Start* викликається один раз за діючу гру перед її запуском, що робить його ідеальним місцем для ініціалізації даних, саме тут створюється об’єкт класу *SecondOrderDynamics* та передаються всі потрібні параметри.

Метод *Update* викликається кожного кадру, що робить його незамінним для відслідковування вводу гравця з клавіатури, миші та інших девайсів вводу. Логічно, тут відбувається відслідковування вводу гравця з клавіаутри.

Метод *FixedUpdate* схожий з попереднім методом, лише має частоту фізичної системи, тобто викликається він рідше, ніж *Update*. Він ідеально підходить для математичних обрахувань, що необхідно проводити кожного фрейму, саме через це в ньому викликається рядок коду, відповідальний за переміщення нашого персонажу

```
transform.position = instance.UpdateValues(Time.fixedDeltaTime, targetMovePos);
```

де *Time.fixedDeltaTime* – інтервал, через який оновлюється кадр у фізичній системі Unity.

5.4 Реалізація процедурної анімації руху павука та інтеграція в Unity

Розіб’ємо реалізацію процедурної анімації пересування ніг павука на такі кроки:

Перший крок – використаємо інверсні кінематику, щоб реалістично контролювати ноги павука. Рисунок 5.3 зображує, як це виглядає в Unity.

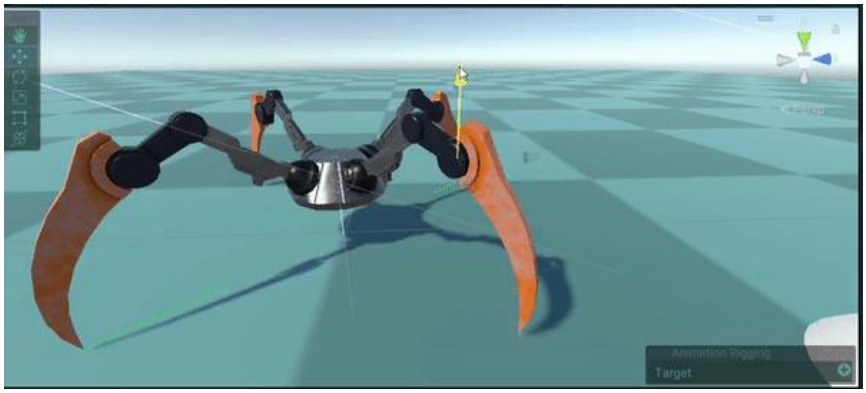

Рис 5.3 – Рух ноги павука за допомогою інверсної кінематики

За допомогою коду ми зафіксуємо нижню частину ноги на землі, це необхідний крок для подальшої реалізації анімації пересування лап.

Наступний крок – додаємо допоміжний об’єкт, що буде прив’язаний до основного тіла, тобто буде його дитиною (child object). В нашому випадку, це буде сфера. Через код додамо до неї деяку функціональність, а саме: з низу сфери буде виходити промінь, за допомогою якого сфера зможе рухатись по різних поверхнях. Це необхідно для того, щоб під час гри наш персонаж міг рухатись по нерівних поверхнях. Рисунок 5.4 демонструє цю поведінку:

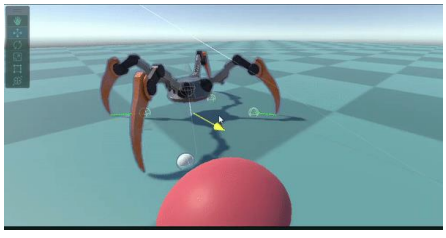

Рисунок 5.4 – Реалізація другого кроку, без фіксування лап до землі

Насправді, допоміжна сфера нам необхідна не лише для того, щоб мати змогу ходити по різних поверхнях. Ще однією її місією є те, що за допомогою сфери ми вимірюємо відстань, яка необхідна для того, щоб павук зробив крок. Тобто, якщо відстань від нижньої частини лапи до сфери стає більше за певне значення, павук зробить крок, тобто перемістить лапу до сфери.

У зв'язку з тим, що в нашого персонажа всього чотири лапи, ми повторимо всі описані вище кроки чотири рази, при цьому виставивши необхідні значення для кожної ноги.

Важливо згадати, що якщо павук йде по нерівним поверхням, наприклад така, яка описана на рисунку 5.4, то важливо тримати фіксованою відстань від тіла павука до землі, щоб він не «втонув» в текстурах або не літав над ними.

Додамо до павука скрипт *SpiderController*, за допомогою якого зможемо рухати павука. Скомбінувавши всі описані вище кроки, отримаємо в результаті наступну поведінку, показану на рис 5.5:

Рисунок 5.5 – Процедурна анімація руху

Якщо звернути увагу на той момент, коли павук зупиняється, то можна побачити, що перед зупинкою йдуть певні коливання. Це результат застосування системи другого порядку у скрипті *SecondOrderSystem*, який було описано вище.

6 ІНТЕГРАЦІЯ В РЕАЛЬНУ ГРУ ТА ОЦІНКА ЕФЕКТИВНОСТІ

Після виконання всіх попередніх кроків, що необхідні для створення реалістичної процедурної анімації для руху, стало можливим запровадити здобуті нами результати до реальної гри.

Як можна переконатись, рушій Unity надає багато зручних технологій для створення можливих рішень цієї проблеми. Налаштувавши параметри системи на свій розсуд, використавши цікаві моделі для навколишнього середовища, виставивши світло, додавши цікаві особливості, такі як туман, та продумавши свій концепт гри, можна створити якісний та цікавий продукт, який зацікавить гравців.

Щодо ефективності, то після вимірювань кадрів на секунду на моєму апараті, отримав результат – 300 кадрів за секунду, що є дуже гарним результатом. Це свідчить про те, що розроблена нами система є ефективна та не перешкоджає швидкості та продуктивності. Звичайно, варто пам’ятати, що чим комплексніша гра та чим більше об’єктів знаходиться на сцені, тим менше буде значення кадрів за секунду.

ВИСНОВКИ

Отже, тривимірна графіка та анімація – одні з найпопулярніших та найважливіших галузей сьогодення. Одна зі сфер їхнього застосування – комп’ютерні ігри. У цій роботі було розроблено гру за допомогою застосування 3D графіки та процедурної анімації. Після аналізу здобутих результатів можна зробити ряд висновків:

Оцінка одержаних результатів та їх відповідність сучасному рівню наукових і технічних знань і технологій. На мою думку, отримані результати є досить важливими та цікавими в області процедурної анімації. Також, описані мною теоретичні основи є фундаментальними та відповідають сучасному рівню знань і технологій.

Ступінь впровадження та можливі галузі або сфери використання результатів роботи. Я вважаю, що дану роботу можна використовувати як допоміжну у сфері комп’ютерних ігор. Надане математичне підгрунтя та опис алгоритмів для процедурної анімації можуть бути корисними, якщо перед розробником стоїть ціль створення процедурної анімації для руху. Також, на мою думку, описані теоретичні основи 3D графіки та анімації можуть допомогти людям, які зацікавлені в їх освоєнні.

Інформація щодо створення нового програмного продукту. У ході розробки було створено комп’ютерну гру, основною ціллю якої стало впровадження процедурної анімації руху.

Науково-технічна значущість роботи. Розробці гри передувало описання математичного підгрунтя для створення процедурної анімації для руху. На мою думку, значущістю даної роботи є те, що вона довела, що математику та фізику можна використовувати в найнесподіваніших, проте досить цікавих ситуаціях.

Доцільність продовження досліджень або розробок за відповідною тематикою. Описані в цій роботі основи – лише вершина айсберга. Це

стосується і теоретичних основ 3D графіки та анімації, і математичного апарату для процедурної анімації. На мою думку, подальше дослідження цікавих концепцій процедурної анімації дозволить розробникам створювати системи, що привнесуть реалізму та нових відчутів усім любителям комп’ютерних ігор.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

Книги: один автор

1. Eric L. Mathematics for 3D Game Programming and Computer Graphics / Lengyel Eric. – Boston: Third Edition, 2012. – 566 с.
2. Peter C. Mathematical and Computer Programming Techniques for Computer Graphics / Comninos Peter. – London, 2006. – 531 с. – (Springer).

Електронні ресурси

3. Вплив анімації та графіки на життя. [Електронний ресурс] – Режим доступу до ресурсу: https://www.explanimate.com.au/how-does-animation-affect-our-daily-life/.
4. Історія 3D графіки [Електронний ресурс] – Режим доступу до ресурсу: https://www.3dhorse.com/blogs/3d/history-of-3d-computer-graphics.
5. Полігональні моделі [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/computer-science/polygonal-model.
6. 3D моделювання та його застосування [Електронний ресурс] – Режим доступу до ресурсу: https://conceptartempire.com/what-is-3d-modeling/.
7. Методи моделювання [Електронний ресурс] – Режим доступу до ресурсу: https://artisticrender.com/10-different-types-of-3d-modeling-techniques/.
8. Загальний опис 3D моделювання [Електронний ресурс] – Режим доступу до ресурсу: https://dreamfarmstudios.com/blog/a-quick-guide-to-3d-modeling/.
9. Генерація сіток(meshes) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Mesh_generation.

10. 3D текстурування та його методи [Електронний ресурс] – Режим доступу до ресурсу: https://www.adobe.com/ua/products/substance3d/discover/3d-texturing.html.
11. UV-mapping and unwrapping [Електронний ресурс] – Режим доступу до ресурсу: https://conceptartempire.com/uv-mapping-unwrapping/.
12. Текстурні карти [Електронний ресурс] – Режим доступу до ресурсу: https://conceptartempire.com/texture-maps/.
13. Формати для зберігання 3D моделей [Електронний ресурс] – Режим доступу до ресурсу: https://www.vntana.com/blog/demystifying-3d-file-formats-for-3d-commerce-and-more/.
14. Визначення рендерингу [Електронний ресурс] – Режим доступу до ресурсу: https://www.techopedia.com/definition/9163/rendering.
15. Алгоритми рендерингу: трасування променів та растеризація [Електронний ресурс] – Режим доступу до ресурсу: https://blogs.nvidia.com/blog/2018/03/19/whats-difference-between-ray-tracing-rasterization/.
16. Визначення комп'ютерної анімації [Електронний ресурс] – Режим доступу до ресурсу: https://www.britannica.com/topic/computer-animation.
17. Застосування інтерполяції [Електронний ресурс] – Режим доступу до ресурсу: https://businessofanimation.com/a-thorough-guide-to-interpolation-animation-for-animators/.
18. Комп'ютерна анімація та її види [Електронний ресурс] – Режим доступу до ресурсу: https://unity.com/solutions/computer-animation-explained.
19. Методи комп'ютерної анімації [Електронний ресурс] – Режим доступу до ресурсу:

https://www.tutorialspoint.com/computer_graphics/computer_animation.htm.

20. Диференціальні рівняння другого порядку [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/engineering/second-order-system.

21. Системи другого порядку [Електронний ресурс] // MIT – Режим доступу до ресурсу: https://ocw.mit.edu/courses/2-003-modeling-dynamics-and-control-i-spring-2005/57d44d83366ec969c16208c8fac3982d_notesinstalment2.pdf.

22. Коефіцієнт затухання [Електронний ресурс] – Режим доступу до ресурсу: https://encyclopedia.pub/entry/29032.

23. Власна частота [Електронний ресурс] – Режим доступу до ресурсу: https://www.simscale.com/docs/simwiki/fea-finite-element-analysis/what-is-natural-frequency/.

ДОДАТОК А

Клас *SecondOrderSystem* – клас що реалізує систему другого порядку та її розв'язання за допомогою напівнеявного методу Ейлера.

Код цього класу:

```
public class SecondOrderDynamics
{
    private Vector3 xp;
    private Vector3 y, yd;
    private float k1, k2, k3;

    public SecondOrderDynamics(float f, float z, float r, Vector3 x0)
    {
        k1 = z / (f * Mathf.PI);
        k2 = 1 / ((2 * Mathf.PI * f) * (2 * Mathf.PI * f));
        k3 = (r * z) / (2 * Mathf.PI * f);

        xp = x0;
        y = x0;
        yd = Vector3.zero;
    }
    public Vector3 UpdateValues(float T, Vector3 x, Vector3? xd = null)
    {
        if (xd == null)
        {
            xd = (x - xp) / T;
            //Debug.Log(xd + " x derivative");
            xp = x;
        }
        y = y + T * yd;
        yd = yd + T * (x + k3 * xd.Value - y - k1 * yd) / k2;
        //Debug.Log("Y derivative is " + yd);
        return y;
    }

}
```

Основна різниця в імплементації напівнеявного методу Ейлера та звичайного методу Ейлера полягає в тому, що для обрахування *yd* ми використовуємо наступне значення *y*. Тим часом як в звичайному методы використовується попереднє значення *y*.