МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Інформаційна технологія розпізнавання об'єктів. Бортова система безпілотного авіаційного комплексу для розпізнавання природних та інфраструктурних об'єктів»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Довбиш А.С.**

Студент гр. ІН-73 **Гриненко О.В.**

Суми 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп’ютерних наук

Затверджую_____
Зав. кафедрою Довбиш А.С.
“_____” _____2021 р.

ЗАВДАННЯ

до випускної роботи

Студента четвертого курсу, групи ІН-73 спеціальності “Комп'ютерні науки” денної форми навчання Гриненко О.В.

Тема: “**Інформаційна технологія розпізнавання об’єктів. Бортова система безпілотного авіаційного комплексу для розпізнавання природних та інфраструктурних об’єктів**”

Затверджена наказом по СумДУ
№_____ від _____2021 р.

Зміст пояснювальної записки: 1) Інформаційний огляд; 2) Вибір методу розв’язання задачі; 3) Інформаційне і програмне забезпечення.

Дата видачі завдання “_____”_____2021 р.
Керівник випускної роботи _____ Довбиш А.С.
Завдання прийняв до виконання _____ Гриненко О.В.

РЕФЕРАТ

Записка: 69 стор., 66 рис., 1 табл., 1 додаток, 14 джерел.

Об’єкт дослідження – вхідний математичний опис для безпілотного літального апарату.

Мета роботи — розробка наближеної до реальності 3D моделі ландшафту з об’єктами людської діяльності та розробка модулю управління дроном.

Методи дослідження — методи створення вхідного математичного опису для безпілотного літального апарату (реалізація віртуального ландшафту, об'єктів та природних насаджень). Реалізація модулів управління дроном методами візуального програмування.

Результати — Проведено аналіз інструментів реалізації ландшафту та об’єктів, після чого обрано оптимальні варіанти. За допомогою Unreal Engine 4 розроблено ландшафт наближений до реальності з автопідбором текстур в залежності від куту нахилу поверхні. За допомогою Blender створено об’єкти сільської місцевості (будинки, стадіони, ларьки, тощо.) та дороги. Також досліджені новітні моделі квадрокоптерів та створено цифрову копію однієї з них. Розроблено модуль керування квадрокоптером, наближений до реальності.

ЗМІСТ

ВСТУП

Геоінформаційна Система (ГіС) – це новітні технології для нотування та планування карт місцевості, які несуть в собі певну інформацію про конкретну ділянку та об’єкти на ній. Система може включати в себе бази даних, редактори зображень (растрові, векторні) або аналітичні інструменти для аналізу поверхні [1].

Розвиток геоінформаційних систем почався разом із заснуванням інформаційних систем у другій половині 20-го сторіччя. В 70-х роках Канадське керівництво вирішило застосовувати ГіС для кадастрового опису, який збирали і форматували у вид карти. В 21-му сторіччі ГіС має багато застосувань у різних куточках світу та з різними цілями [1]. Але не завжди для випробування дронів є потрібні умови або ресурси.

Проблему з витратою зайвих ресурсів або формуванням потрібних кондицій можливо вирішити за допомогою програмного забезпечення для 3D моделювання ландшафту, на якому є будинки, дорога, гори, ліси, тощо. Тобто, це імітація реальної місцевості. Працюючи в цифровому симуляторі можна моделювати фізичні процеси (дощ, вибухи, рух автомобілю), які будуть відбуватись на створеній поверхні. Для імітацій потрібен сам ландшафт.

Вищеописана технологія може бути застосована в громадському середовищі як, наприклад, викривач вирубки лісів, відстежування засихання озер, розлив нафти в океані, тощо. У військовому середовищі ідентифікація місцевості наразі є основою ведення бойових дій та розвідки.

В роботі розглядається задача розробки інформаційного та програмного забезпечення системи інтелектуального аналізу геоінформаційних даних, що були отримані за допомогою безпілотних літальних апаратів.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Сучасні технології формування та обробки геоінформаційних даних

Геоінформаційна система – це набір геоінформаційних даних, які об’єднаний в систему за певною логікою та потрібним функціоналом, який дає змогу виконувати різні маніпуляції з геоданими, наприклад, створювати списки, прокладати маршрути, відмічати потрібні місця, створювати опис для кожної точки, позначати область дій, тощо [1].

Прикладом може стати звичайна карта України, де нанесено відповідні позначки міст, селищ міського типу, тощо. Із більш складних прикладів можна представити платформу Google Maps, яка має різноманітну інформацію про вулиці, номери будинків, назви закладів (та обговорення з коментарями), інформацію про автомобільні дороги, інформація про велодоріжки і так далі.

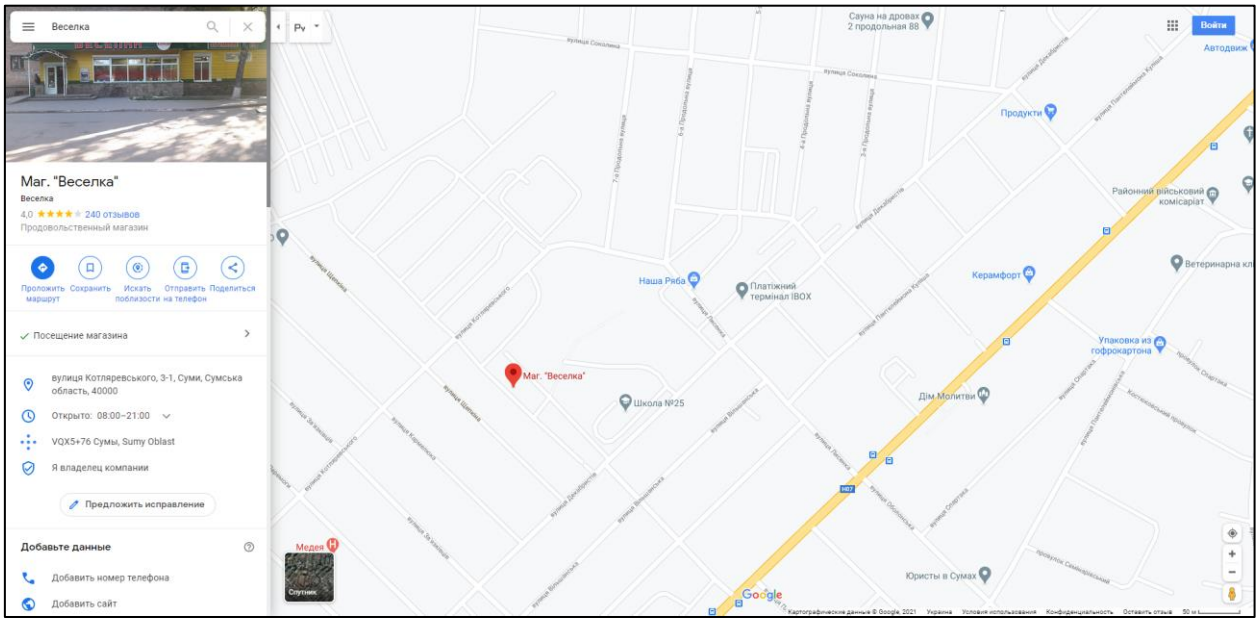


Рисунок 1.1 – Приклад роботи Google Maps

Кажучи про типи даних в системах треба зазначити, що всі операції проводяться на стартовій карті [1]. Це зображення або їх компіляція, на якій нанесено певні позначки для базової орієнтації на місцевості. Оперуючи даними, ГіС поділяє їх на два види:

- Атрибутивний – це вид даних в геоінформаційній системі для опису певних зон. Даний тип є гнучким по відношенню до користувача, адже його структуру визначає саме користувач. Наприклад: нанесення певної характеристики на один із районів міста, який представлений у ГіС, пунктирною лінією [2].

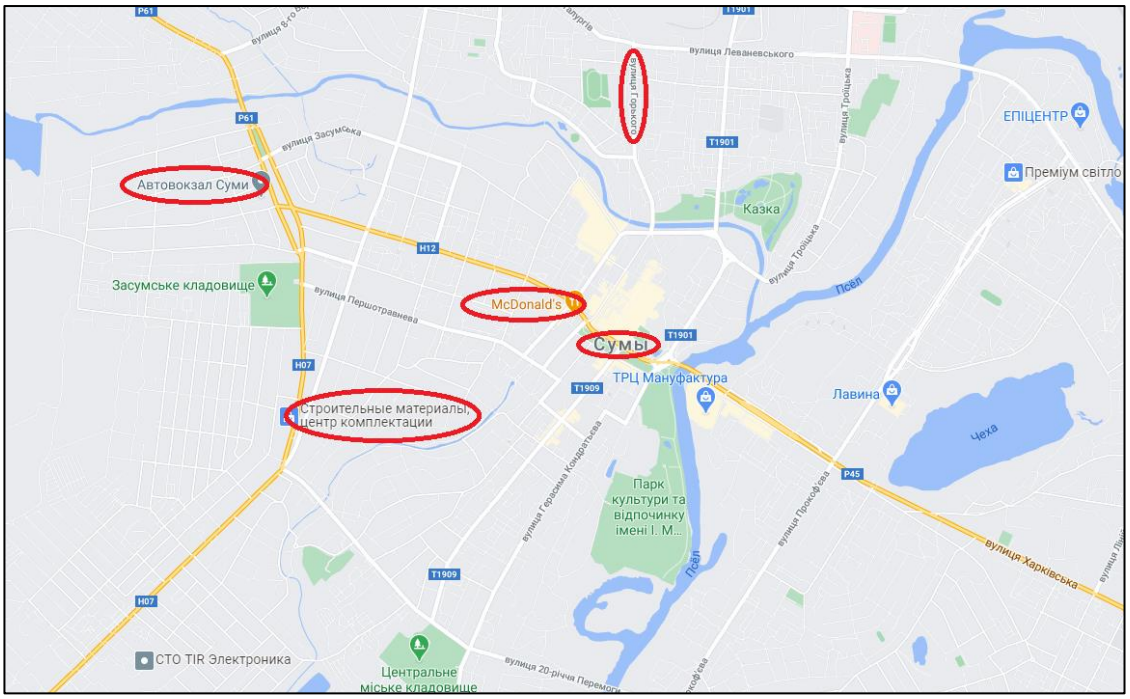
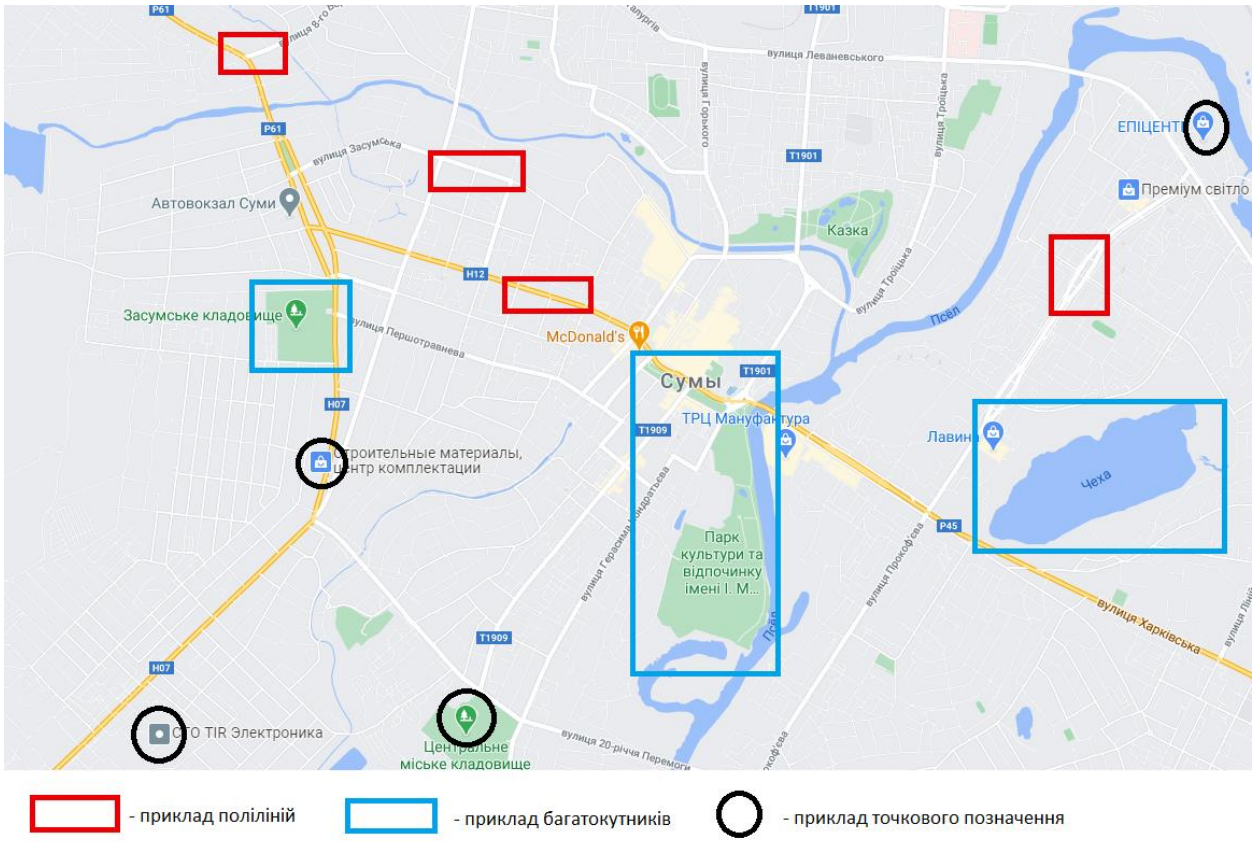


Рисунок 1.2 – Атрибутивні написи на прикладі Google Maps

- Позиційний (просторовий) – це вид даних, який відповідає за позначення місцеположення доріг, будинків, лісів, боліт, тощо. Також позиційні дані передають постійні дані, наприклад опади (рівень опадів вказується навіть якщо опадів не було), рельєф і тому подібне [1]. Представити просторовий тип даних в ГіС можливо двома способами:

Векторним способом. Тобто за допомогою геометричних простих фігур, підвидів фігур три: точки, за допомогою яких позначають місцеположення, а не форму об’єкта. На карті світу таку позначку мають міста, а в межі міста такі позначки притаманні різним будівлям, паркам, тощо; багатокутники використовують для позначення меж міста на карті країни або для позначки меж країн на карті світу, тобто застосовується якщо треба позначити чіткі контури; для зображення лінійних об’єктів (дорога, велодоріжка, залізниця, тощо.)

використовують так звані полілінії, які залежать від масштабу карти, тобто дорога регіонального масштабу буде невидима на карті країни або світу [1];



- приклад поліліній - приклад багатокутників - приклад точкового позначення

Рисунок 1.3 –Приклад векторного подання геоданих

Растровий спосіб полягає у встановленні міри для певних об’єктів. Це прямокутна шкала поділена на потрібну кількість частин. Кожна частина шкали називається пікселями. Растрові дані зберігаються у типових форматах для зображень: TIFF там JPEG. Прикладом можна назвати карту середніх температур вод океанів та мірною сіткою [1].

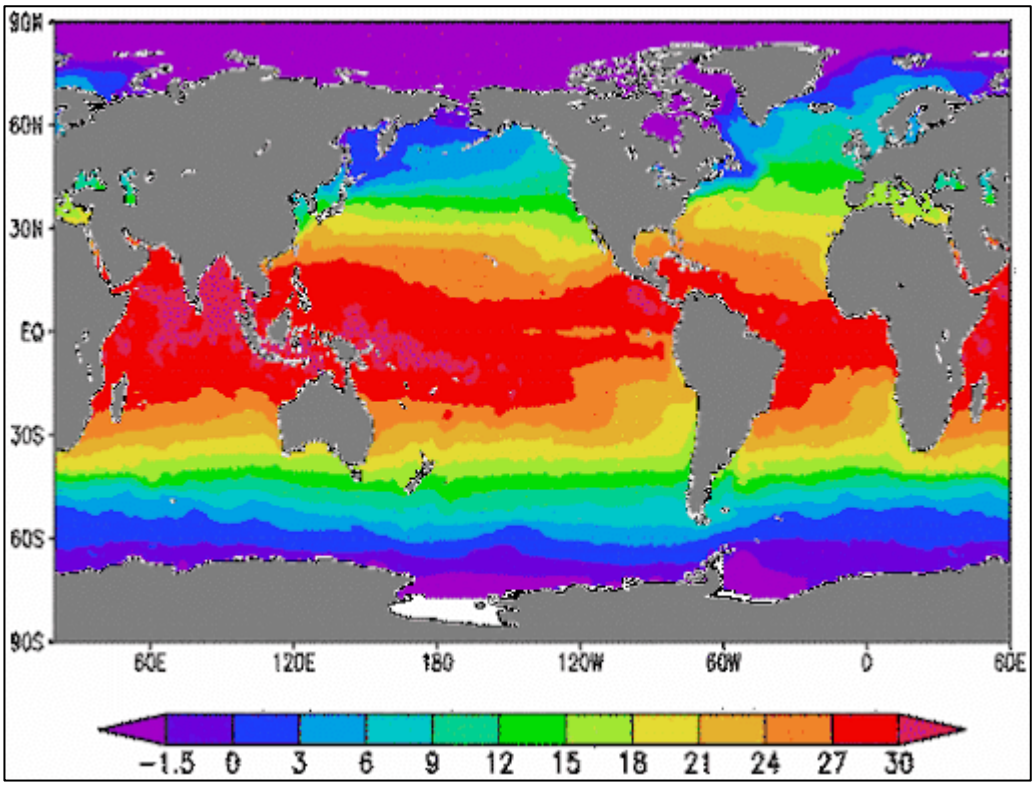


Рисунок 1.4 – Приклад нанесених растрових шкал

Користуються ГіС різні люди: від спеціалістів свого діла до звичайних громадян. Прикладом громадянських ГіС є Гугл Карти, які мають широкий спектр використання. Ця ГіС включає в себе не тільки навігаційні можливості. Вона включає в себе систему рейтингів закладів, розташованих на карті, які можна віднайти вручну або скористатись пошуком, в якому, до речі, можна шукати конкретні вулиці або конкретні будівлі через їх назву. Також Гугл Карти мають кілька слоїв відображення геоінформації: першим шаром є намальована карта за допомогою векторів (не мають пікселізації при приближенні мапи); другим шаром є знімки з супутника, які оновлюються кожного місяця.

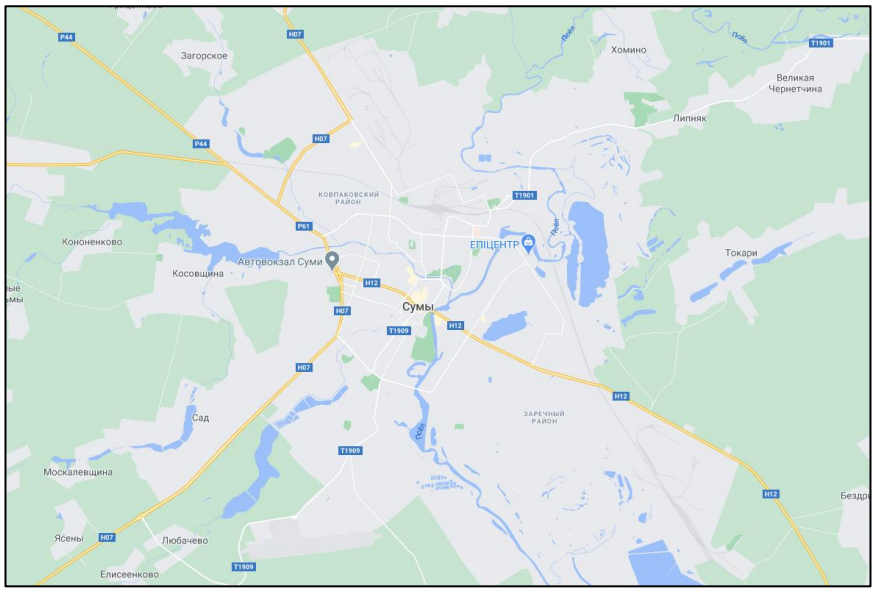


Рисунок 1.5 – Загальний вид Гугл Мапс у векторному

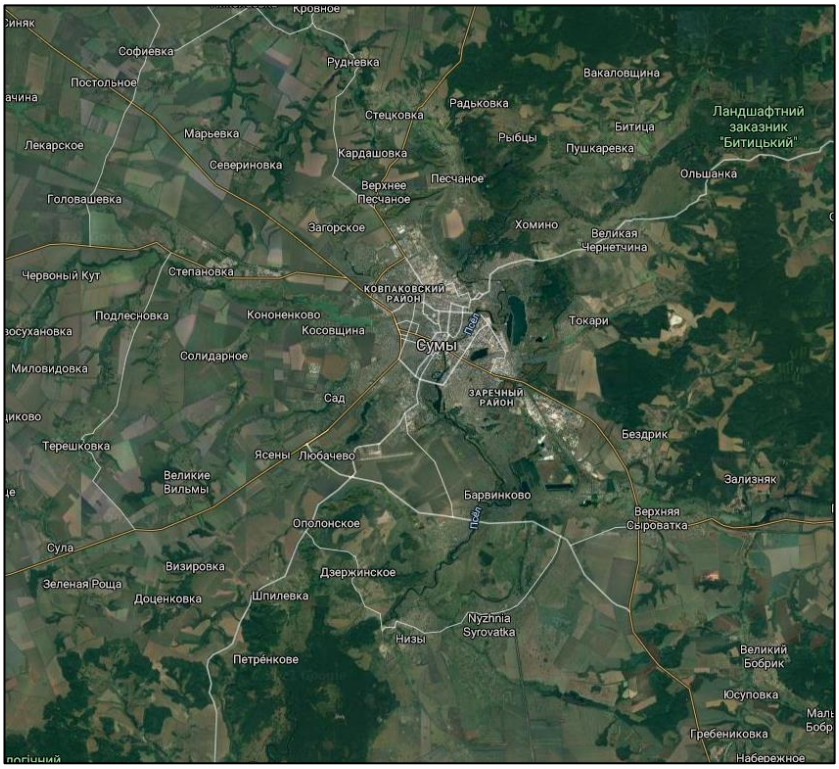


Рисунок 1.6 – Вид «з супутника»

Правоохоронні органи не так давно почали активно використовувати геоінформаційні системи для зручного позначення місць злочинів, їх розслідування та приміток. Ця ГіС відноситься до більш вузьконаправлених. Раніше МВС користувались простою геосистемою під назвою «Карта на стіні з

позначеннями маркером або шпильками». Наразі це програма в комп'ютері зі зручним інтерфейсом та можливістю видаляти і збирати інформацію.

Система Гугл Мап здається складною, але є ще складніша ГіС, яка частинно є в функціоналі вищеописаної геосистеми, але в повному обсязі присутня в додатку Гугл Планета. Вона відрізняється від Гугл Мап своєю відокремленістю від браузера (тому, що Гугл Мап – це компактна та легкодоступна ГіС для швидкого використання) та, логічно, додатковим функціоналом. В даній системі є трьохвимірна проекція міст, гір і інших об'єктів оточення, які можна досліджувати.

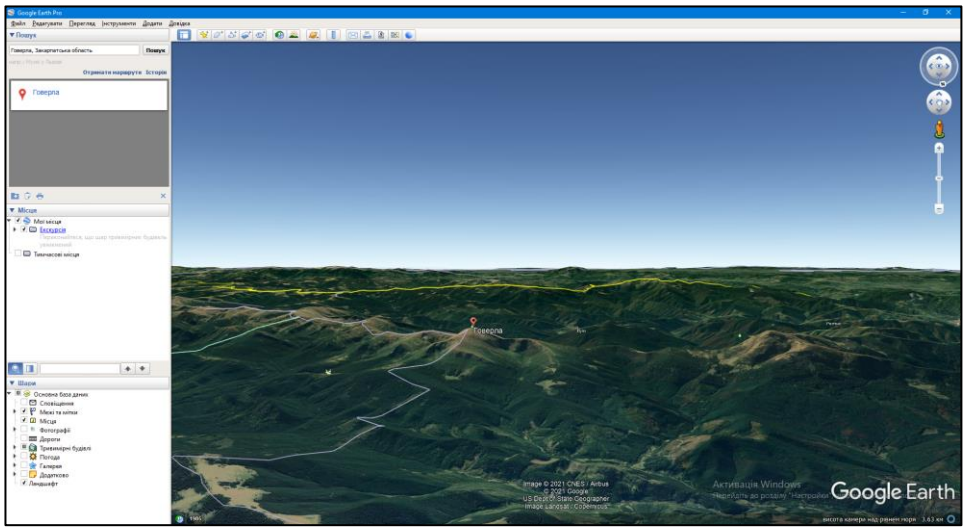


Рисунок 1.7 – Приклад показу висоти гори Говерла

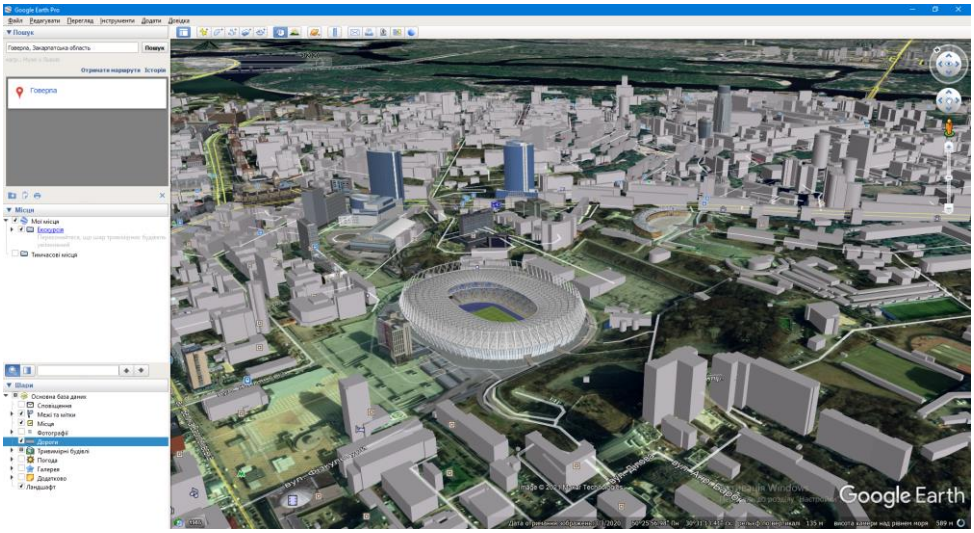


Рисунок 1.8 – Тривимірні будівлі згенеровані Гугл Планетою

Можна помітити що кожна будівля має свою форму, а стадіон Олімпійський унікальний в своєму роді. Зроблено висновок, що для створення подібного на рисунку ландшафту потрібні 3D об’єкти. Кожен із них є унікальним та неповторним, тому об’єкти належить змоделювати та побудувати у програмах призначених для цього.

1.2 Створення об’єктів та вибір програм реалізації

На етапі планування власної геоінформаційної системи був зроблений висновок про те, що об’єкти для 3D оточення потрібно робити самостійно, в одній із призначених для цього програм. Створені об’єкти мають бути якісними і простими водночас, не мати фотореалістичних текстур, великої кількості полігонів і тому подібне задля оптимізації карти в цілому. Враховуючи всі вимоги до моделювання, є вибір серед декількох програм моделювання об’єктів видів:

SolidWorks – це комплекс систем автоматизованого проектування, який забезпечує розробку деталей, об’єктів різного розміру та рівнів складності. Даний продукт SolidWorks Corp. представляє собою серйозне програмне забезпечення для проектування складних об’єктів (наприклад: шестерні, при проектуванні яких немає права на погрішність +- 1мм), інженерних споруд, ланцюгів, дротів, тощо [3].

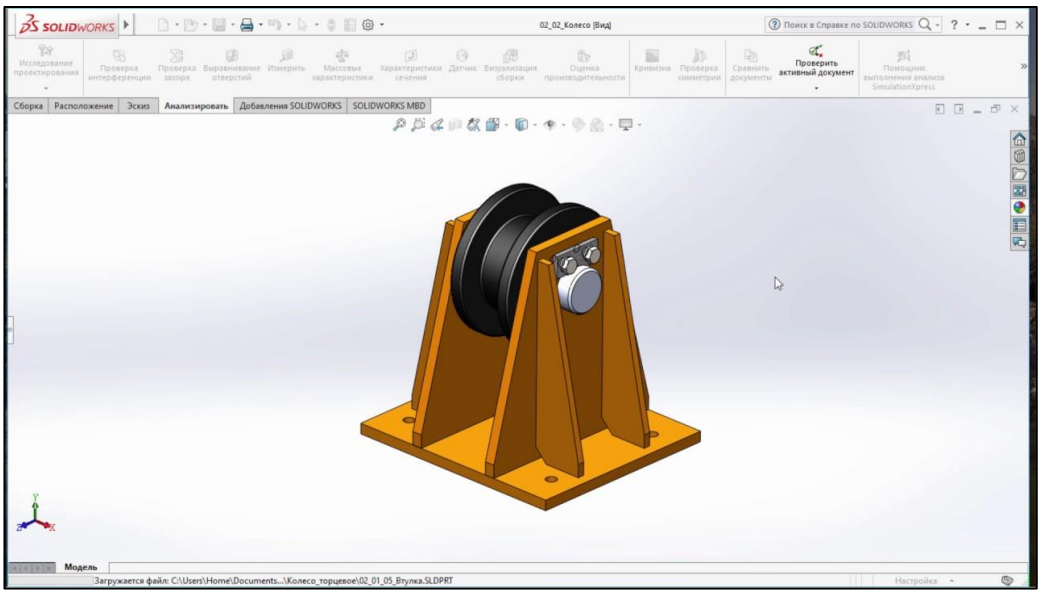


Рисунок 1.9 – Вікно програми SolidWorks

Головний мінус – це ціна. Є випробувальний період, але через 30 днів треба буде заплатити за ліцензію. Також вимоги до гарної деталізації немає. Підсумовуючи всі плюси і мінуси зроблено висновок, що SolidWorks не підходить для створення об’єктів ГіС.

3DS Max – це продукт компанії Autodesk для моделювання та анімації. Ця програма дає можливість проектувати та будувати будинки та інші об’єкти, не так детально як в SolidWorks. Дане програмне забезпечення має потужні інструменти рендерингу сцен, фотореалістичні способи рендерингу, та ін [4]. Також хочеться підмітити, що безоплатної ліцензованої копії немає, тільки купувати підписку[5]. Враховуючи не безоплатність програмного забезпечення, його призначення та великий обсяг потрібний на жорсткому диску, можемо зробити висновок, що дане програмне забезпечення не підходить для створення ГіС.

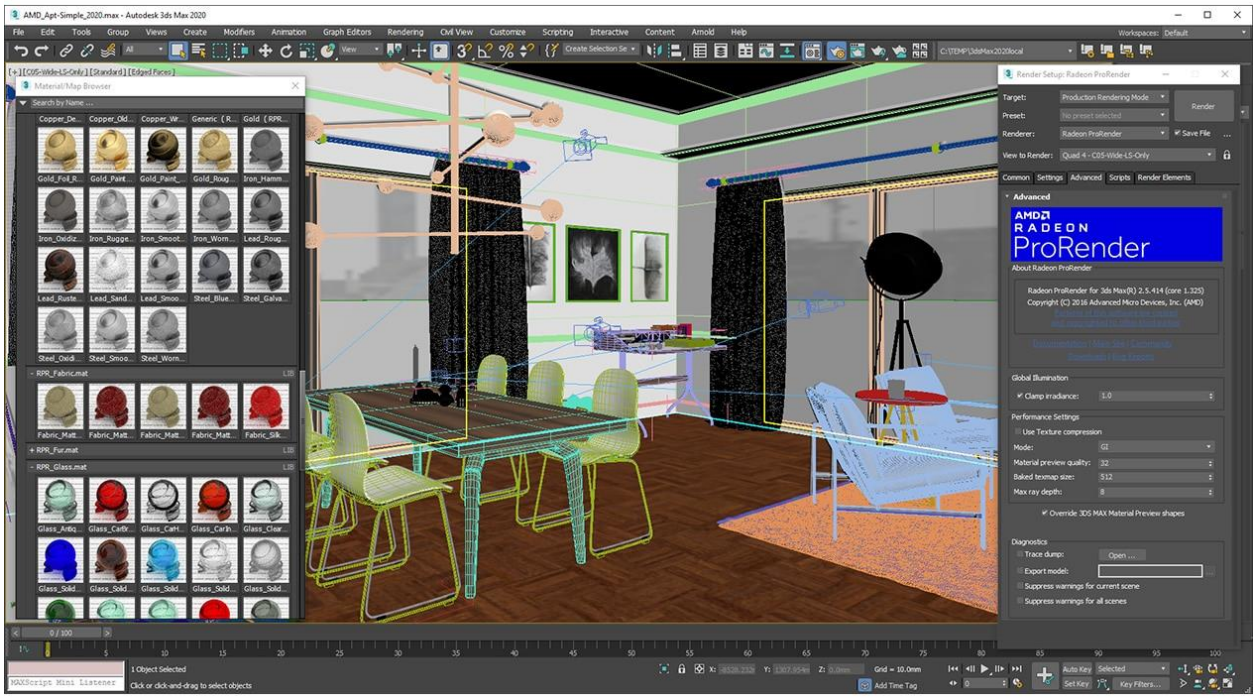


Рисунок 1.10 – Вікно програми 3DS Max

Blender – це продукт для створення 3D об’єктів, анімацій. За розробку відповідає компанія Blender Foundation та Том Розендал. Головним плюсом є безоплатність продукту, швидка технічна підтримка та стрімкий розвиток

Blender [6]. Також хочеться відмітити простий інтерфейс та відносно невеликий розмір самої програми[7]. По більшості параметрів підходить для створення об’єктів в ГіС.

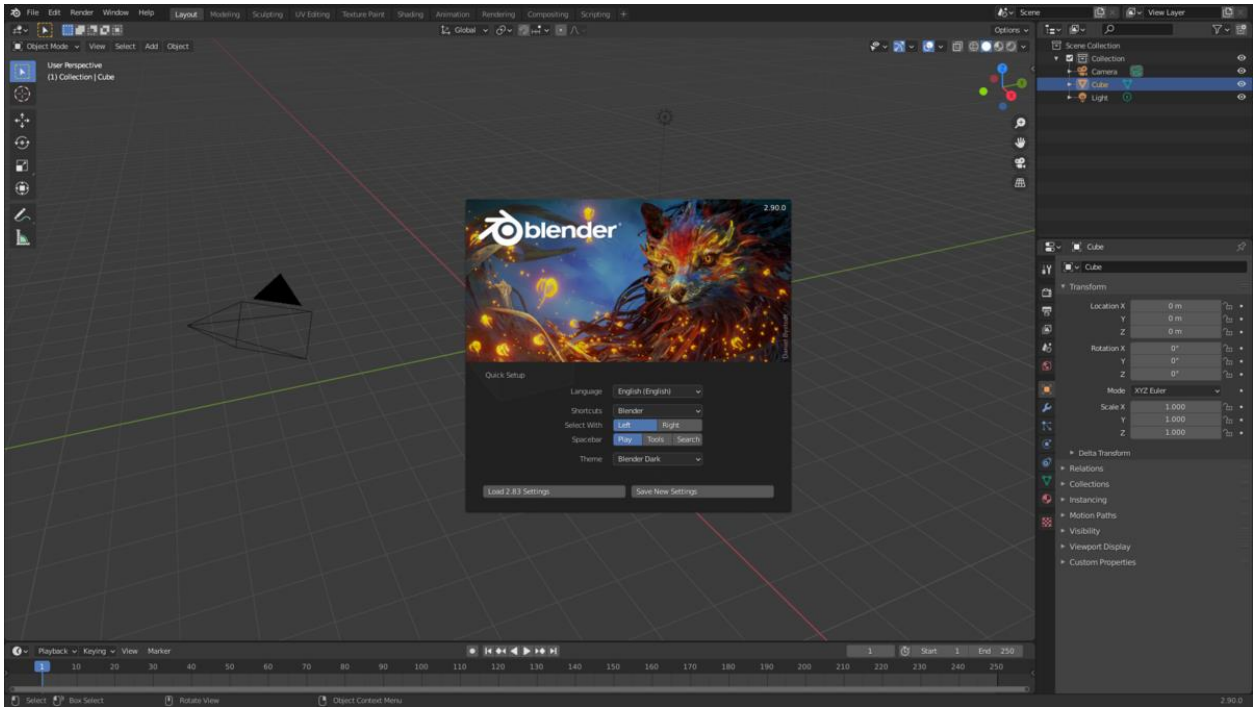


Рисунок 1.11 – Стартове вікно Blender

Отже, на даний момент обрано програму для створення об’єктів в 3D геоінформаційну систему, на котрій неможливо розроблювати логіку дрону. Тому, тепер треба обрати платформу, на якій можливе просте створення ландшафту, реалізації логіки дрону та подальше розміщення створених будівель, лісів, тощо. Головною вимогою є адаптивність до файлів імпорту з Blender, тобто розширення fbx. Вибір належить зробити серед двох великих платформ:

Unity – це міжплатформенне середовище для розробки компьютерних ігр, створене в далекому 2009-му році. До переваг можемо віднести безкоштовне використання ліцензії до певного прибутку від продукту, який розроблявся за допомогою технологій Юніті. Також хочеться відмітити наявність простого інтерфейсу. Окрім «плюсів», є «мінуси» такі як: відсутність налаштувань мультиплеєру, який в випадку з Юніті треба дозавантажувати з сторонніх сайтів [8]. Також хотілося б відмітити відсутність візуального програмування, яке

дозволяє не тільки писати код, але й бачити ієрархію, функції і тому подібне в простому табличному виді.

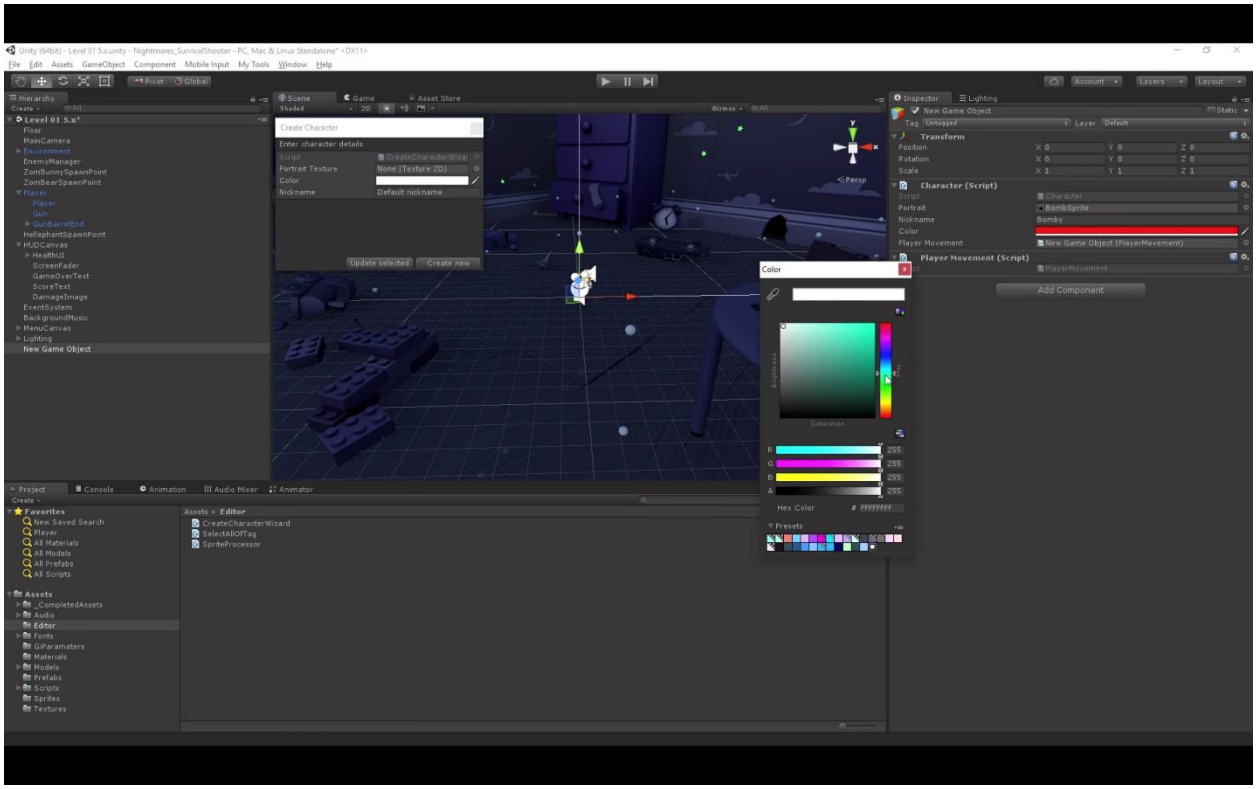


Рисунок 1.12 – Інтерфейс Unity

Unreal Engine – це розробка компанії Epic Games, яка була створена в 1998 році та стала основою для першої гри цієї компанії. Згодом Epic Games виклала в вільний доступ частину виконуючих кодів гри, після чого користувачі почали розробку модифікацій для ігрового рушія. На відміну від Unity, Unreal Engine має серверно-клієнтську взаємодію передустановленою функцією. Також із плюсів Epic Games для даного рушія надає велику базу безоплатних та платних готових об’єктів, яка є фактично магазином створених людьми об’єктів. З 2015 року Unreal Engine є безкоштовним, роялті у виді 5% від доходу треба платити якщо він перевищує 3000 доларів за квартал. Одним із важливих плюсів є можливість візуального програмування [9].

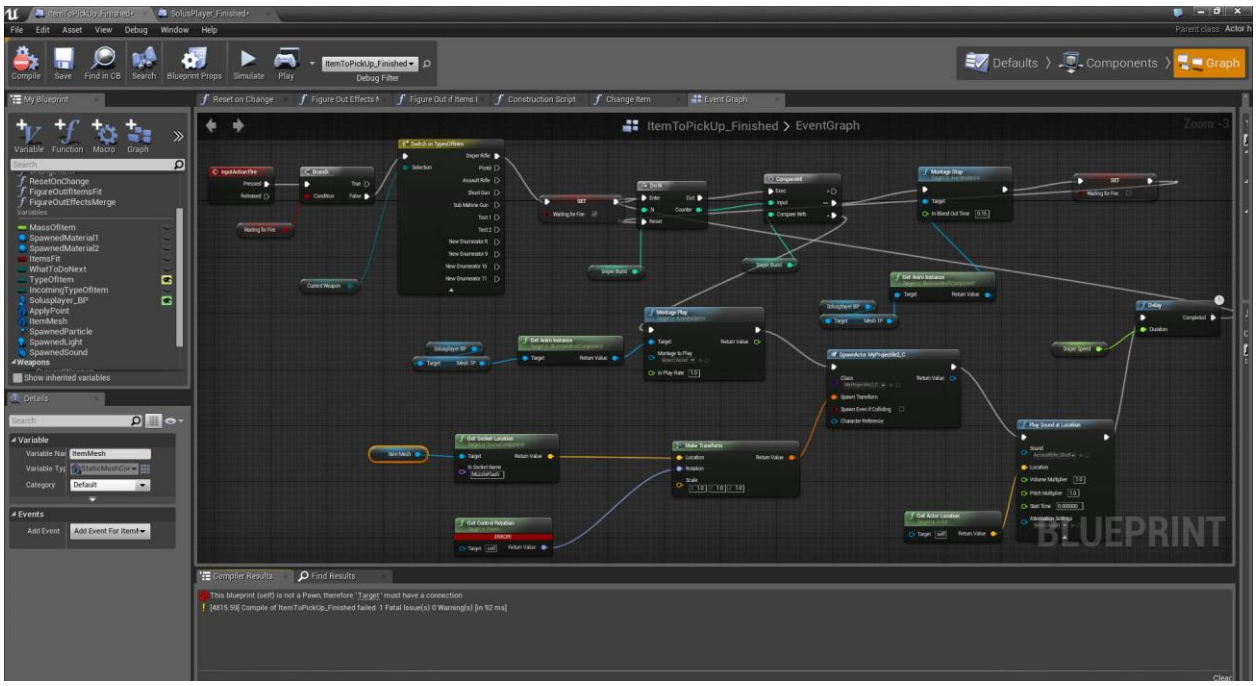


Рисунок 1.13 – Вікно налаштувань об’єкта, приклад візуального програмування

Прочитавши та зважившись на всі недоліки та переваги обох ігрових рушіїв, можна зробити висновок, що більш за все для проекту ГіС підходить Unreal Engine. Завдяки своїм можливостям можна налаштувати клієнтсько-серверну взаємодію та програмувати як візуально, так і у звичному способі (збільшити розуміння логіки та коду).

В проекті є потреба в малюванні текстур самостійно, отже треба обрати редактор зображень. За весь час комп’ютерних наук розроблено багато програм-редакторів, але обрано Photoshop від компанії Adobe. Він надає можливість швидко та ефективно редагувати фотографії, малюнки і тому подібне. Має простий інтерфейс, багато матеріалів для навчання.

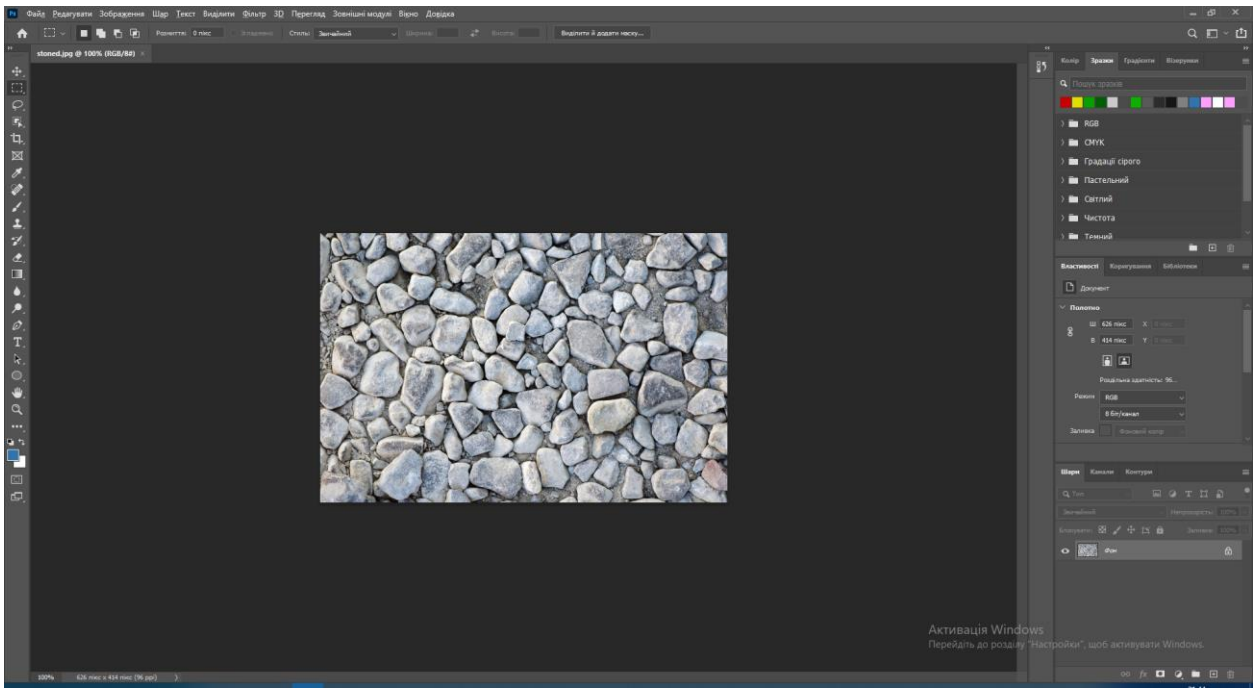


Рисунок 1.14 – Вікно Adobe Photoshop

Таким чином, в нас обрано програми для реалізації ГіС та об’єктів для неї, після чого поділимо їх на декілька категорій:

1. Споруди, такі як заправки, домівки, продовольчі магазини, тощо.
2. Декоративні об’єкти які не мають особливого призначення.
3. Інфраструктурні об’єкти, наприклад: дороги будь якого виду, пішохідні доріжки та все що з ними зв’язано.
4. Натуральні насадження

Заплановане одне поселення, лісові насадження, пагорби та центральна дорога, яка йде через всю карту.

1.3 Постановка задачі

Задачею є створення 3D моделі місцевості з об’єктами на ній. Об’єкти змоделювати самостійно. Реалізувати керування дроном та його 3D модель.

1) Створити 3д-об’єкти, які будуть використанні при моделюванні місцевості у рамках геоінформаційної системи.
2) Розробити 3д-макет, на якому будуть знаходитися різноманітні геоінформаційні об’єкти, які характерні для обраної території.

3) Створити трьох вимірну модель обраного безпілотного літального апарату (БПЛА).
4) Програмно реалізувати модуль керування БПЛА.

2 ВИБІР МЕТОДУ РОЗВ’ЯЗАННЯ ЗАДАЧІ

2.1 Моделювання поверхні місцевості (ландшафт)

Ландшафт створимо за допомогою Unreal Engine, попередньо створивши його план.

Рисунок 2.1 – План розміщення об’єктів на майбутньому ландшафті

Сама поверхня створена за допомогою інструменту Landscape, після чого створиться плоска планка зі стандартним покриттям у виді трави. Але такий її вид не підходить, адже вона занадто яскрава. Змінено текстуру та почато налаштування текстурування в залежності від нахилу поверхні.

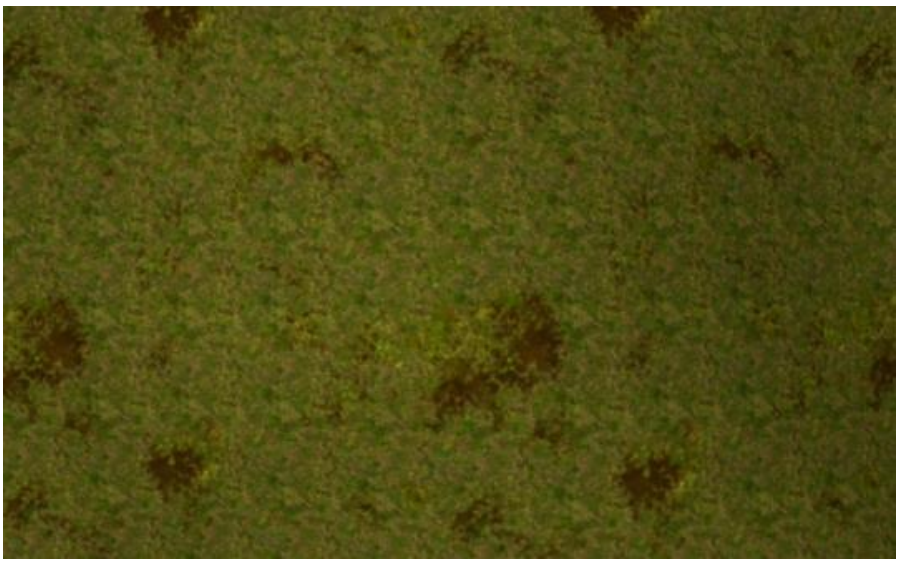


Рисунок 2.2 – Нова текстура землі

Приступимо до функції автоматичного текстурування поверхні. Це потрібно для спрощення та автоматичного змінення текстур, тому, що їх потрібно було б малювати вручну, або створювати її заздалегідь. Реалізація зміни текстури залежно від нахилу буде реалізовано за допомогою візуального програмування (нод). Було створено дві функції матеріалів, щоб мінімізувати загальний вид. Функція матеріалу скелі виглядає так:

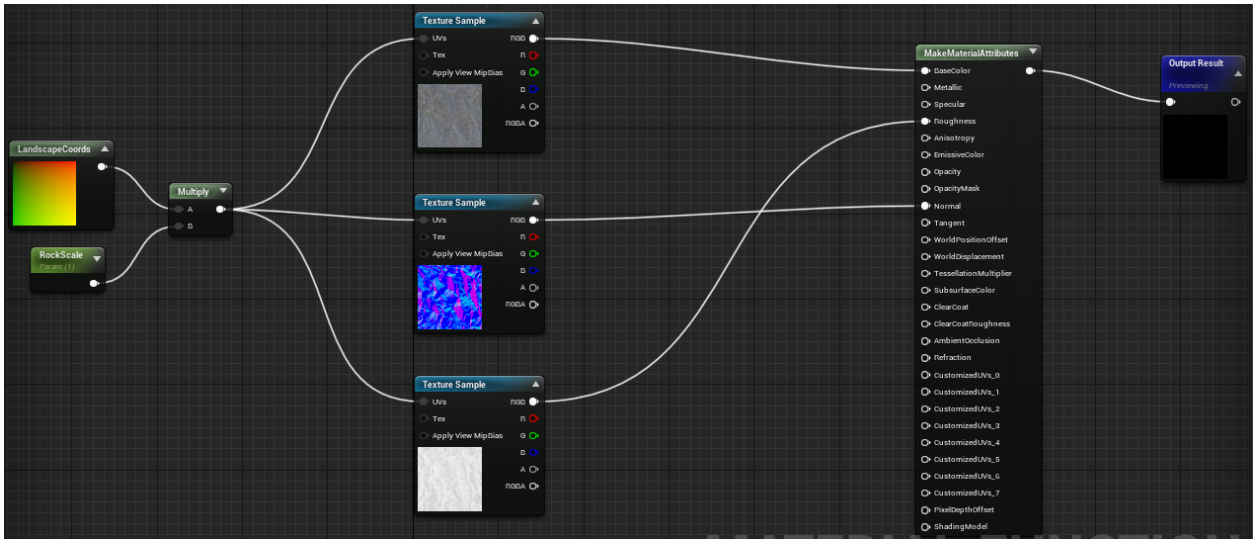


Рисунок 2.3 – Матеріал «Скеля»

На рисунку видно різні представлення самої текстури: стандартна текстура, карта нормалей (для показу об’ємності) та карта висот (для реалістичної візуальної деформації). Всі параметри підключаємо до відповідних

параметрів у MakeMaterialAttr. В даній композиції присутній параметр RockScale, який відповідає за масштаб скелі на загальному виді. Аналогічно створюємо функцію матеріалу землі, замінюючи текстури каменю на трав’яну поверхню. Параметр змінюється з RockScale на GrassScale.

Рисунок 2.4 – Текстура землі

Створивши дві функції матеріалів, треба перейти до об’єднання в систему автотекстурування.

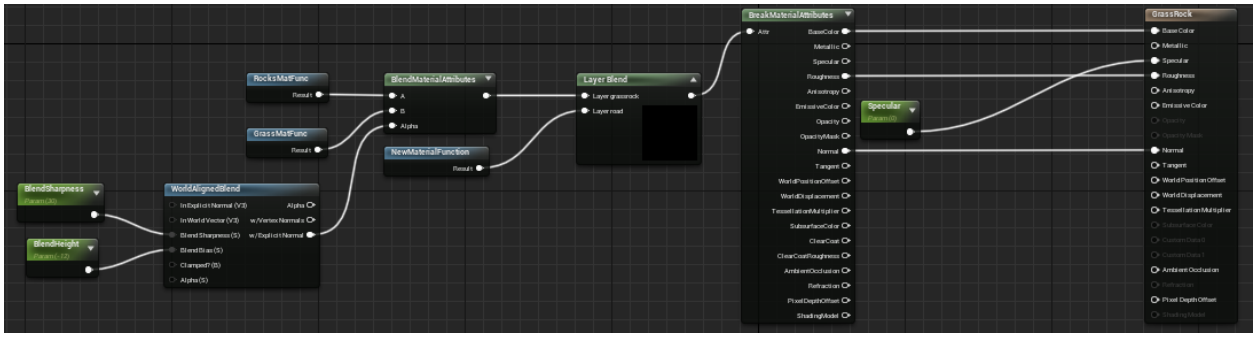


Рисунок 2.5 – Ноди для автопозиціонування

Перш за все розстановлено функції матеріалів, які створили. На рисунку їх три, адже третя – це функція матеріалу доріжки в поселенні, яка потрібна для «малювання» по ландшафту. Перш за все у нодовому просторі розміщуємо ноду

BreakMaterialAttributes, яка узагальнить всі вхідні дані та зможе передати в звичайну ноду матеріалу. Створюємо її та з'єднуємо RockMatFunc з параметром A та GrassMaFunc з'єднуємо з B. Останнім кроком є реалізація самого розділення: виставлення міри та різкість «градієнту» від камня до трави. Створюємо ноду WorldAlignedBlend і два параметри. Перший параметр BlendHeight керує кутом нахилу, при якому трава буде замінена на камінь, а другий BlendSharpness керує плавністю перебігу. Підключаємо BlendSharpness до Sharpness, а BlendHeight до BlendBias. Із вихідних параметрів потрібен Explicit Normals, зв'язуємо його з параметром Alpha в ноді BlendMaterialAttributes. Напряму до параметру Specular у основній ноді матеріалу підключаємо параметр для її подальшого регулювання без внесення змін в основний код.

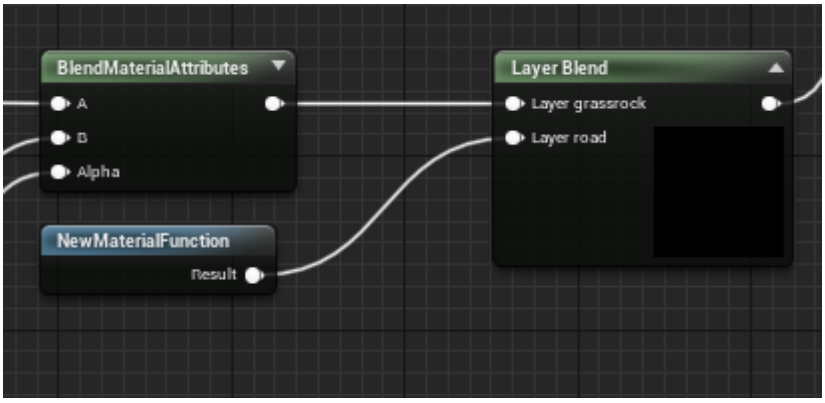


Рисунок 2.6 – З'єднані функції автотекстурування і звичайної текстури

На рисунку 2.6 вказаний спосіб реалізації слоїв для малювання дороги в поселенні. В NewMaterialFunction поміщено текстуру доріжки з нодою координат текстури, виставленої на збільшення в 3 рази. Цю та ноду BlendMaterialAttributes з'єднуємо в LayerBlend.

Таким чином, отримано ландшафт з адаптивним покриттям та можливістю малювати текстурою на ньому.

2.2 Моделювання природних об'єктів

Природні об'єкти – це об'єкти природного походження (дерева, камінь, тощо.). На ландшафт ГіС плановано додати лісові насадження та алею понад дорогою. Для цього треба дерева, деякі створені «з нуля» та деякі знайдені на Unreal Engine Marketplace, після чого змінено їм колір (темно-зелений) та

налаштовано розміри. Отже, є список дерев, які потрібні: тополя для насаджень біля дороги та берези для лісів. Останні знайдено на Unreal Engine Marketplace, змінено колір та встановлено розміри.

Рисунок 2.7 – Береза звичайна

Тополя зроблена за допомогою програми Tree It, в обирається кількість граней на стволі дерева, кількість гілок і т.д [10].

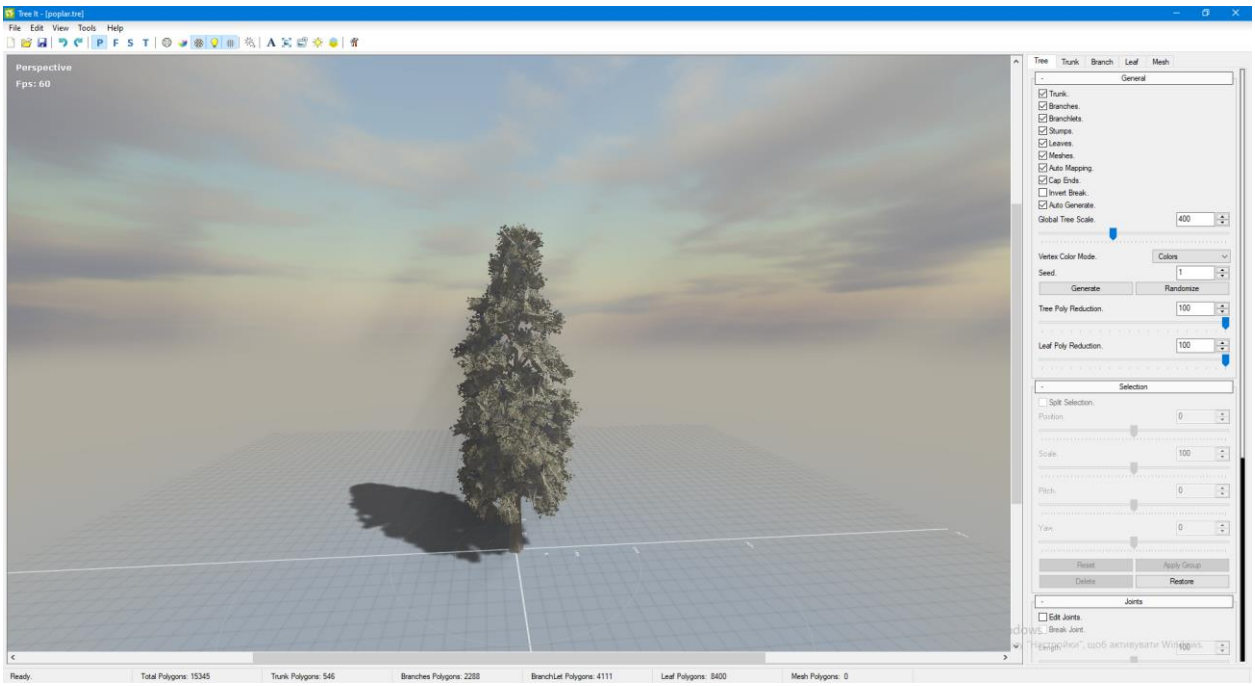


Рисунок 2.8 – Вікно TreeIT

Справа є кілька вкладок, кожна з них формує індивідуальний стиль дерева. Обираємо текстуру стовпа, кількість гілок, висоту гілок, нахил.

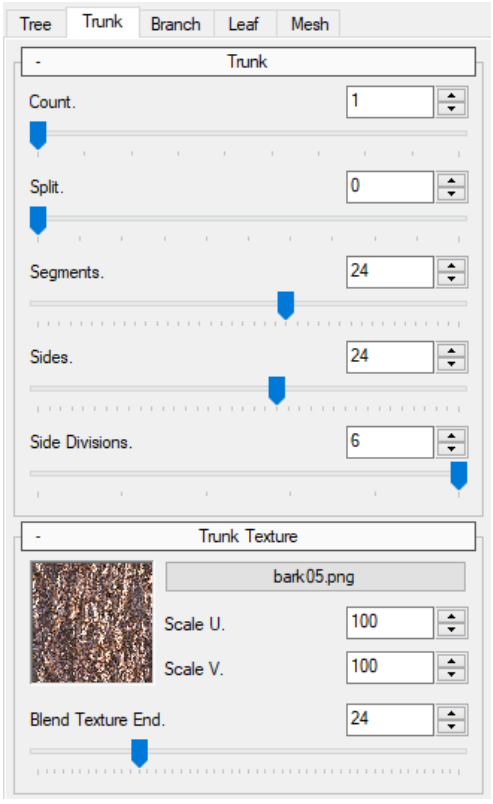


Рисунок 2.9 – Вкладка Trunk та текстура стовпа

Завершальним етапом є створення текстури листви на гілках. Це потрібно зробити вручну, адже в інтернеті немає нічого схожого. Скористаємось Adobe Photoshop. За основу візьмемо фото загального плану тополі.

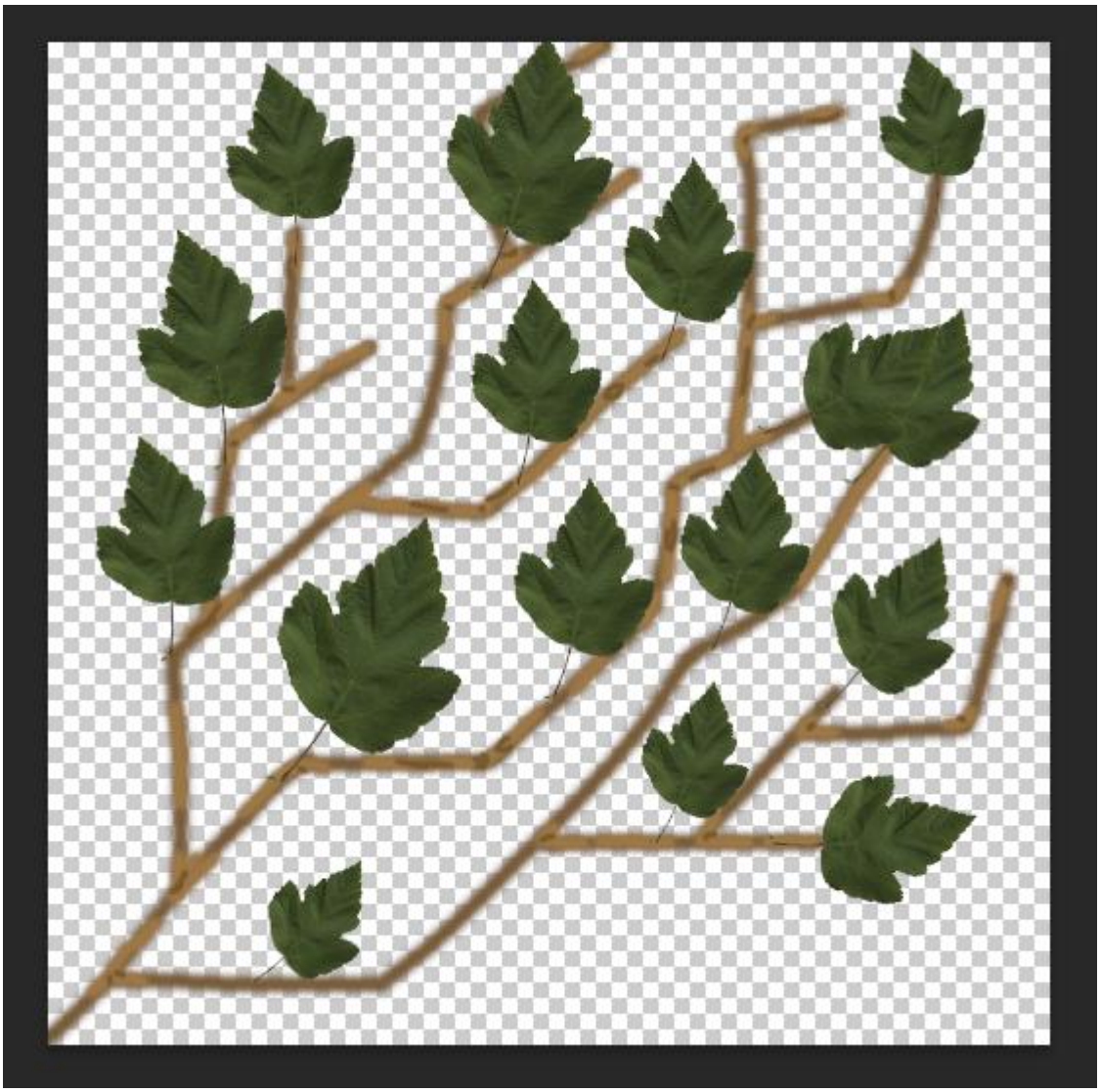


Рисунок 2.10 – Готова текстура гілочок тополі

Зберігаємо рисунок у форматі PNG щоб збереглись прозорі частини, після чого завантажуємо в TreeIt.

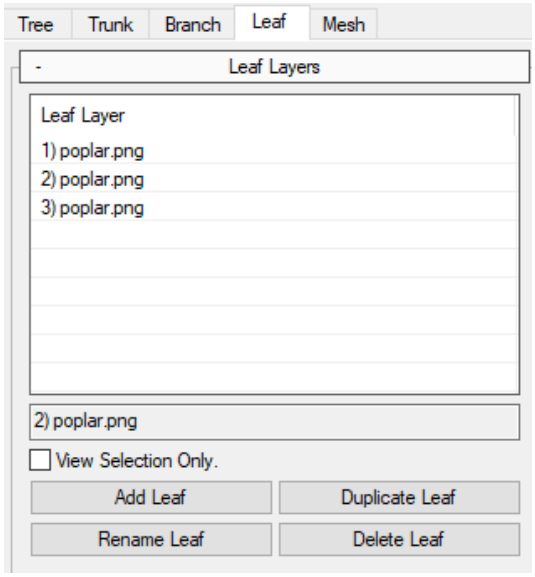


Рисунок 2.11 – Завантажене листя

Тополя готова, треба експортувати модель в Unreal Engine. Для цього переходимо в меню File -> Export -> FBX. Даний файл переносимо у вікно програми UE4, після чого даємо згоду на імпорт та перевіряємо об’єкт.

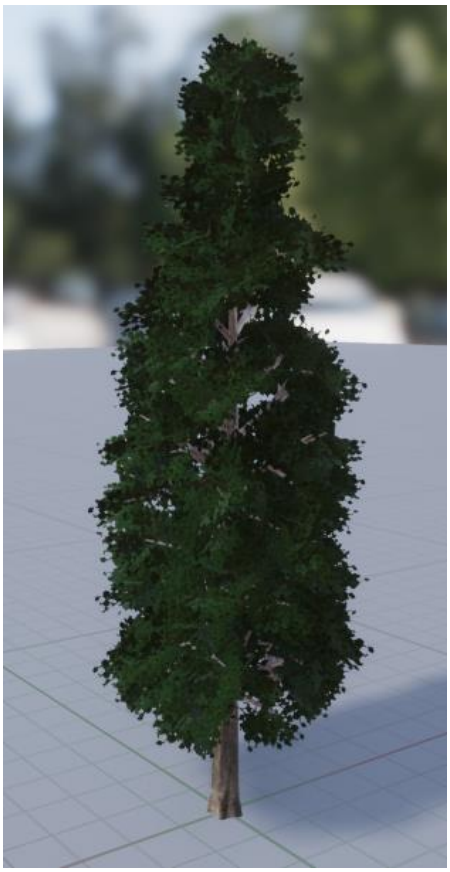


Рисунок 2.12 – Дерево в UE4

Таким чином, є дерева, які готові для розміщення на ландшафті. Розміщуємо згідно плану, представленому на рисунку 2.1.

Розміщення реалізовано через режим відображення Unreal Engine 4 Foliage. Після переходу в режим, зліва у вікно треба пересунути потрібні об’єкти для розміщення.

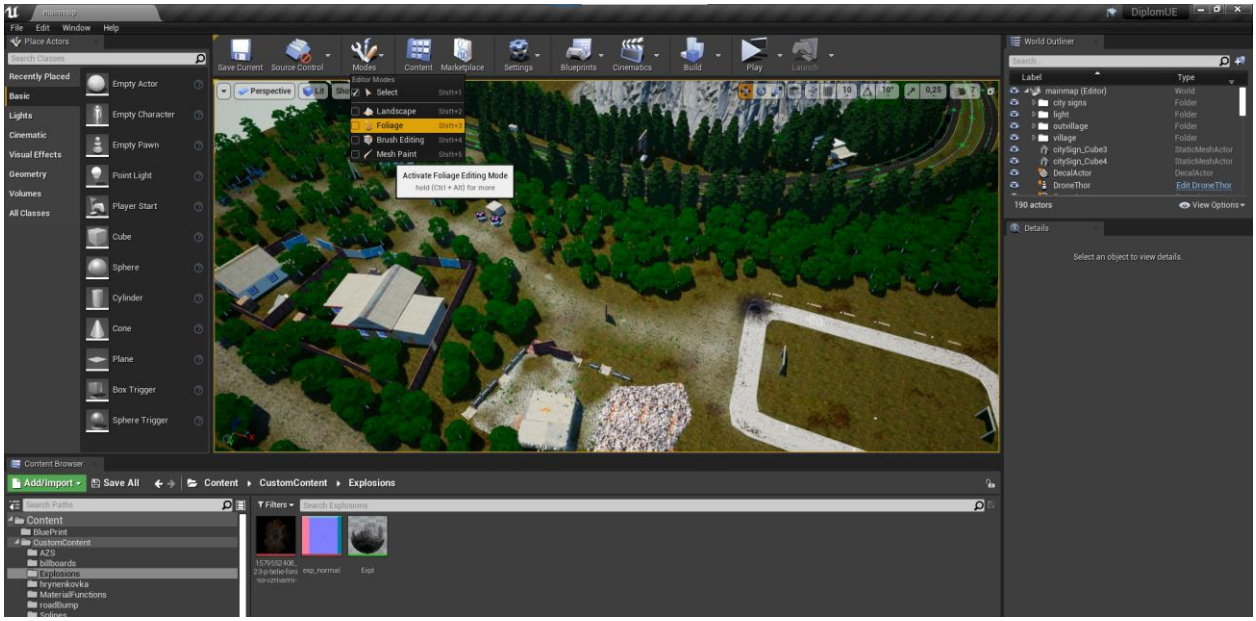


Рисунок 2.13 – Foliage в контекстному меню

Потрібно обрати розмір кожного об’єкта в форматі: мінімальний розмір – максимальний розмір, таким чином розмір буде варіюватись. Потрібно обрати розмір «пензля» та густоту заповнення при малюванні. Густота впливає на кількість дерев в середині пензля.

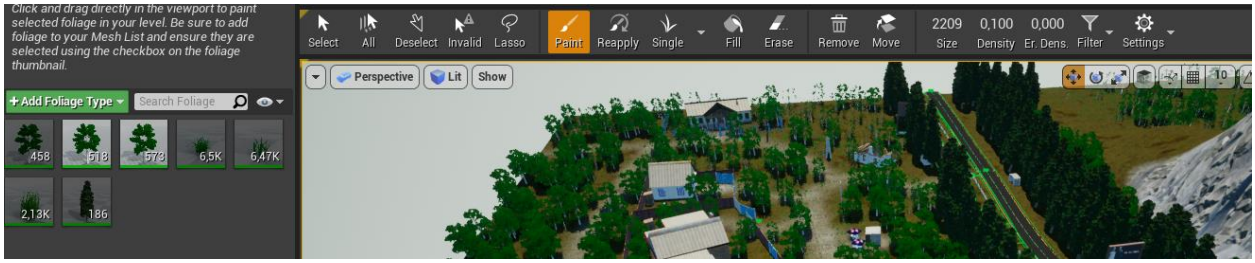


Рисунок 2.14 – Налаштування пензля малювання

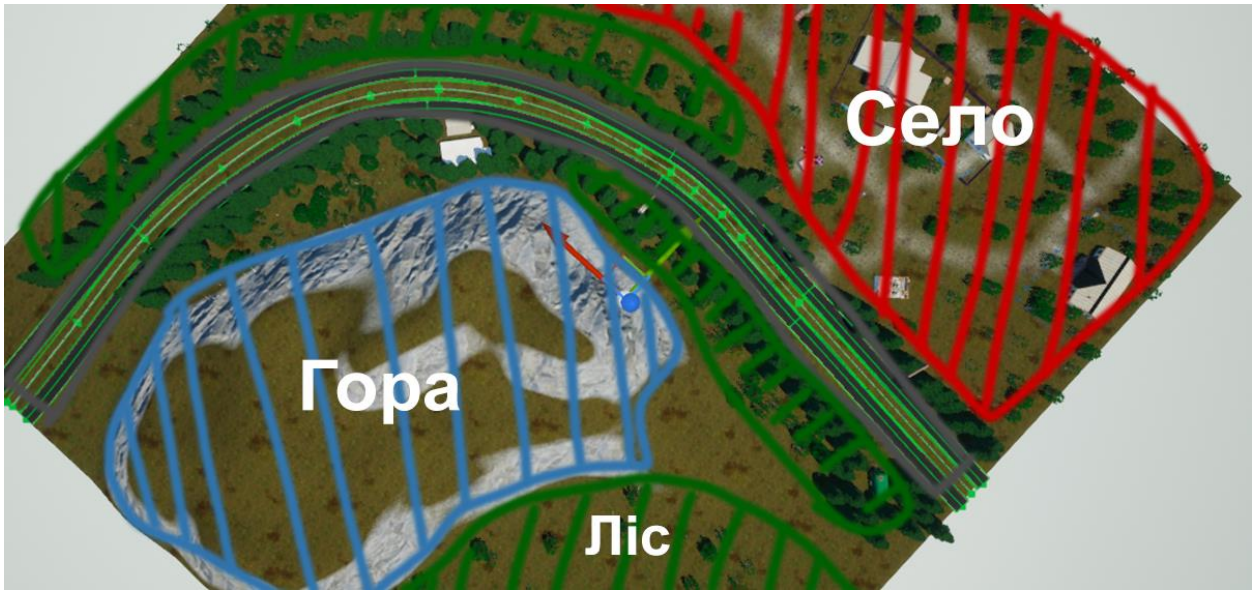


Рисунок 2.13 – Розставлені дерева та інші насадження

Таким чином, в наявності на даний момент лісові насадження та дерева впродовж дороги та ландшафт, на якому це все розміщено. По плану залишилось змоделювати та побудувати поселення обабіч дороги.

2.3 Моделювання інфраструктурних об’єктів

Перед початком моделювання треба ознайомитись з інтерфейсом програми Blender. Після входу в програму треба створити новий проект (для кожного об’єкта свій проект). На рисунку 2.14 показано стартове вікно, в якому треба натиснути General під категорією New File.

Рисунок 2.14 – Стартове вікно Blender

Методи та інструменти моделювання та текстурування продемонстровані на прикладі створення одного з будинків, після побудови якого можна буде змоделювати будь що. Перше, що треба зробити – знайти відсилку в реальному світі. Якщо її немає, то треба продумати побудову, матеріали, тощо. Поточний об’єкт повинен бути старим, на половину закинутим будинком зі знищеними вікнами. Можливі ознаки неофіційних жильців.

Основою будь-якого приміщення є стіни. У випадку зі старими будинками, матеріал для основи стіни може бути зруб. Після натиснення комбінації клавіш Shift + A, у меню вибрати Mesh, а там Cylinder (рис. 2.15).

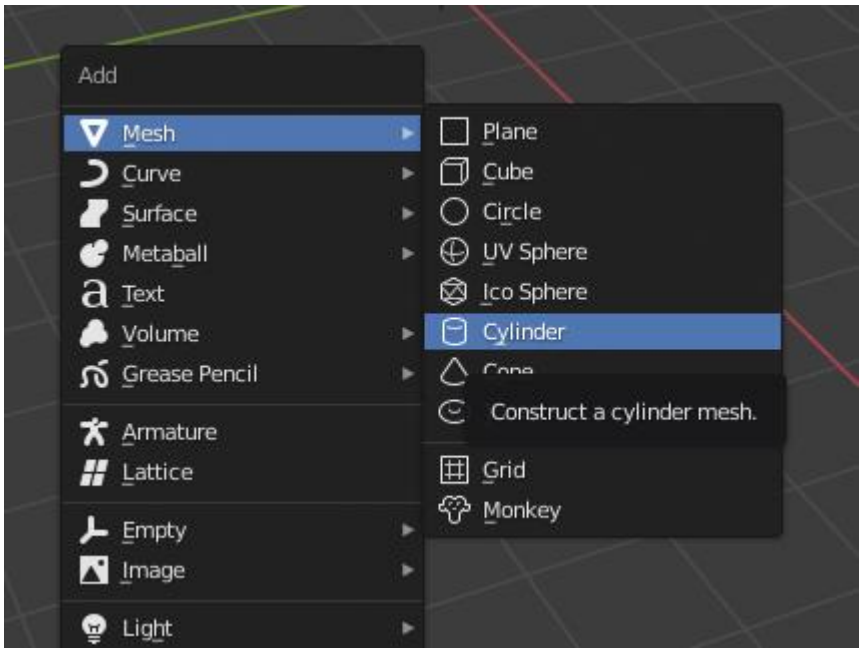


Рисунок 2.15 – Тип об’єкта Cylinder

При створенні об’єкта в нижньому лівому куті з’явиться вікно налаштувань. Для оптимізації моделі використаємо шестигранник, адже більш гладка форма циліндра є непрактичною, тому, що дрон летить зверху, на певній висоті. Отже, в 3D просторі є циліндр, який потрібно зробити довгим та скопіювати декілька раз для утворення стіни. Щоб змінити розмір, треба обрати фігуру та натиснути гарячу клавішу S (від слова Scale), яка відповідає за розмір вибраного об’єкта. Тепер можна змінити розмір, але у всі сторони (x, y, z). На цьому етапі треба зазначити, що в Blender є 3D сцена, на якій об’єкти можна роздивитись з різних сторін. Для зручності була додана Декартова система

координат. В 3D просторі представлені 3 осі: X (червона), Y (зелена) та Z (синя). Для зручності був доданий курсор для зручного візуального переміщення по осям.

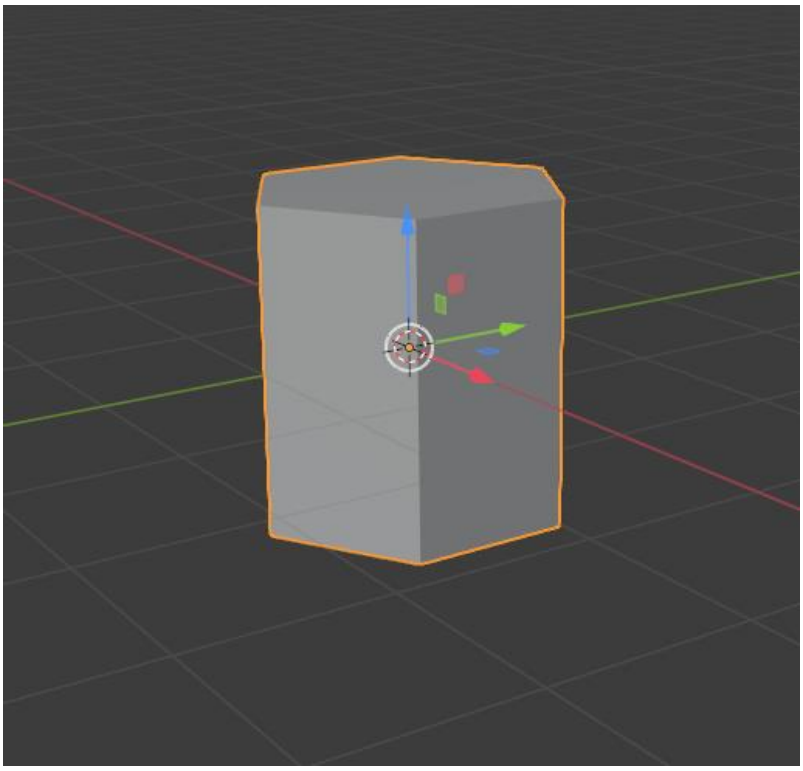


Рисунок 2.16 – 3D курсор

На рисунку 2.16 видно, що для схожості з брусом циліндр слід змінити в розмірі по осі Z. Для цього натиснемо S, після чого треба вказати на яку вісь вплине зміна розміру. Тиснемо гарячу клавішу Z, на 3D сцені з’явилась синя полоса впродовж осі, після чого тягнемо мишу та збільшуємо, після чого натискаємо ліву кнопку миші. Циліндр змінився, розширившись з центру в дві сторони. Так вийшло, тому що у кожному об’єкту на сцені Blender присвоює «центр об’єкту», від якого відштовхуються всі дії з фігурами. Отже, зараз є продовгуватий циліндр, але він стоїть вертикально. Щоб провернути, треба натиснути гарячу клавішу R (від слова Rotate) і вказати на вісь навколо якої буде обертатись стовп. В даному випадку, різниці між віссю X або Y немає, тому я обираю перше. Після цього можна покрутити циліндр, але модна більш точно та швидко повернути фігуру. Є спеціальна функція для повороту по градусам. Для її використання треба під час повороту через клавіатуру цифрами написати кількість градусів, після чого натиснути лівою кнопкою миші. Таким чином,

написавши 90 градусів для повороту, циліндр розташується горизонтально. Після будь яких змін об’єкту треба оновити його характеристики, тому, що потім деякі модифікатори або деформації будуть виглядати неправильно. Це можна зробити через меню, яке викликається комбінацією клавіш Control та A та при цьому обновлювана фігура має бути обрана. Стіна складається з декількох брусів, отже треба скопіювати об’єкт декілька разів. Та в цьому немає потреби, адже в Blender є інструмент «Модифікатори», який надає доступ до переліку функцій з різним призначенням. З цього переліку потрібен модифікатор «Array», який скопіює об’єкт задану кількість разів. Щоб накласти модифікатор треба перейти у вкладку «Modifier Properties» та натиснути «Add Modifier», де обрати «Array». З’являться налаштування, в яких важливим є параметр Count та група параметрів Relative Offset.

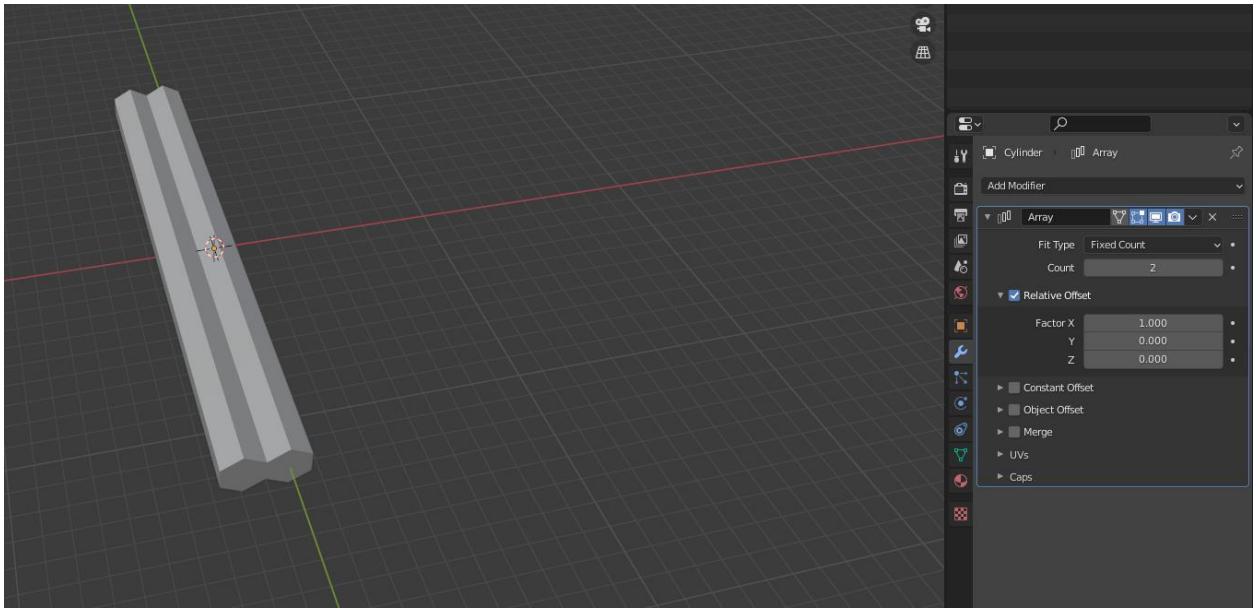


Рисунок 2.17 – Циліндр з модифікатором «Array»

Таким чином, можна змінити параметр Count на 6, після чого змінити напрям копіювання. Для цього треба напрям X змінити на 0 (прибрати зсув по осі X), після чого збільшити зсув по осі Z (коректне копіювання можливе після оновлення його rotation).

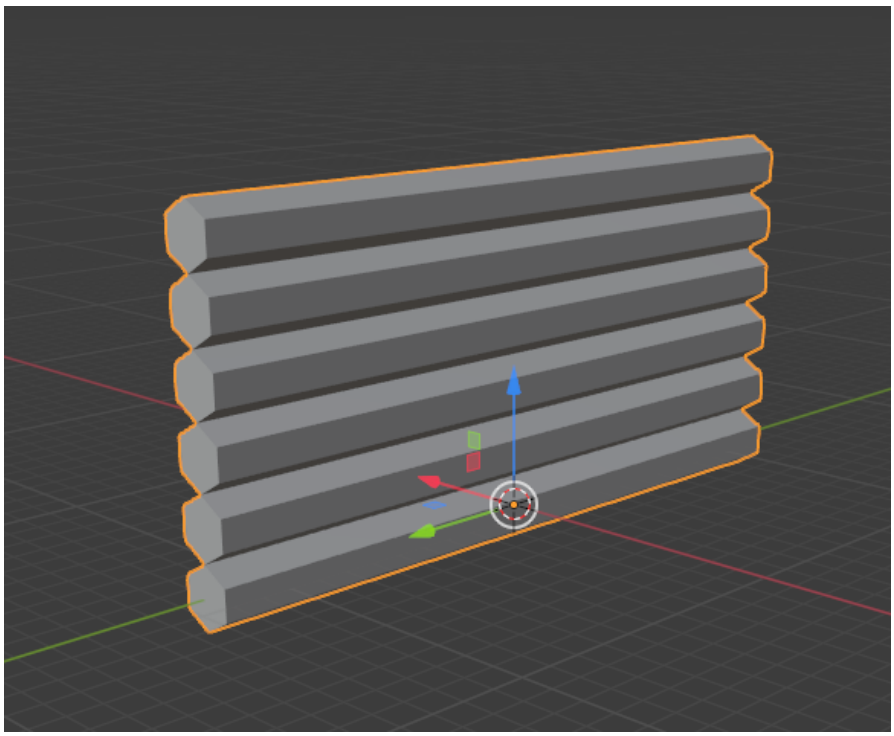


Рисунок 2.18 – Результат роботи модифікатора

На рисунку 2.18 показано роботу модифікатора. Така стіна схожа на реальність. Останнім кроком є утвердження Array. Для цього в меню з параметрами модифікатора розгортаємо стрілочку та натискаємо Apply. Відтепер всі бруски єдине ціле.

Будинок – це простір, оточений чотирма стінами. Це означає, що треба скопіювати стіну три рази. До модифікатора звертатись не будемо, звернемось до звичайного копіювання через Shift + D з вибраним об’єктом. Ця функція сумісна з переміщенням по координатам, отже можна рівно по осям перемістити стіни майбутнього будинку. Застосовуючи набуті навички результатом є:

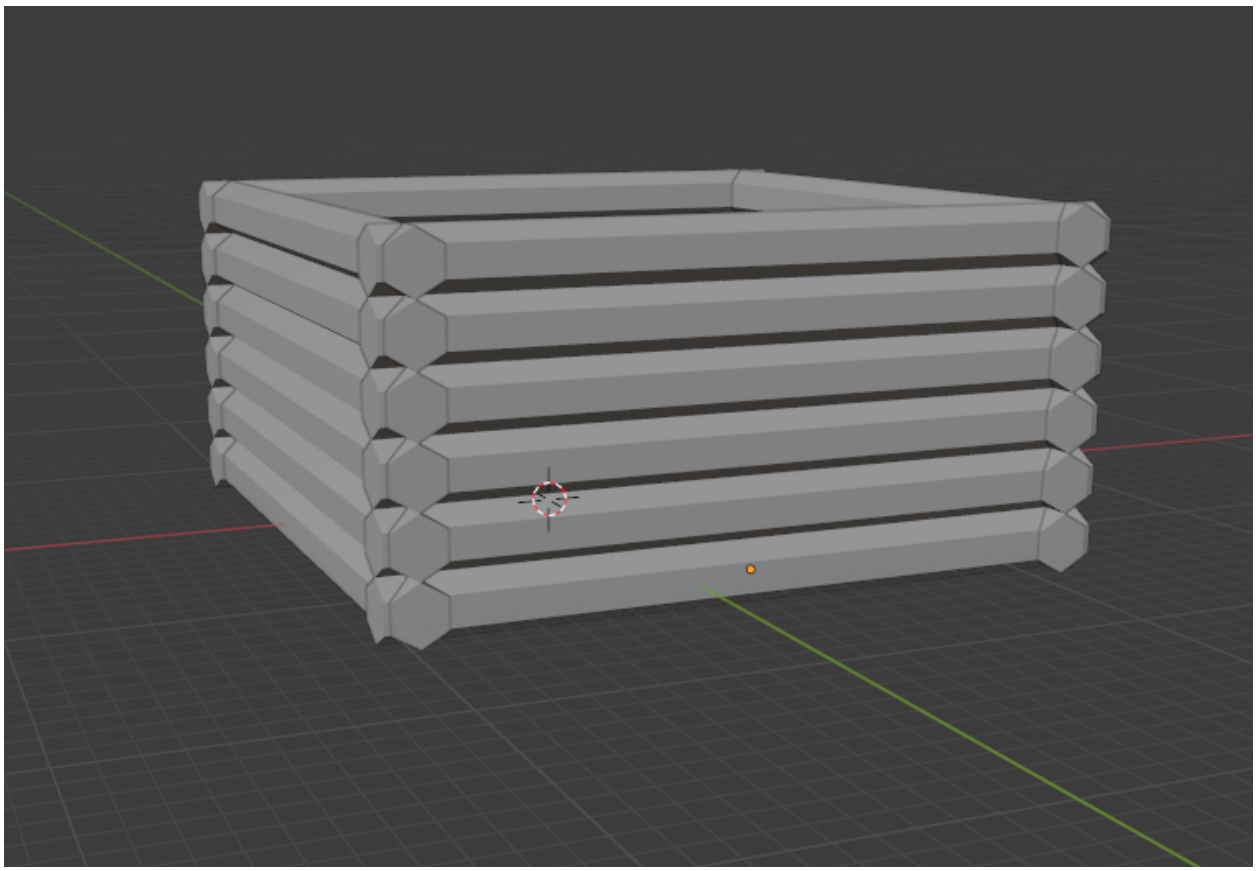


Рисунок 2.19 – Результат моделювання

Найголовніша деталь будинку – двері. Перед їх моделюванням потрібно вирізати дверний отвір. Blender має декілька інструментів для вирізки отворів на об’єкті: Ножиці, але вони підходять для різання плоских об’єктів, планок; Булевий модифікатор, який працює за логікою, додавання або об’єднання об’єктів. Так як стіни зроблені не з планок – обираєм булевий модифікатор. Щоб вирізати певну область на об’єкті треба створити ще одну фігуру з формою, яку треба вирізати з іншої фігури. Отже, треба створити прямокутник та перемістити на місце вирізу.

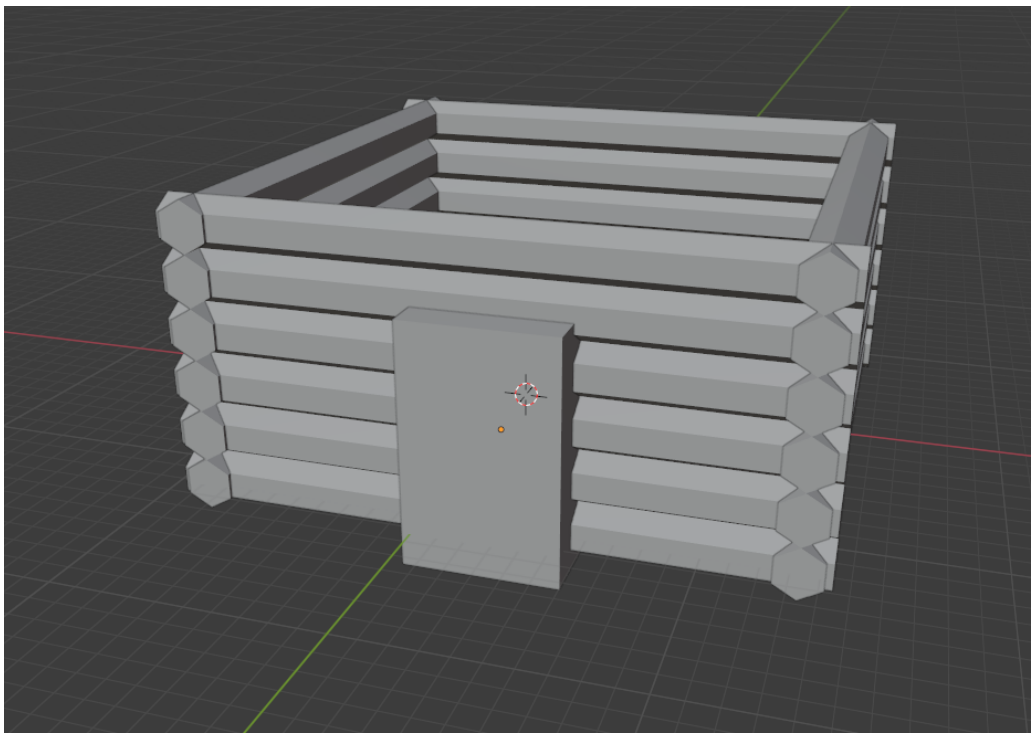


Рисунок 2.20 – Об’єкти готові вирізати

Тепер, натиснувши на об’єкт в якому треба вирізати частину, треба перейти до меню модифікаторів, після чого обрати там Boolean. У меню, яке з’явилось, обрати потрібний тип булевого зв’язку (intersect, union, difference). Intersect зберігає те, що всередині прямокутника, видаляючи все інше. Union об’єднує два об’єкта. Difference видаляє все, що потрапило під прямокутник, по суті – це обернена Intersect. Останнім можливо вирізати отвір. Завершальним кроком є вибір об’єкта, який зробить отвір. В меню модифікатора в параметрі Object треба застосувати інструмент «піпетка» та натиснути на прямокутник, після чого застосувати модифікатор. Після цього можна видалили тимчасовий прямокутник. Таким чином, модифікатор Boolean застосовується часто, не тільки для дверей, але і для вікон, наприклад.

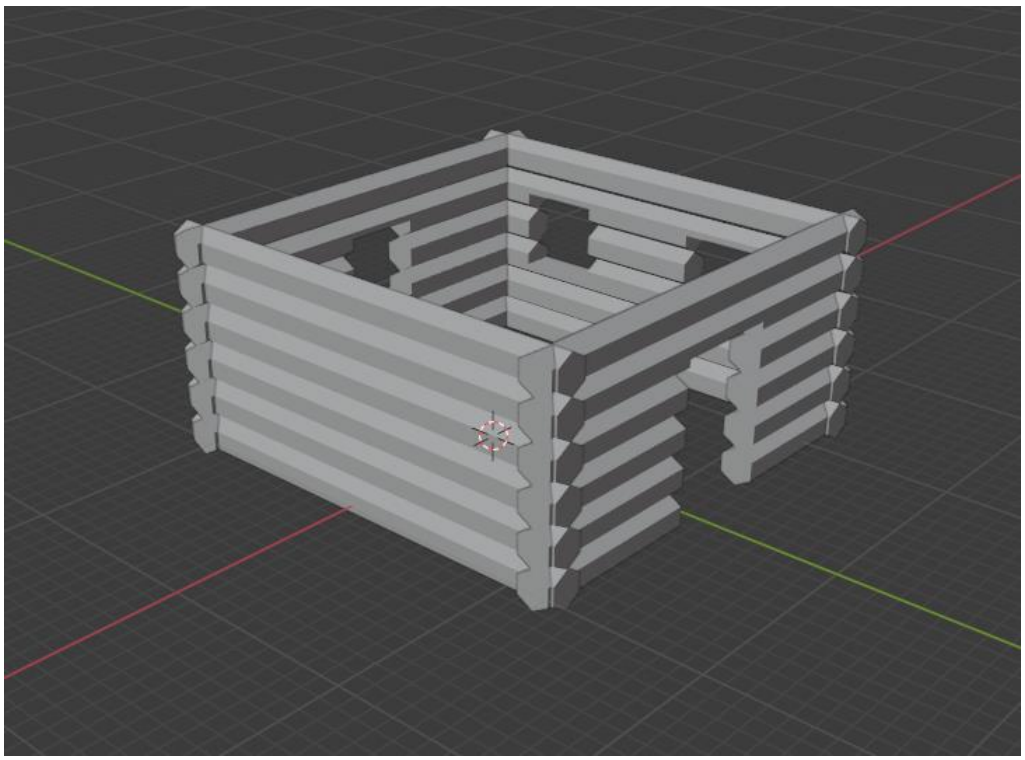


Рисунок 2.21 – Будинок після застосування надбаних навичок

Так як будинок закинутий, дах повинен бути пошкоджено та відремонтовано підручними засобами (брезент, тканина, листовий метал, тощо). Основна частина даху зроблена з глини, а зверху неї все інше. Для створення основи знадобиться об’ємний прямокутник, який потрібно нахилити. Для наступної дії треба навчитись користуватися Edit Mode. Це режим редагування полігонів, який дозволяє рухати окремі полігони (faces), вершина (кути) та грані. Вони потрібні для більшої деталізації об’єкта, адже кожна вершина Щоб перейти в Edit Mode треба натиснути на Tab, після чого об’єкт буде обведений лініями та кути відмічені точками. Таким чином, можна перемістити окрему частину об’єкту в будь яке місце.

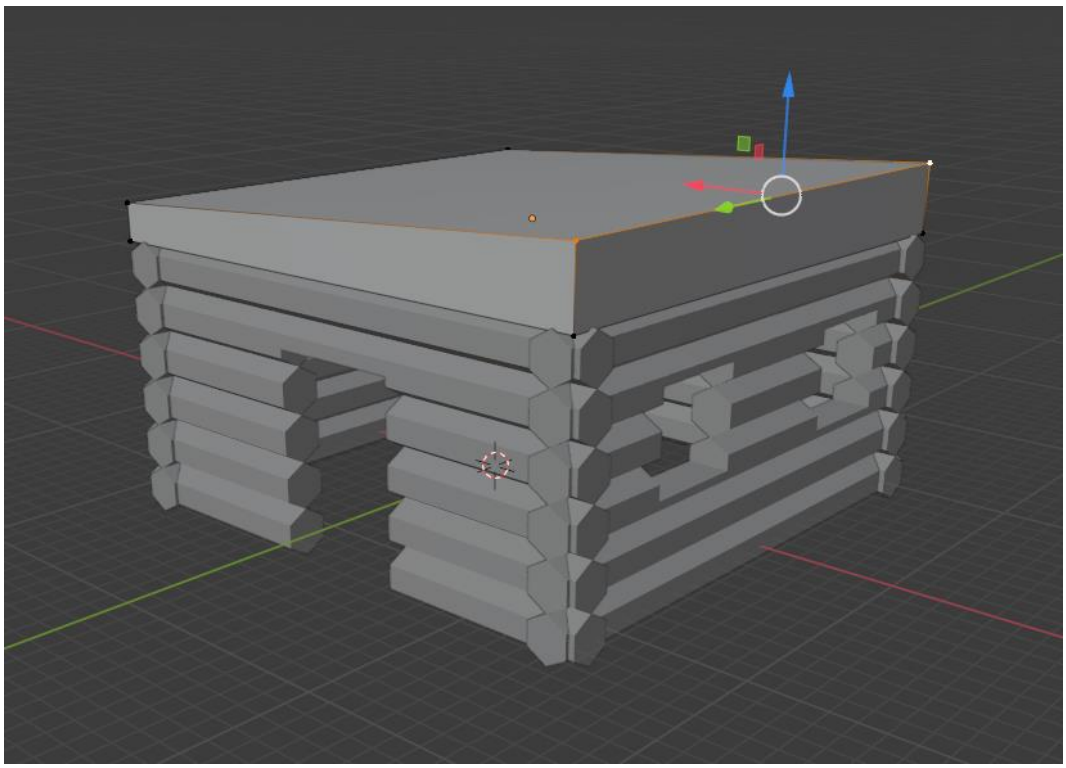


Рисунок 2.22 – Дах з піднятим ребром

Після всіх маніпуляцій можна вимкнути режим Редагування. За допомогою здобутих навичок використання модифікатора Boolean прорублені отвори в крівлі.

Зважаючи на дірявий дах, треба його відремонтувати підручними засобами. Отже, накинемо брезент, листовий метал та трішки шиферу. Шифер реалізовано видозміненням прямокутної фігури, на яку згодом буде нанесена текстура шиферу. Шматок брезенту покладемо на дах за допомогою можливості Blender симулювати прості фізичні процеси. В даному випадку потрібна фізика падіння тканини.

Для створення симуляції потрібно створити сам брезент. Він реалізований через фігуру планку (не об’ємний чотирикутник). Після створення треба наділити її фізикою тканини. Нижче меню модифікаторів є настроювання фізики.

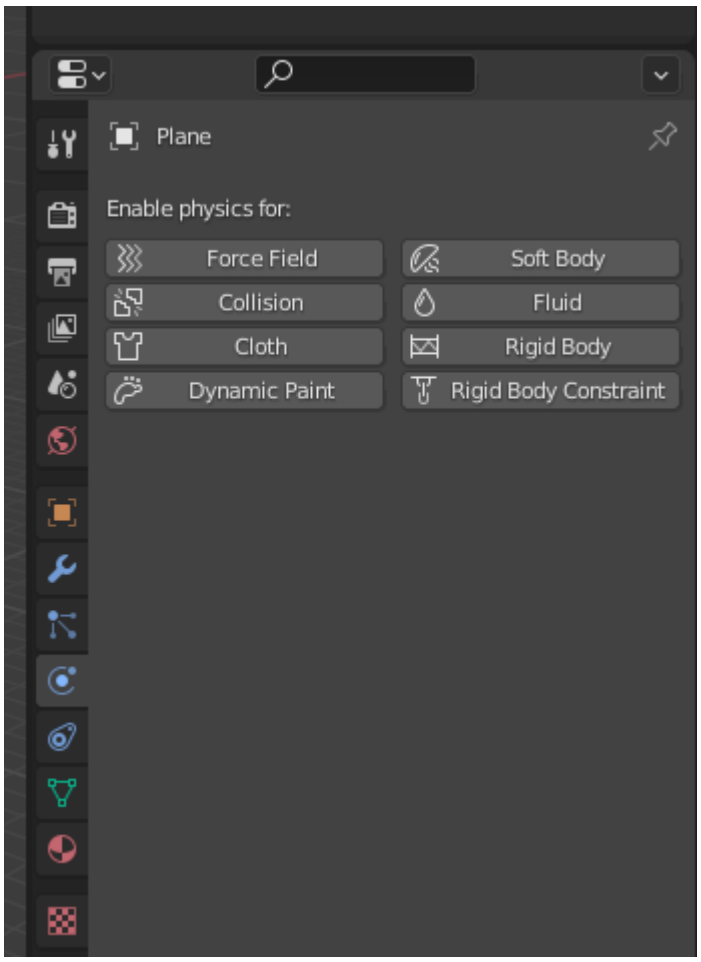


Рисунок 2.23 – Настроювання фізичних властивостей

З представлених опцій на рисунку 2.23 треба ввімкнути параметр Cloth, після чого з’явиться вікно налаштувань. Також для коректної роботи симуляції треба увімкнути основі даху колізію (можливість зіткнення між об’єктами), це можна зробити в тому самому меню (опція Collision, нічого змінювати не треба). Також треба зробити планку-брезент більш полігональною (збільшити кількість вузлів, які фізика може змінювати та переміщувати). Для даної задачі використано інструмент Subdivide, який працює в режимі Edit Mode, та при виборі всіх вузлів викликається через праву кнопку миші. Після активації з’явиться меню, в якому параметр number of cuts змінює кількість розрізів. Максимум – 10, але можна повторити процедуру декілька разів (достатньо 15 розрізів в сумі). Функція застосовується самостійно. Далі є два варіанта розвитку подій, при яких є варіант з «вивернутими» нормалями, результатом яких є неправильна робота симуляції. Нормалі – це перпендикулярна до поверхні

невидима лінія, яка показує направлення кожного полігону. За їх допомоги розраховуються відбиття світла, накладку текстур (при імпорті в Unreal Engine це важливо), розрахунки фізичних процесів, тощо. В Blender є візуальне представлення направлення нормалей – це синій та червоний колір. Якщо увімкнути режим відображення направлення нормалей, то можна побачити один з двох кольорів. Червоний – це направлення нормалей всередину, отже вони потрібні всередині об’єкта або фігури. Синій колір показує зовнішні нормалі, які направлені назовні. Таким чином симуляція може побачити внутрішню нормаль та не зіткнутись з нею. Зіткнення відбудеться коли об’єкт впаде на зовнішню нормаль цього ж об’єкта, яка знаходиться всередині. Результатом є неправильна робота фізики. Такі проблеми виникають, в основному, при зміненні розміру та його затвердженні (скинути розмір до 1). Після зміни розміру фігури нормалі можуть помінятись місцями. Для перевороту нормалей (в Blender це називається Recalculate Normals) треба в режимі Edit Mode виділити всі вузли клавішею A після чого комбінацією Shift та N перерахуються нормалі.

Після того, як нормалі будуть повернуті в правильному напрямку можна запускати анімацію. В нижній частині вікна Blender є часова шкала, яка відповідає за інформування користувача про час анімації. Щоб природно покласти брезент на дах, треба увімкнути програш анімації на клавішу Space.

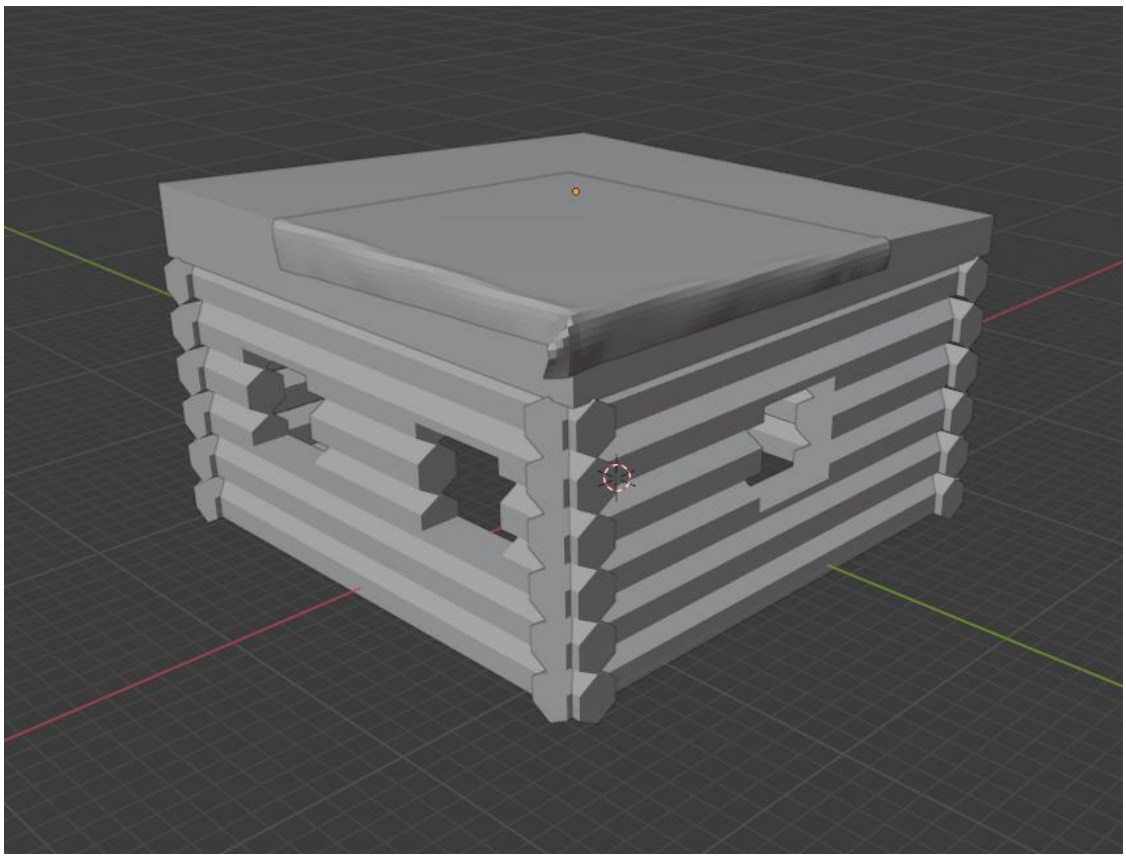


Рисунок 2.24 – Результат роботи анімації

На рисунку 2.24 продемонстрований 46-й кадр анімації шматка брезенту. Щоб залишити цю форму планки треба в меню «Модифікатори» натиснути на Apply в опції Cloth.

Двері створені з дерев’яних дошок, які прибиті до залізних планок з петлями. За допомогою набутих навичок можна змоделювати деформовані та діряві планки та розмножити їх модифікатором Array. Щоб зробити дверну петлю треба на створеному прямокутнику застосувати інструмент LoopCut. Петля знаходиться на одному з країв залізної планки, отже треба відділити частину прямокутника з краю. LoopCut доступний в режимі Edit Mode, створює жовту лінію, яку можна перемістити та натиснувши ліву кнопку миші затвердити.

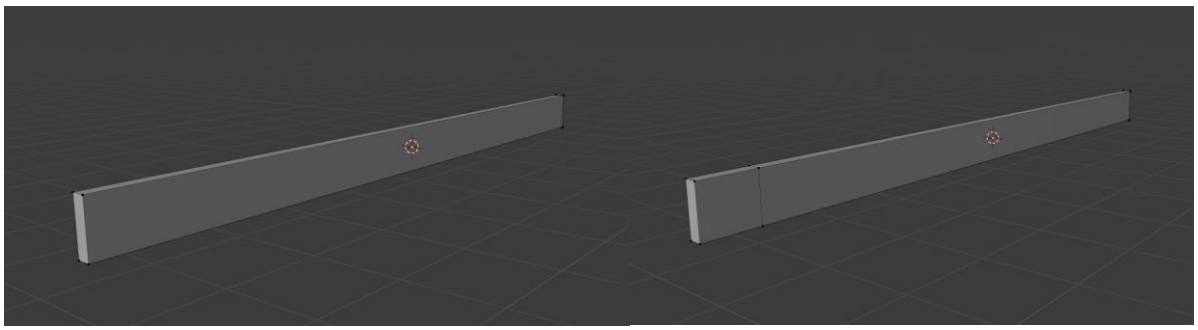


Рисунок 2.25 – До та після LoopCut

Створення петлі на планці передбачає присутність рельєфу для закріплення на дверній рамі. На рисунку 2.25 продемонстровано роботу інструменту LoopCut, за допомогою якого зроблено розділення полігонів. Треба обрати менший полігон на плоскій більшій стороні та екструдувати. Extrude – це інструмент для копіювання вузлів (vertices) та з’єднання між оригіналом та копією. Таким чином можна скопіювати цілий полігон. При виборі чотирьох вершин так, щоб утворився чотирикутник, після чого підсвітиться обраний полігон. Після Extrude можна взаємодіяти з копією за допомогою раніше вивчених функцій та інструментів (переміщення по осям). Затвердити позицію копії можна лівою клавішею миші. Тепер петлю треба округлити за допомогою інструменту Bevel. Він округлює вибрані грані або вершини додаючи їм полігонів. Використати інструмент можна в режимі Edit Mode, після входу в який треба обрати потрібні грані (2 зв’язаних вершини або більше) та натиснути комбінацію клавіш Control та B. За допомогою руху комп’ютерної миші можна налаштувати розмір округлення, а колесом миші можна виставити кількість розрізі. Застосовується по аналогії з іншими «швидкими» функціями, які визивають через гарячі клавіші.

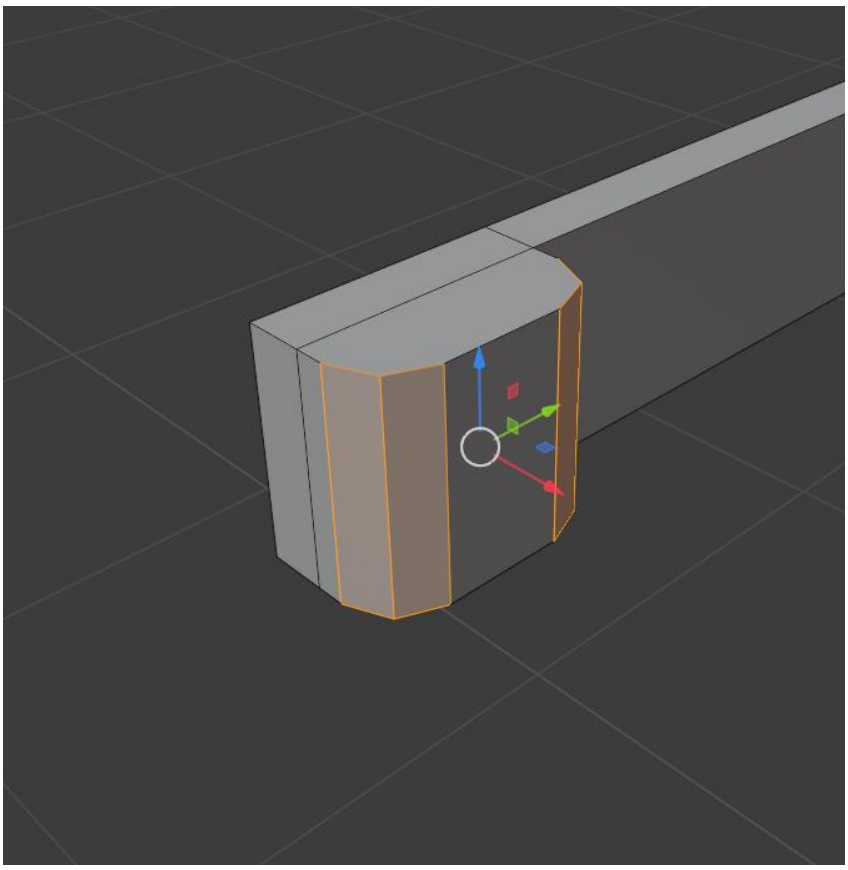


Рисунок 2.26 – Результат роботи Bevel

Завершальним кроком створення дверей є ручка. Її можна створити за допомогою вивчених раніше Subdivide, Extrude. З нових функцій можна виділити Bridge Faces, за допомогою якого можна зв’язати два полігона.

Вікна зроблено за допомогою куба, зміненого в розмірах під зроблені отвори в стінах та поділеного Subdivide. Після ділення у виді віконної рами та видалити полігони та, де могло бути скло. Результатом буде об'єкт схожий на вікно, але с дірками в полігонах. В даному випадку треба інструмент Fill (гаряча клавіша F). Для заповнення треба виділити 4 сусідніх вершини та активувати інструмент, після чого на місці пустоти з'явиться полігон. Повторити стільки разів, скільки отворів в полігоні залишилось.

На даний момент фігура має сіре забарвлення, це стандартна текстура. Текстура – це зображення накладене на об’єкт або його окремі частини з ціллю відобразити матеріал та його характеристики. В Blender є спеціальний нодовий простір, який дозволяє за допомогою візуально простого інтерфейсу додавати

текстури, додавати їм властивості. Перед початком накладання зображень на об’єкти треба ознайомитись з інструментами редагування в середовищі Blender.

UV карта – це проекція сторін об’єкта на площині текстури з подальшою можливістю переміщувати або змінювати розмір кожного полігону текстури. Для кращого розуміння можна привести приклад паперових саморобок.

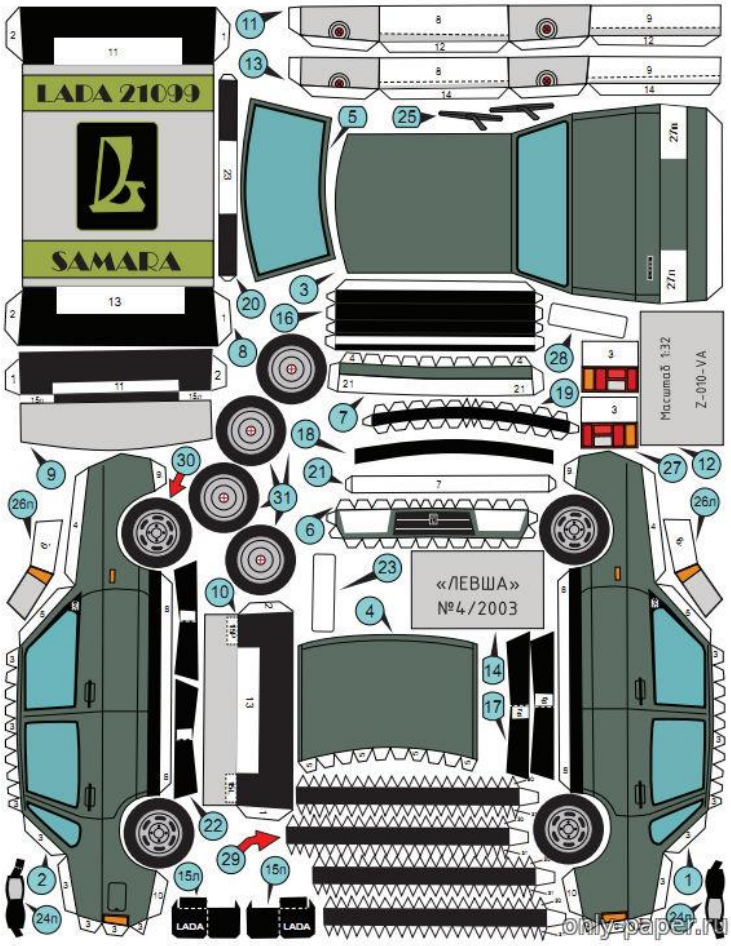


Рисунок 2.27 – Паперова модель ВАЗ-21099

Одним із головних інструментів по роботі з UV розгортками в Blender є UV Unwrap. Цей інструмент самостійно розділяє об’єкт на сторони, після чого розміщує їх на UV розгортці. Але, часом буває, що фігура надто складна по формі і UV Unwrap не спрацює, то ж треба використовувати Smart UV Project. Він схожий на попередній інструмент, але без автоматично складених сторін. Кожен полігон стоїть окремо, по порядку. Важливо примітити, що обидві функції правильно лише з розміром 1 (застосувати Scale).

Деякі текстури можна малювати вручну, але більшість треба високої якості. Отже, джерелами текстур є різноманітні пошукові системи (Google, Bing, тощо.).

Рисунок 2.28 – Приклад текстури

Щоб почати текстурування треба перейти на вкладку Shading, на якій кожен об'єкт буде показаний зі своїми нодами матеріалів. Зазвичай, матеріал треба створити з нуля. У вікні нижче робочої області натискаємо «New Material», після чого створюється матеріал і стандартна нода. Щоб додати текстуру достатньо перенести її з Провідника (програма для переходу та огляду папок). Параметр Color з'єднати з Base Color, таким чином передаючи зображення в матеріал.

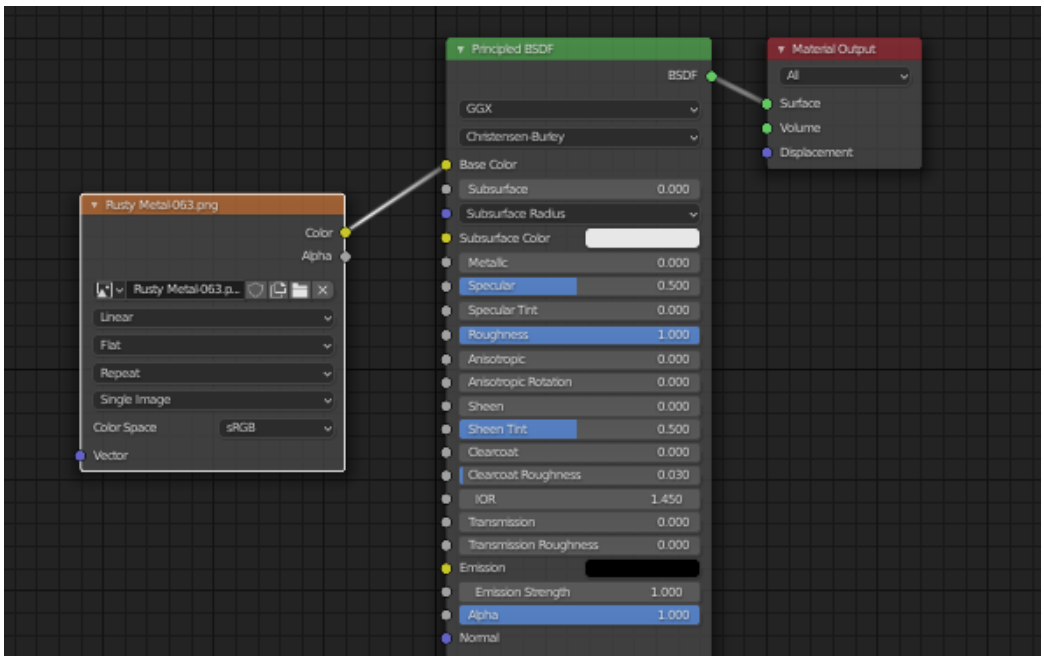


Рисунок 2.29 – Зображення текстури, підключене до матеріалу

З текстурою ще треба працювати, адже на вид вона не ідеальна. UV unwrap працює в Edit Mode у вкладці UV Editing. При виділенні вершин або всього об’єкту зліва вікно розділиться і там з’явиться раніше накладена текстура та зверху розгортка UV. Середовище редагування UV карт працює в декартовій системі координат, без осі Z. При натисненні на клавішу U відкриється контекстне меню з багатьма опціями. Серед них Unwrap побудує чітку розгортку. Бувають моменти, коли розгортка працює але некоректно, в такому випадку треба некоректний результат вручну перемістити за допомогою візуальних осей та задати розмір (на весь розмір текстури). Таким чином, текстури накладаються на об’єкт. З більш складними фігурами працювати довше, адже розкол чи відсутність частини об’єкту (в основному, це робота модифікатору Boolean) треба виділяти окремо та виставляти на текстурі.

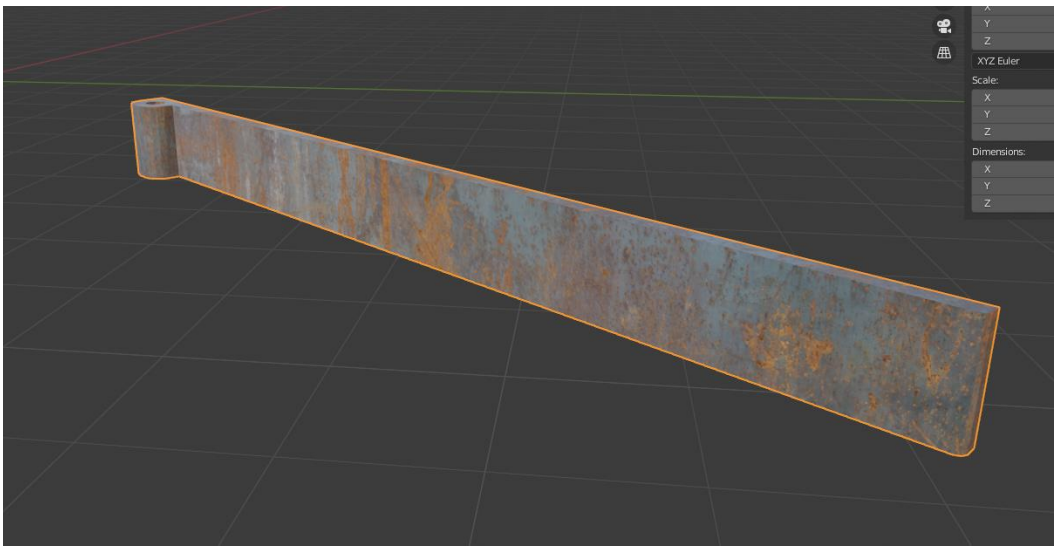


Рисунок 2.30 – Результат накладання текстури

Таким чином, здобуті навички текстурування та моделювання за допомогою яких реалізовані інші інфраструктурні об’єкти для геоінформаційної системи, окрім доріг.

Автомагістраль реалізовано за допомогою вмонтованої функції в Unreal Engine - «Сплайн». Завдяки цій можливості, можна «протягнути» відповідну лінію і призначити їй матеріал, який до цього був підготований в Blender. Сплайн працює як компоновщик або з'єднувач об'єктів. Таким чином, можна з маленької частини (рис. 2.31) зробити повноцінну дорогу.

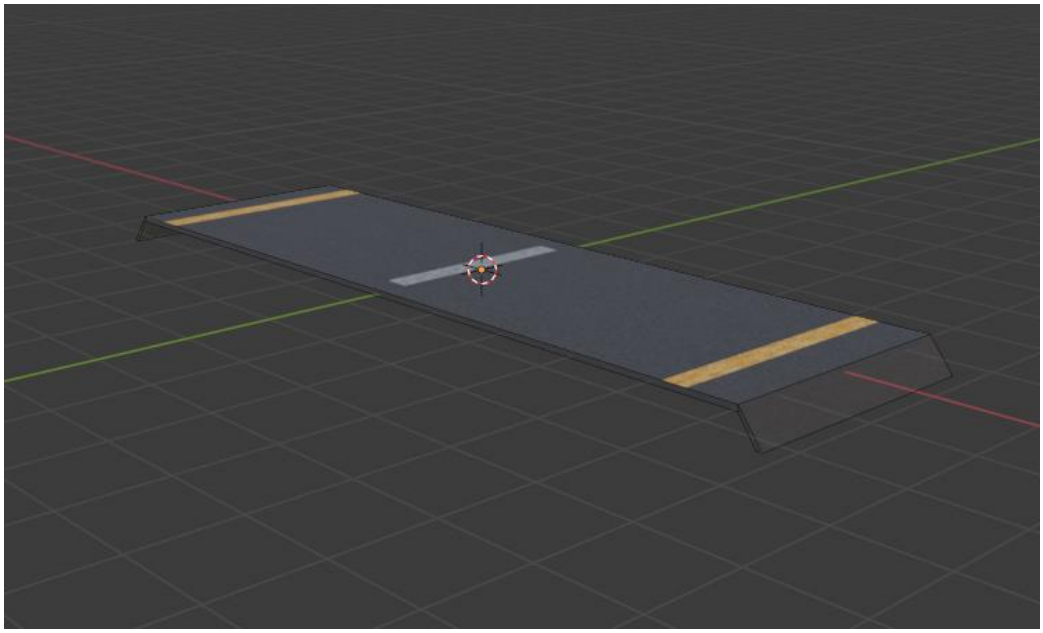


Рисунок 2.31 – Дорожна частина для сплайну

Щоб розмістити сплайн треба перейти в режим «Landscape», після чого лівіше обрати сплайн та з зажатим Control малювати по карті. Треба обрати об’єкт для сплайну. На рисунку 2.32 показано меню, в якому в перший раз треба натиснути «+», після чого в самому великому полі Mesh обрати заздалегідь імпортований об’єкт дороги (рис. 2.31).

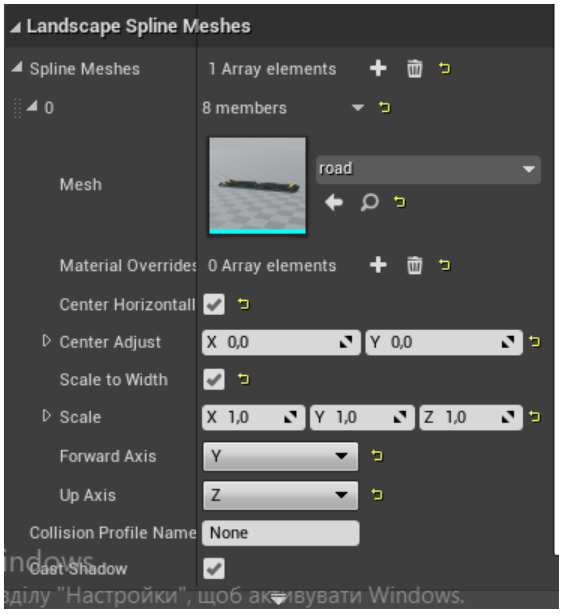


Рисунок 2.32 – Вікно з налаштуваннями сплайну

Окрім автомагістралі є сільська дорога. Вона реалізована через малювання текстурами на шарах (Layers). Їх треба увімкнути через меню налаштування ландшафту, де увімкнути опцію Edit Layers.

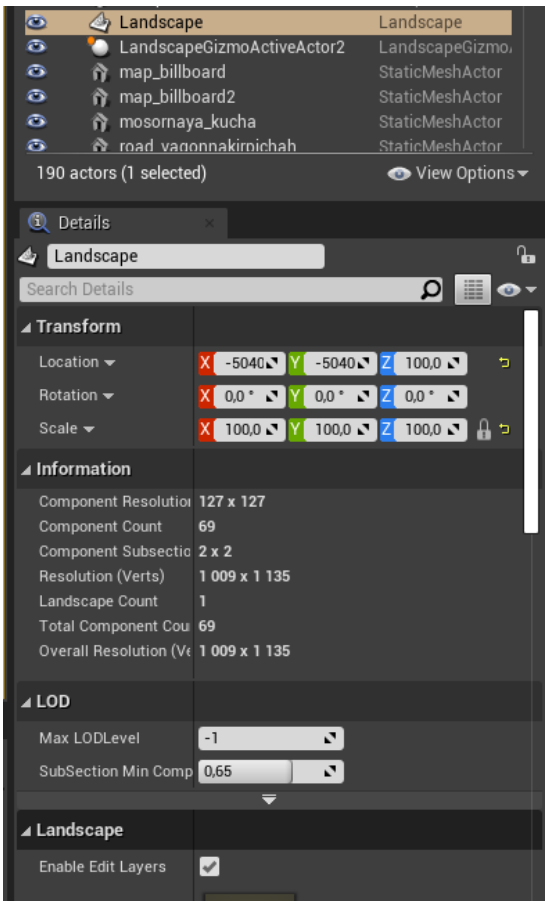


Рисунок 2.33 – Меню налаштування слоїв

Для реалізації дороги треба знайти відповідну текстуру, зменшити для більш реалістичного виду. Зменшення відбулось і через Photoshop шляхом дублювання текстури кілька разів, так і через матеріалі дороги в Unreal Engine. Нода TexCoord відповідає за зменшення текстури. Параметром є число, в скільки разів текстура буде зменшена.

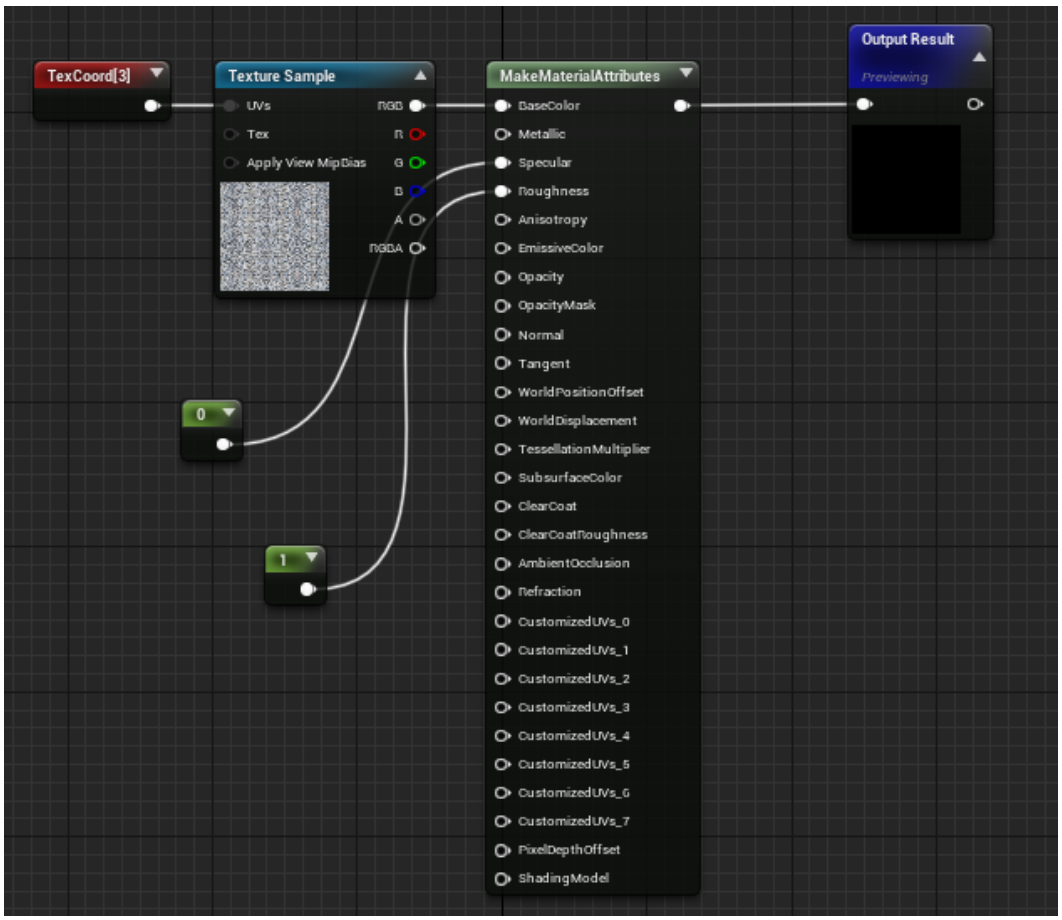


Рисунок 2.34 – Матеріал щебеневої дороги в селі

В пункті 2.1 при створенні ландшафту реалізовано поділення на шари. При переході в режим Landscape (Shift - 2) зліва у вікні є вкладка Target Layers, під якою є дві текстури (які комбіновано в 2.1). Їх треба зберегти, натиснувши на «+» поруч із кожним шаром. Щоб запобігти зміну шару з основною текстурою ландшафту треба створити новий слой та переключитись на нього та почати малювати через вкладку Paint в режимі Landscape. Серед параметрів пензлика є сила (strength), форма пензля, радіус, стиль градієнту по краям та плавність зникнення від центру пензля до країв.

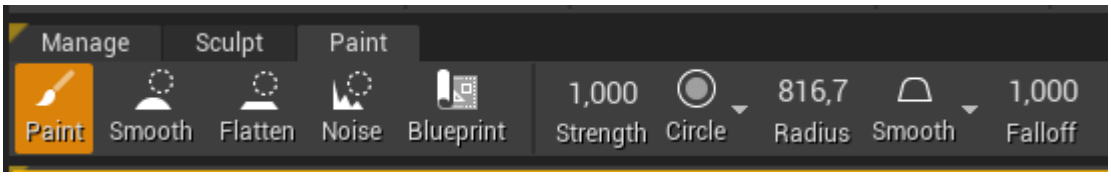


Рисунок 2.35 – Параметри малювання

Рисунок 2.36 – Результат роботи із шарами в режимі LandScape

Під час експорту створених інфраструктурних об’єктів із Blender налаштування залишити стандартними, тільки змінити формат на FBX. При імпорті об’єктів в Unreal Engine залишити налаштування стандартними, якщо є потреба в роздільному імпорті об’єктів з одного проекту (стовпи для вулиці, наприклад). В іншому випадку треба увімкнути «Combine Meshes», що скомбінує та перетворить кілька об’єктів в один.

На ландшафті переміщення реалізовано тією ж системою, що і в Blender. Гарячі клавіші для зручного переміщення та деформування об’єктів: W – переключитись на осі переміщення; E – увімкнути сфери для повороту об’єкта; R – зміна розміру предмету;

При розміщенні об’єктів може виявитись проблема розмірів. Їх можна контролювати на рівні створення в Blender, але інколи моделювання не таке точне, а розміри можуть незначно коливатись. Дану проблему можна вирішити і в Unreal Engine, збільшивши вручну за допомогою інструменту розмір (гаряча клавіша R при обраному об’єкті).

За допомогою Blender та Unreal Engine створено інші інфраструктурні об’єкти (будинки, кіоски, стовпи, тощо.) для геоінформаційної системи. Результати моделювання продемонстровані в розділі 3.4.

3 ІНФОРМАЦІЙНЕ І ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Створення моделі безпілотного авіаційного апарату

Безпілотний літальний об’єкт (БпЛА) – це літальний апарат, який в змозі виконувати поставлені задачі без присутності пілота в ньому [11]. Приступаючи до створення моделі безпілотного літаючого апарату треба визначитись з типом та конкретною моделлю. Отже, існує два види літаючих безпілотних апаратів: квадрокоптер та більш серйозна система БпЛА. Квадрокоптери відрізняються своєю дешевизною та можливістю зависати в повітрі. Окремо можна відзначити простоту керування, адже квадрокоптер може запустити будь хто, перед цим прочитавши інструкцію. Із мінусів можна зазначити швидкість переміщення та відносно до серйозних літальних об’єктів малу висоту польоту [12]. Як приклад можна привести нову розробку Ізраїльської компанії Elbit Systems – квадрокоптер THOR, який розвиває швидкість до 40 км на годину, має стійкість до спеки або холоду та максимальну висоту до 600 метрів. Акумулятора вистачає на 40 хвилин польоту [13].

Рисунок 3.1 – Квадрокоптер “THOR”

Безпілотний авіаційний комплекс – це безпілотний апарат або декілька, які керуються зі спеціально спорядженої станції [13].

Рисунок 3.2 – Вигляд станції керування БпЛА

Середовищем використання є військова галузь, в якій розвідка місцевості проводиться без ризиків втрат особового складу. Дані літальні апарати представляють собою дрони, які схожі на літаки. БпЛА можуть високо літати, набувати більшу швидкість ніж у квадрокоптера та можуть переносити біля 60 кілограм ваги. Логічним є те, що ціна даних літальних апаратів близько 70 млн доларів. Ціна – один з найголовніших мінусів цього безпілотника, та не єдиний. З інших мінусів можна зазначити: доступність, адже не кожен пересічний громадянин має 70 млн.; Горизонтальний зліт, що потребує рівну поверхню або апарат для запуску дронів; Відсутність можливості «зависнути» в повітрі [14]. Найпопулярнішим представником даного типу є турецький Bayraktar. Як класичний представник безпілотних авіаційних комплексів, має станцію управління та спеціально навчених операторів літальних об’єктів. Розробка для воєнних завдань, яка може переносити на собі зброю, бомби, ракети.

Рисунок 3.3 – БпЛА Bayraktar TB2 в польоті

Після ознайомлення з обома видами БпЛА, було обрано перший тип (Квадрокоптер). Головною причиною вибору є можливість вертикального зльоту, можливість зупинятись в повітрі та виконувати швидкі розвороти. Буде змодельовано квадрокоптер Thor від компанії Elbit. Вимоги до моделі: лопаті, які зможуть крутитись. Для цього їх треба зробити окремо від корпусу.

Моделювання починається зі створення корпусу коптера, який має особливу форму. Таким чином, треба підготувати малополігональну версію для подальшого збільшення полігонів. Корпус має форму розтягнутого куба, позаду якого кришка для доступу до батареї, модуля керування і т.д. Розділений він впродовж периметру посередині, для ремонту більш серйозних поломок. Створено куб та змінена довжина, після чого розрізано на дві частини шляхом виділення вершин та видалення за допомогою клавіші X, в контекстному меню обрано Vertices. Після видалення залишиться половина, після чого для зручності увімкнути модифікатор Mirror, який генерує дзеркальне відображення половини. Таким чином, можна зменшити собі роботу та зробити дрон більш симетричним. Використовуючи LoopCut, розрізано фігуру та із вершин сформований корпус.

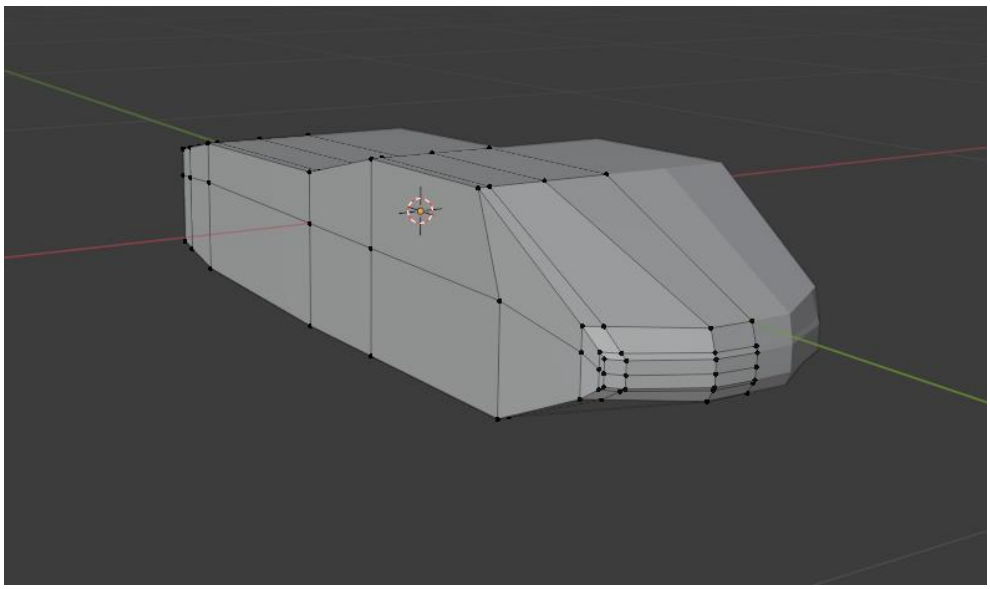


Рисунок 3.4 – Корпус літального апарату в режимі Edit Mode

На рисунку 3.4 видно гострі кути корпусу дрона, але на фото із реального життя корпус гладенький. Цю проблему можливо вирішити раніше вивченим інструментом Bevel, але в даному випадку граней багато. В такому випадку застосовано модифікатор Subdivision Surface. Його суть в збільшенні кількості полігонів та згладжуванні гострих кутів. У вікні модифікатора треба обрати Levels ViewPort, який відповідає за кількість додаткових полігонів та сгляжування. Між Catmull-Clark та Simple є одна відмінність, перший параметр множить полігони з їх подальшим округленням.

Після створення основи квадрокоптера за допомогою модифікатора Boolean реалізована виїмка від з’єднання верхньої та нижньої частини. Позаду є кілька полігонів, які можна екструдувати (Extrude) трішки всередину.

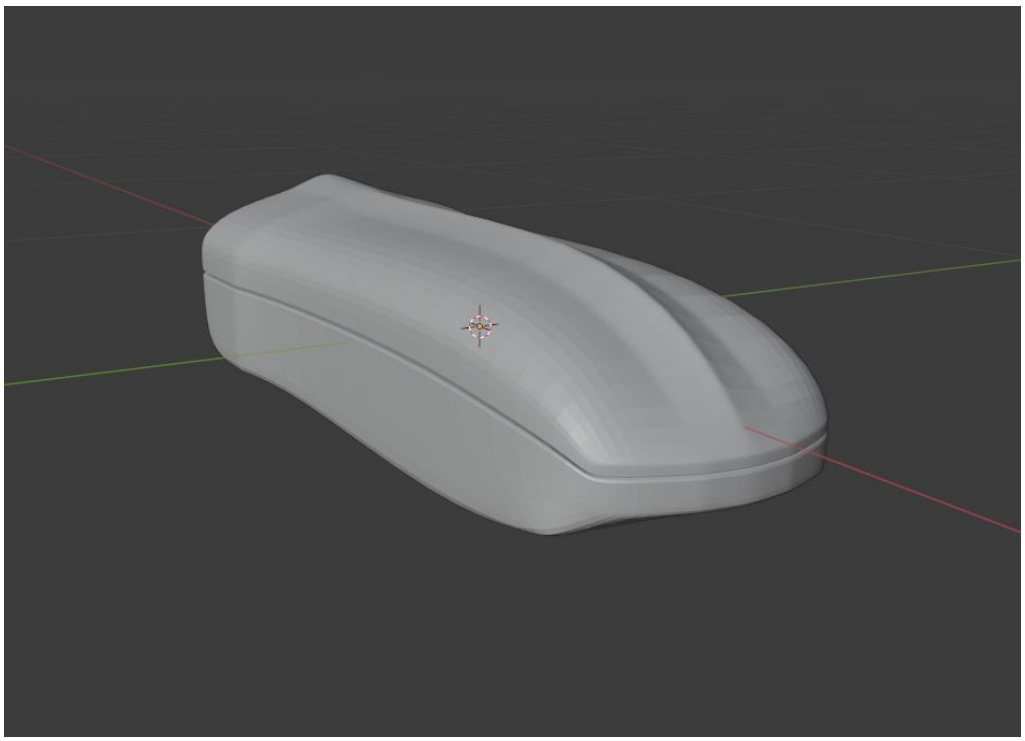


Рисунок 3.5 – Готовий корпус

Держаки для лопатей зроблено з декількох компонентів: місце для встановлення трубки, на якій розміщені лопаті; тримачі для підніжок; трубка; підставка під лопаті; лопаті. Щоб не моделювати кілька разів можна скопіювати (Shift + D)

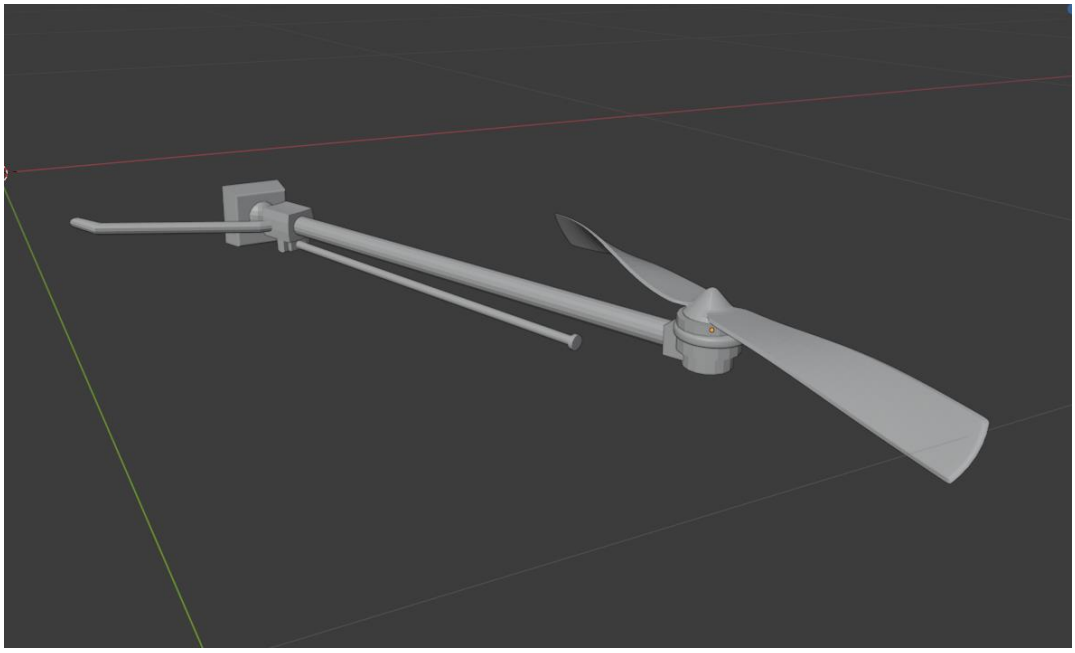


Рисунок 3.6 – Результат моделювання

Для створення лопатей як на рисунку 3.6, застосовано інструмент Proportional Editing. Він дає змогу змінювати геометрію об’єкту так, щоб не було різких переходів, гострих кутів. Щоб побачити результат треба налаштувати зону зміни, в якій буде змінюватись геометрія відносно обраних вершин.

Головною деталлю квадрокоптера є камера розташована знизу. При перегляді фотографій дрона THOR можна побачити багато варіантів. В даній моделі реалізована одна з них.

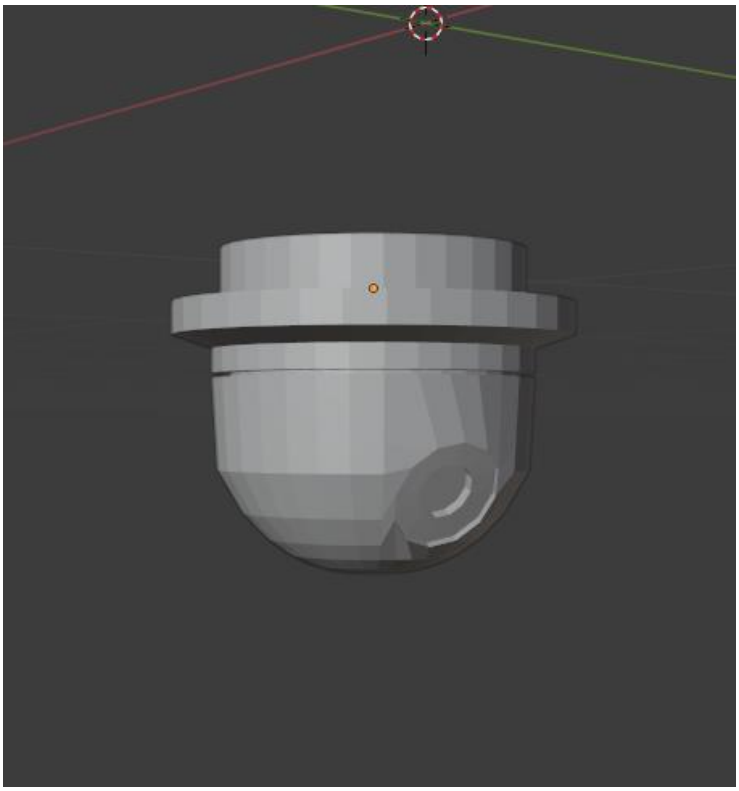


Рисунок 3.7 – Відеокамера квадрокоптера

Текстури можна підібрати з наявних кольорів в Blender. В момент, коли додається матеріал (описано в п. 2.3), то можна Base Color змінити на будь який колір.

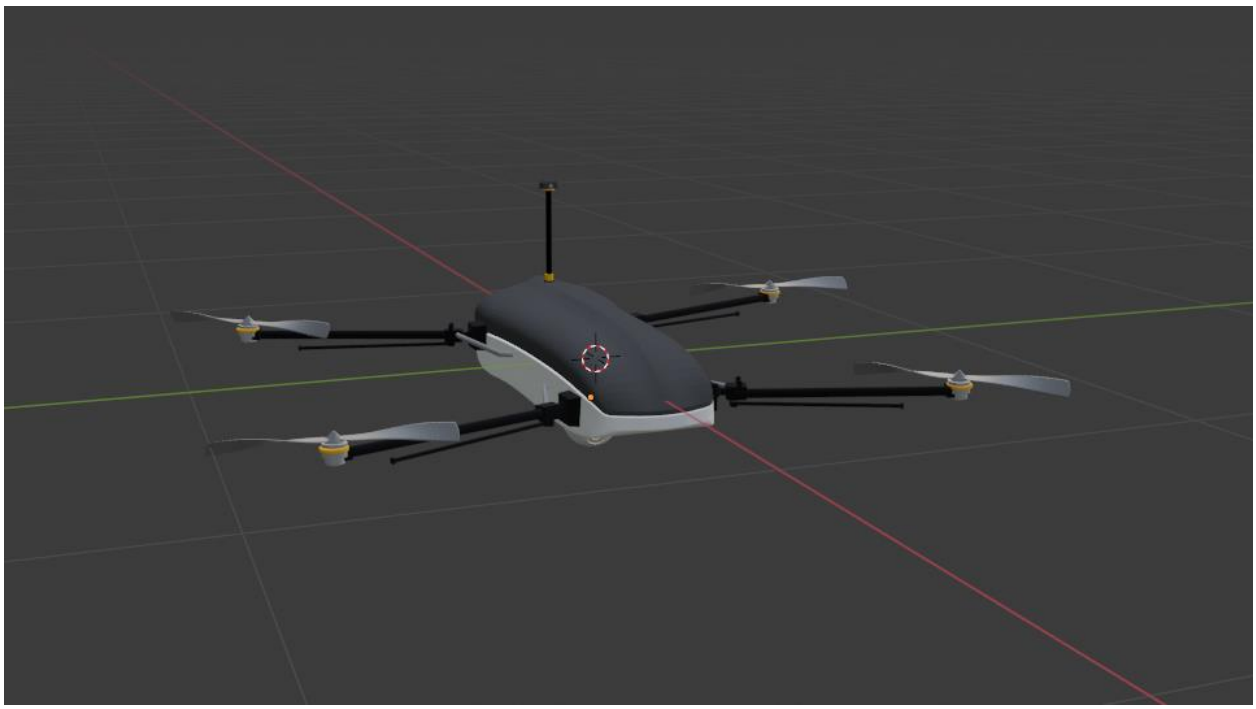
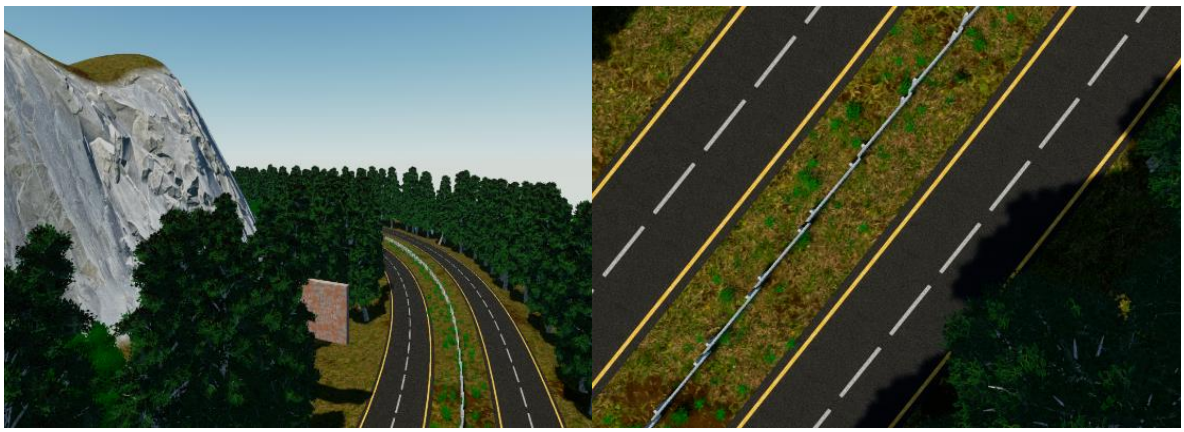


Рисунок 3.8 – Готова модель квадрокоптера

Перед експортом в Unreal Engine треба об’єднати всі елементи між собою, окрім лопатей. Треба виділити всі частини через Shift + лівою кнопкою миші по частині квадрокоптера. Після виділення через Ctrl + J виконано об’єднання. Таким чином, модель готова до імпорту в Unreal Engine з окремими лопатями.

3.2 Опис алгоритму безпілотного літального апарату

Головна задача, яка ставиться перед квадрокоптером, це спостереження за наземними об’єктами та розвідка території. Для виконання цієї задачі на борту безпілотного літального апарату(БПЛА) було встановлено дві камери. Зображення місцевості, які оператор отримує шляхом відеоспостереження за територією, показано на рисунку 3.9.



а *б*

Рисунок 3.9 – Вид з камер безпілотника: а – фронтова камера; б – робоча камера;

Дві камери необхідні для того, щоб за допомогою 3.9.а керувати безпілотником, в той час як 3.9.б дозволяє ідентифікувати місцевість. Слід зазначити, що в наземному центрі керування, який є частинною безпілотного авіаційного комплексу (БАК), окрім звичайної цифрової відеозйомки можуть ще аналізуватися результати, які були отримані у інфрочервоному спектрі або, наприклад, за допомогою електронно-оптичного перетворювача (ЕОП), проте це залежить від специфіки БПЛА. У рамках бакалаврської роботи були розглянуті лише зображення, які отримані за допомогою стандартних оптико-електронних пристроїв.

Однією із важливих особливостей сучасних відеокамер є цифрове та оптичне збільшення зображення. Для БАК це представляє особливу цінність, адже відомо, що безпілотники типу «квадракоптер» можуть літати на висоті понад 500 метрів, що, звісно ж, дає можливість огляду більш великої території, але її аналіз ускладнюється. У таких випадках вкрай важливо сфокусуватися на об’єкті інтересу, ігноруючи зайву інформацію. На рисунку 3.10 показано приклад цифрового збільшення зображення на борту БПЛА.

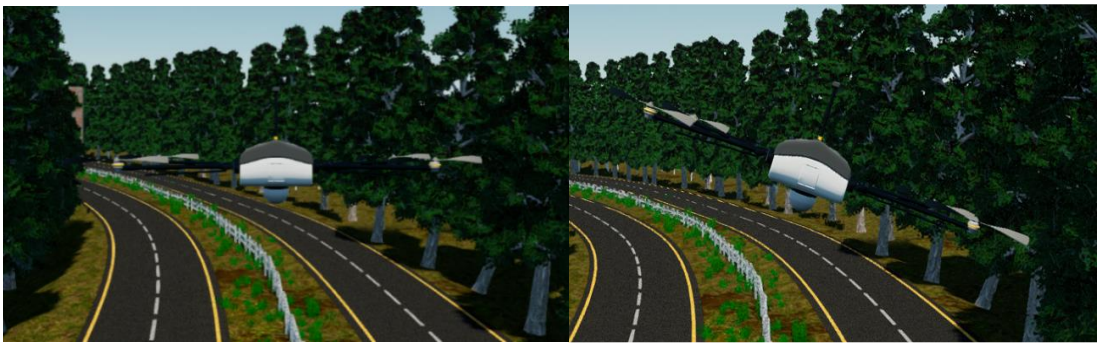
а *б*

Рисунок 3.10 – Процес цифрового збільшення зображення: а – початковий вид місцевості; б – приближений вид на об’єкт;

Реалізований БПЛА не є повністю автономним, його керування буде здійснюватися оператором в наземному пункті. Створена модель безпілотника здатна виконувати наступні дії: рух вперед і назад, поворот вліво і вправо відносно власної осі, підйом і спуск, рух вліво і вправо. Усі описані дії можна комбінувати, наприклад, підійматися по діагоналі. Змодельований квадрокоптер рухається приблизно зі швидкістю 40 км/г, що наближає його до реального прототипу.

Ключовою особливістю, яка виділяє квадракоптери з поміж інших безпілотних літальних апаратів, є їх маневреність та здатність зависати у повітрі. Це дозволяє їм виконувати конвоювання і спостереження за малогабаритними об’єктами. Саме на ці фактори було зроблено акцент під час моделювання фізики руху, яка повинна була бути максимально достовірна.

Важливим моментом є те, що квадракоптер не може рухатися чітко прямо, на нього впливають зовнішні чинники. Щоб це врахувати для створеної моделі окремо обраховується кут нахилу БПЛА, який він приймає при переміщенні.



а *б*

Рисунок 3.11 – Приклад нахилу БПЛА: а – початкове положення; б – нахил під час польоту на право;

Кут нахилу, приклад якого показано на рисунку 3.11, становить від 10 до 20 градусів. Квадракоптер його приймає при переміщенні в будь-якому із напрямків.

Для ще більшої достовірності безпілотник летить по інерції якщо відпустити кнопку польоту. Наприклад, якщо квадракоптер, який рухається вперед зі швидкістю 37 км/г, зупиниться, то він ще на деякий час продовжить свій рух. Це виконується для будь-якого переміщення БПЛА, і ще більше наближає його до реального прототипу.

Щоб зменшити кількість пам’яті, яку буде використовувати бортова система розпізнавання (БСР), було прийнято рішення розбивати відео потік, який поступає з відеокамери (3.9.б) на фрейми і класифікувати їх. Результатом такого підходу є рисунок 3.12, де цілісне відео «розрізане» на частини і склеєне в одне зображення.

Рисунок 3.12 – Приклад фреймів розбитого відео-потоку

3.3 Короткий опис програмного забезпечення

У таблиці 3.1 наведені назви і опис функцій, які були створені для реалізації основних фізичних властивостей безпілотника. Функції з переміщенням дрону створено за допомогою технології візуального програмування Blueprint. Основний код, конвертований автоматично з Blueprint, на мові програмування C++, знаходиться в додатку.

Таблиця 3.1 – Опис функцій бортової системи БПЛА

Методи	
Назва	Призначення
CameraZoom	Цифрове збільшення зображення отриманого з камери.
MovementInput	Здійснення руху безпілотника вперед і назад.
Up\Down	Переміщення безпілотника вгору та вниз
Rotation (Left\Right)	Поворот квадракоптеру, відносно своєї осі, за допомогою клавіатури
MouseRotation (Left\Right)	Поворот квадракоптеру, відносно своєї осі, за допомогою комп’ютерної миші
SwapCamers	Зміна між камерами безпілотника
VintsRotation	Виконання обертань гвинтів квадракоптеру
Keys	Запам’ятовування клавіш, які зараз натиснуті користувачем
PressKey	Перевірка, які із клавіш натиснуті зараз
AutoStabilization	Автостабілізація безпілотника, коли ним ніхто не керує
Lean (Right\Left)	Нахил безпілотника, коли він рухається вліво чи вправо

Lean (Forward\Back)	Нахил безпілотника, коли він рухається вперед чи назад
Hit	Перевірка на зіштовхнення БПЛА з іншим об’єктом

3.4 Результати моделювання

Нижче приведені результати по розробці об’єктів в співвідношенні до категорій:

Споруди:

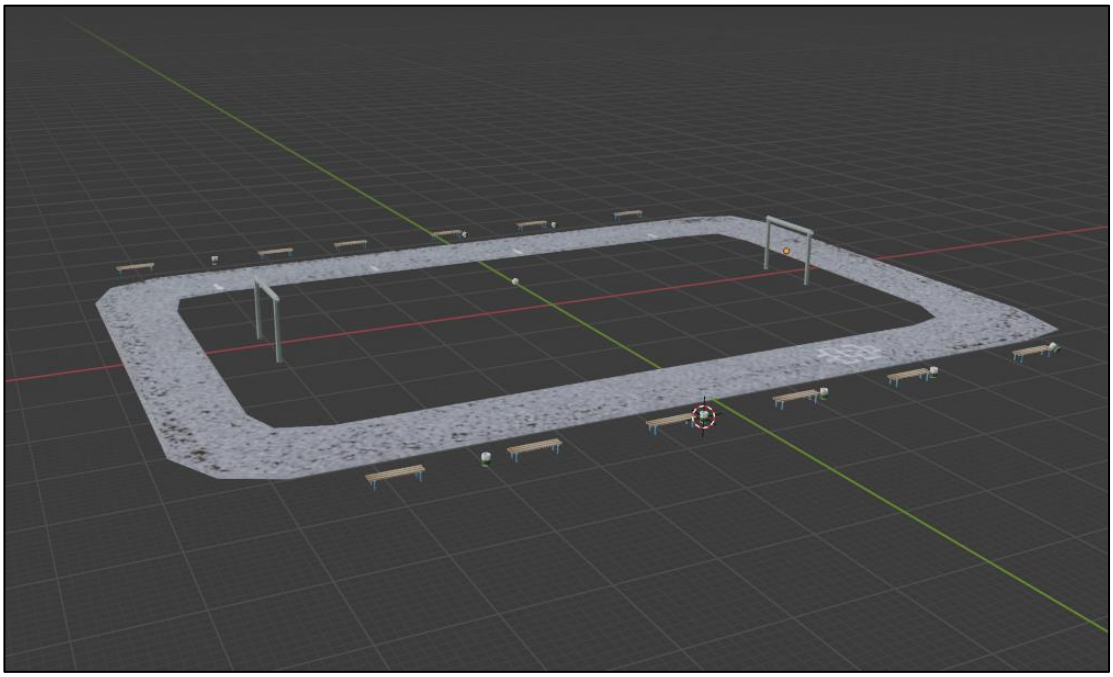


Рисунок 3.5. Стадіон

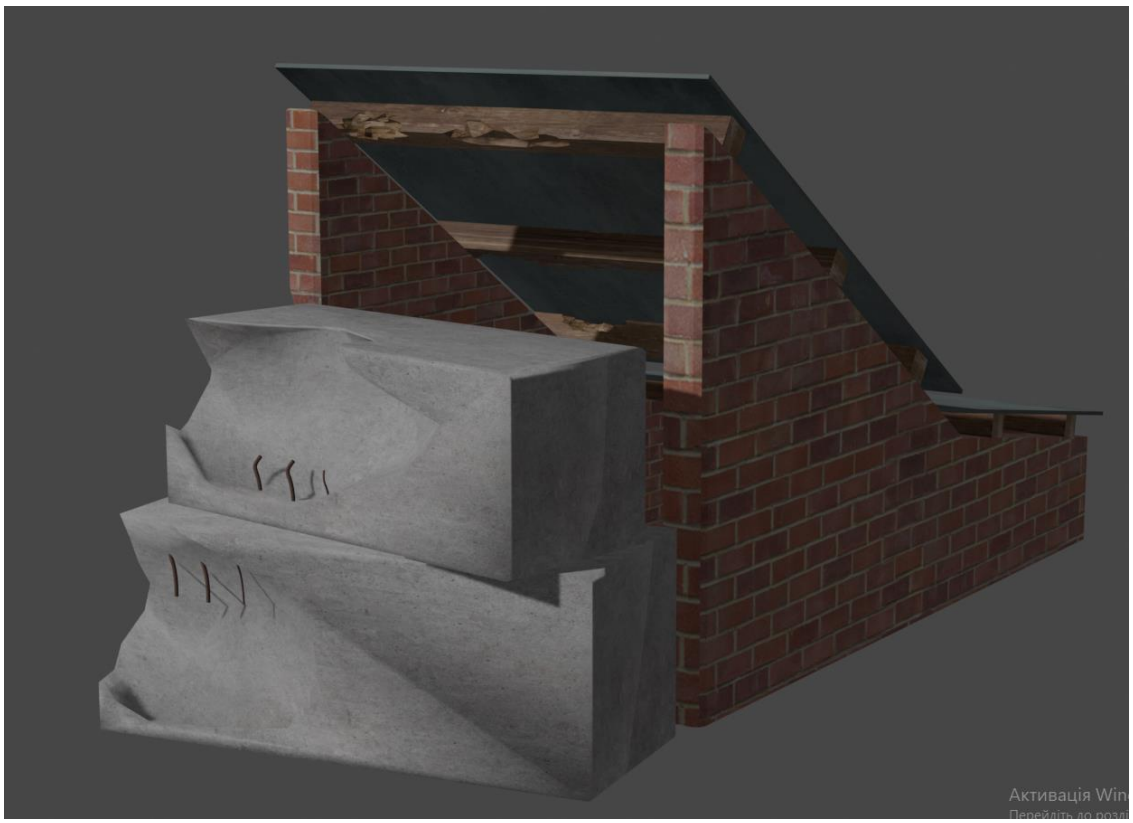


Рисунок 3.6. Підвал

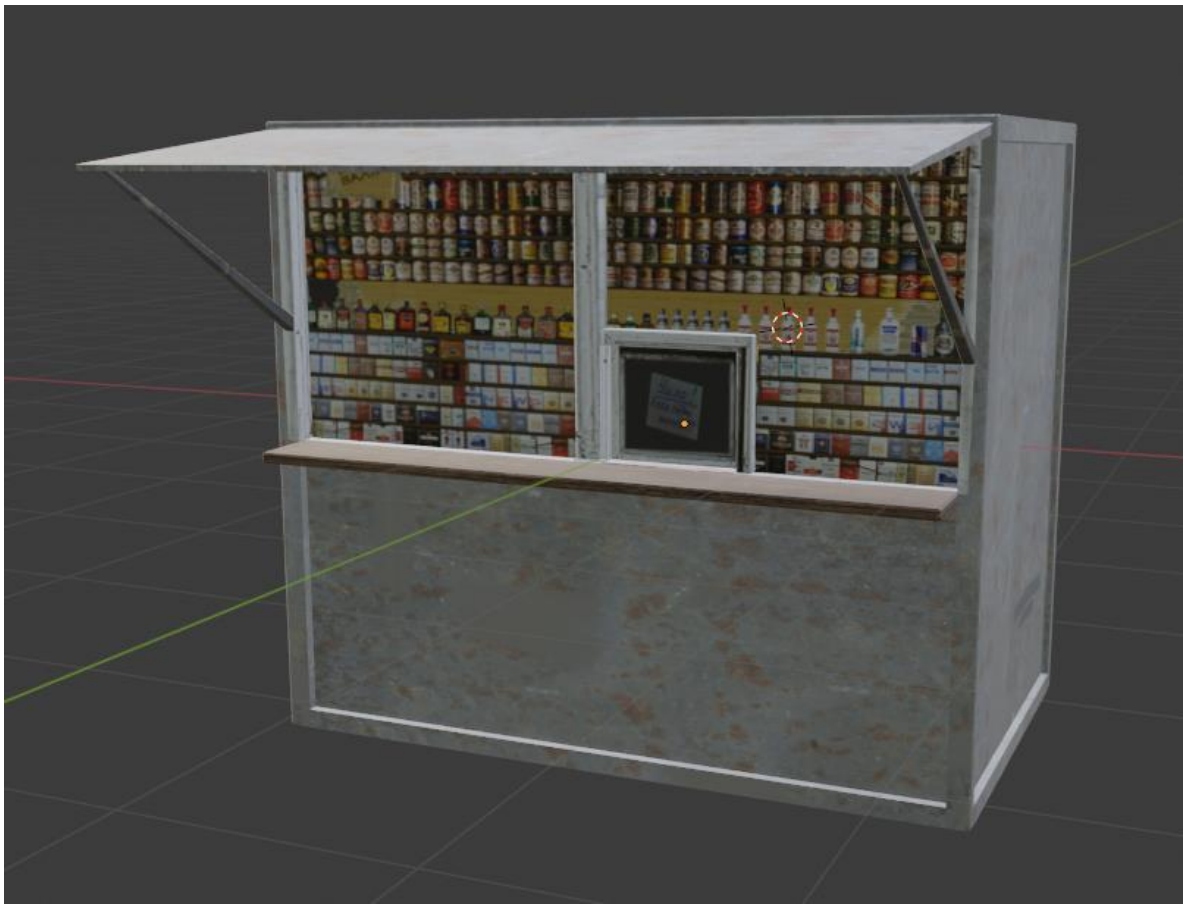


Рисунок 3.7. Торговий ларьок

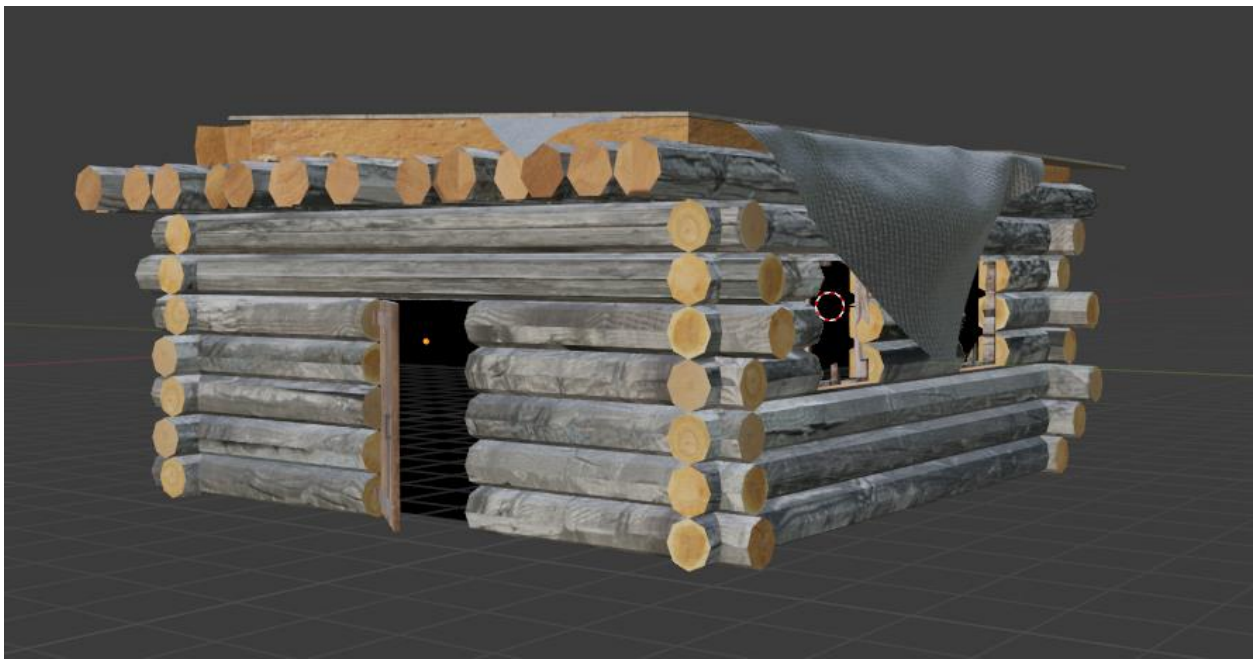


Рисунок 3.8. Закинутий дім в селі

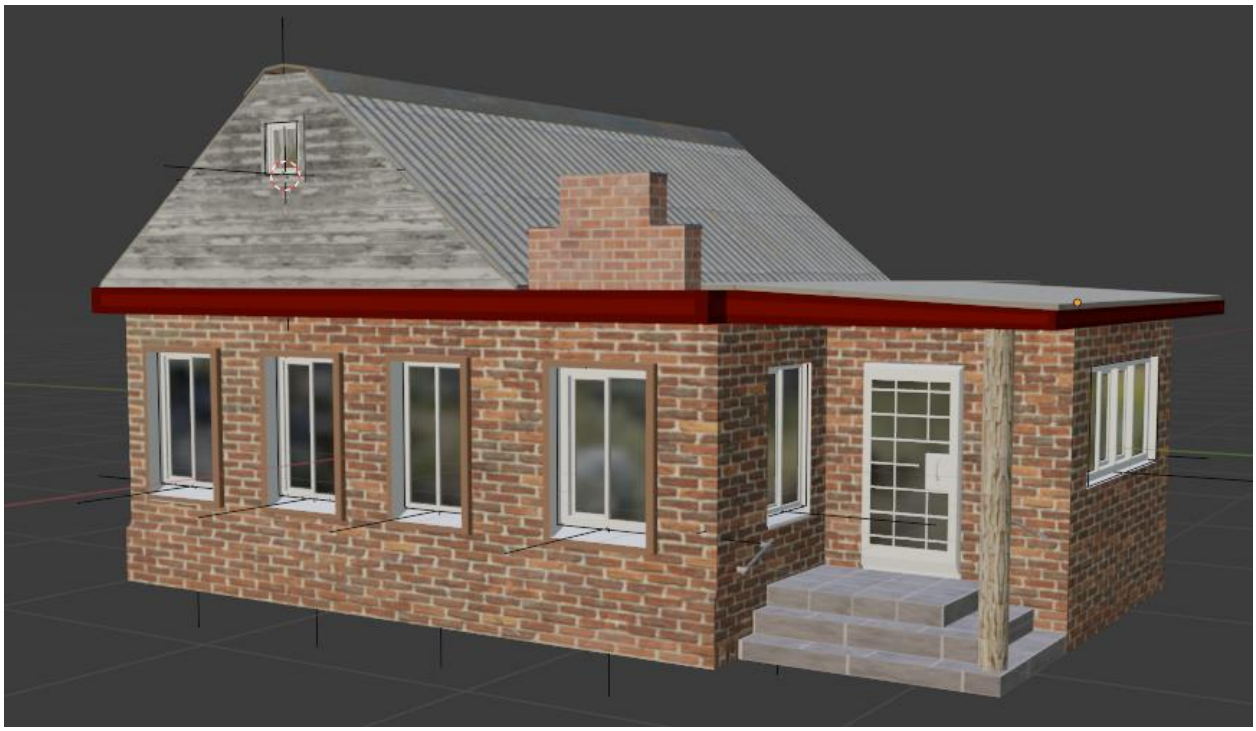


Рисунок 3.9. Дім в селі

Рисунок 3.10. Гараж для автомобіля

Рисунок 3.11. Склад для дров

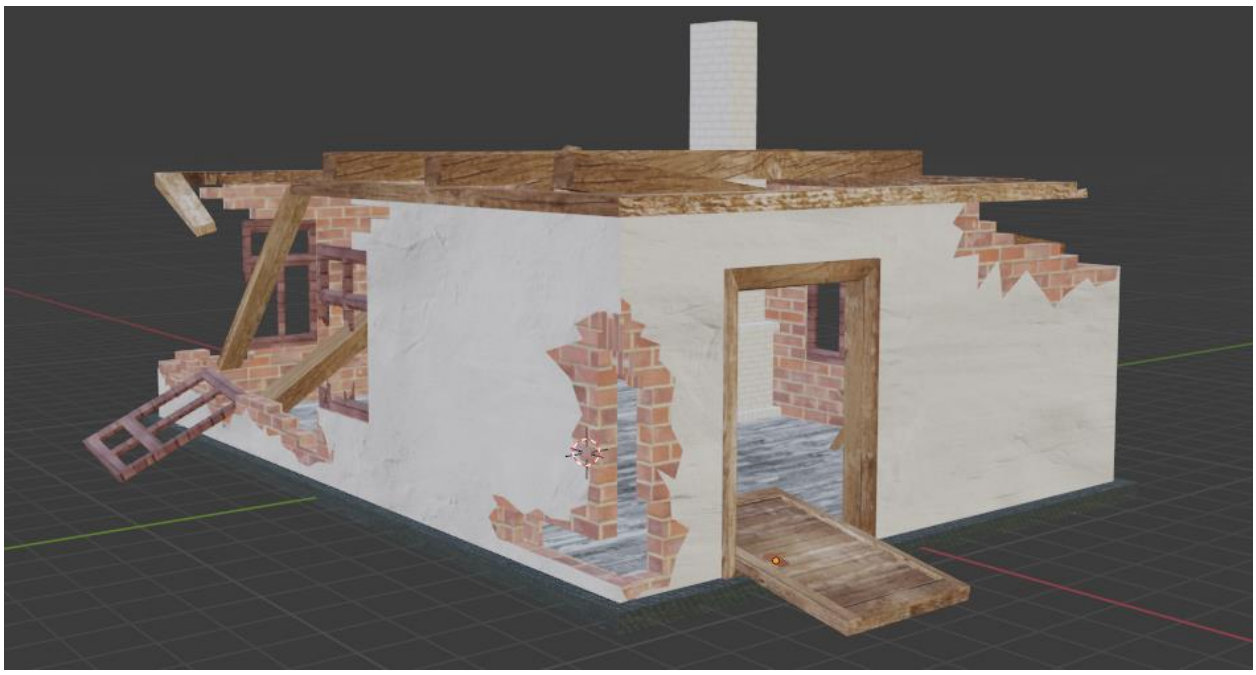


Рисунок 3.12. Знищена хатина

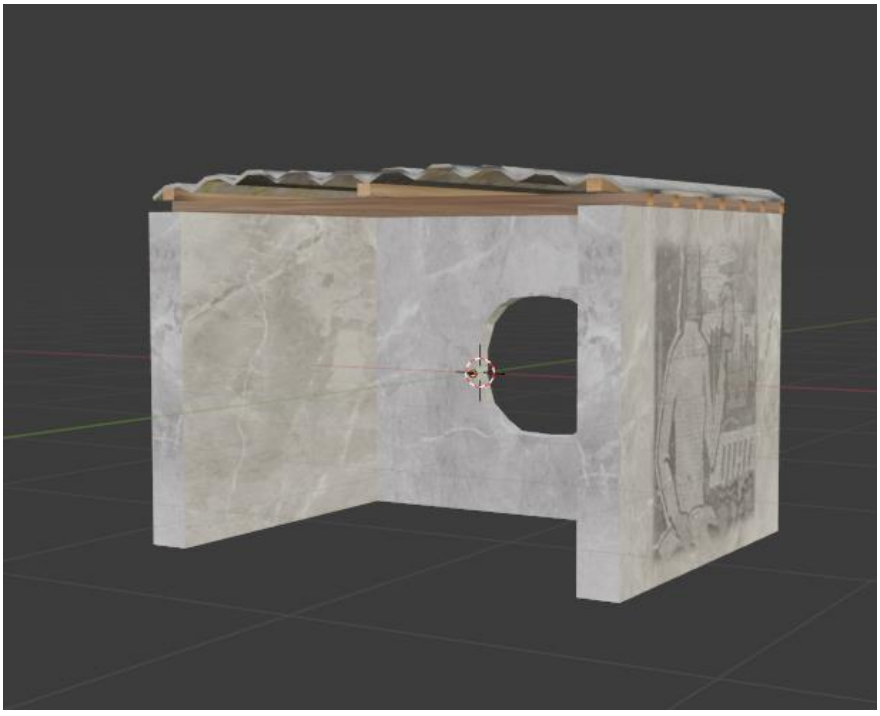


Рисунок 3.13. Автобусна зупинка

Рисунок 3.14. Заправка

Рисунок 3.15. Споруда з дерева

Рисунок 3.16 – Вагончик на колесах

ВИСНОВКИ

Протягом роботи було досліджено поняття геоінформаційної (ГіС) системи, її складової. Визначені вимоги до ландшафту, інфраструктурних об'єктів та природних насаджень.

Проведено порівняння інструментів реалізації ГіС, після чого обрано програми для реалізації ландшафту, об’єктів та створення текстур.

Створено квадрокоптер з попереднім аналізом типів та моделей сучасних апаратів. Враховано критерії для імпорту в Unreal Engine.

Головною перевагою побудованої 3д-моделі місцевості - є її повна достовірність реальним об'єктам, які характерні для території України. Під час передипломної практики проведено аналіз місцевості України, визначено

характерні риси сільських та природних об'єктів. На їх основі змодельовано інфраструктурні об'єкти.

У рамках бакалаврської роботи особливу увагу приділялося техногенним об'єктам, які дуже часто є зоною інтересу для безпілотного літального апарату. Розроблено дорогу двох видів (камінну та асфальтну) та густий ліс.

Однією із типових задач, яка ставиться до безпілотника, є конвоювання транспортного засобу. Для відтворення подібної ситуації було змодельована неперервна інфраструктурна мережа типу: дорога асфальтна, дорога ґрунтова.

СПИСОК ЛІТЕРАТУРИ

1. https://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0
2. https://sites.google.com/site/gisprogrammy/prostranstvennye-dannye
3. https://www.solidworks.com/ru/product/solidworks-3d-cad
4. https://uk.wikipedia.org/wiki/Autodesk_3ds_MAX
5. https://www.autodesk.ru/products/3ds-max/overview?term=1-YEAR
6. https://www.blender.org/features/
7. https://store.steampowered.com/app/365670/Blender/?l=russian
8. https://uk.wikipedia.org/wiki/Unity_(%D1%80%D1%83%D1%88%D1%96%D0%B9_%D0%B3%D1%80%D0%B8)
9. https://uk.wikipedia.org/wiki/Unreal_Engine
10. http://www.evolved-software.com/treeit/treeit
11. https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%BF%D1%96%D0%BB%D0%BE%D1%82%D0%BD%D0%B8%D0%B9_%D0%BB%D1%96%D1%82%D0%B0%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BF%D0%B0%D1%80%D0%B0%D1%82
12. https://uk.wikipedia.org/wiki/%D0%9C%D1%83%D0%BB%D1%8C%D1%82%D0%B8%D0%BA%D0%BE%D0%BF%D1%82%D0%B5%D1%80
13. https://elbitsystems.com/product/thor/
14. https://uk.wikipedia.org/wiki/Bayraktar_TB2

ДОДАТОК

```
#include "Runtime/Engine/Classes/Components/SceneCaptureComponent2D.h"
#include "Runtime/Engine/Classes/Components/SceneCaptureComponent.h"
#include "Runtime/Engine/Classes/Engine/TextureRenderTarget2D.h"
#include "Runtime/Engine/Classes/Engine/TextureRenderTarget.h"
#include "Runtime/Engine/Classes/Kismet/KismetMathLibrary.h"
#include "Runtime/Engine/Classes/Kismet/BlueprintFunctionLibrary.h"
#include "Runtime/Engine/Classes/Kismet/GameplayStatics.h"
#include "Runtime/Engine/Classes/Components/AudioComponent.h"
#include "Runtime/AudioMixer/Public/Quartz/AudioMixerClockHandle.h"
#include "Runtime/Engine/Classes/Sound/QuartzQuantizationUtilities.h"
#include "Runtime/Engine/Classes/GameFramework/ForceFeedbackAttenuation.h"
#include "Runtime/Engine/Classes/Components/ForceFeedbackComponent.h"
#include "Runtime/Engine/Classes/Sound/DialogueWave.h"
#include "Runtime/Engine/Classes/Sound/DialogueTypes.h"
#include "Runtime/Engine/Classes/Sound/DialogueVoice.h"
#include "Runtime/Engine/Classes/Sound/DialogueSoundWaveProxy.h"
#include "Runtime/Engine/Classes/Components/DecalComponent.h"
#include "Runtime/Engine/Classes/GameFramework/SaveGame.h"
#include "Runtime/Engine/Classes/Kismet/GameplayStaticsTypes.h"
#include "Runtime/Engine/Classes/Kismet/KismetSystemLibrary.h"
#include "Runtime/Engine/Classes/Engine/CollisionProfile.h"

void ADroneThor_C__pf332852796::PostLoadSubobjects(FObjectInstancingGraph*
OuterInstanceGraph)
{
	Super::PostLoadSubobjects(OuterInstanceGraph);
	if(bpv__FloatingPawnMovement__pf)
	{
		bpv__FloatingPawnMovement__pf->CreationMethod =
EComponentCreationMethod::Native;
	}
	if(bpv__BoxCollision__pf)
	{
		bpv__BoxCollision__pf->CreationMethod = EComponentCreationMethod::Native;
	}
	if(bpv__SceneCaptureComponent2D__pf)
	{
	bpv__SceneCaptureComponent2D__pf->CreationMethod = EComponentCreationMethod::Native;
	}
	if(bpv__Camera1__pf)
	{
		bpv__Camera1__pf->CreationMethod = EComponentCreationMethod::Native;
	}
	if(bpv__body__pf)
	{
		bpv__body__pf->CreationMethod = EComponentCreationMethod::Native;
	}
	if(bpv__vint__pf)
	{
		bpv__vint__pf->CreationMethod = EComponentCreationMethod::Native;
	}
	if(bpv__vint1__pf)
	{
		bpv__vint1__pf->CreationMethod = EComponentCreationMethod::Native;
	}
	if(bpv__vint2__pf)
	{
		bpv__vint2__pf->CreationMethod = EComponentCreationMethod::Native;
	}
	if(bpv__vint3__pf)
	{
		bpv__vint3__pf->CreationMethod = EComponentCreationMethod::Native;
	}
```

```
	if(bpv__Camera__pf)
	{
		bpv__Camera__pf->CreationMethod = EComponentCreationMethod::Native;
	}
}


void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_1(int32
bpp__EntryPoint__pf)
{
	bool bpfv__CallFunc_Not_PreBool_ReturnValue__pf{};
	int32 __CurrentState = bpp__EntryPoint__pf;
	do
	{
		switch( __CurrentState )
		{
		case 1:
			{
				bpv__Camera2_Active__pf = false;
				__CurrentState = -1;
				break;
			}
		case 117:
			{
			}
		case 118:
			{
			}
		case 119:
			{
			}
		case 120:
			{
				bpfv__CallFunc_Not_PreBool_ReturnValue__pf =
UKismetMathLibrary::Not_PreBool(b0l__Temp_bool_Variable__pf);
				b0l__Temp_bool_Variable__pf =
bpfv__CallFunc_Not_PreBool_ReturnValue__pf;
			}
		case 121:
			{
				if (!b0l__Temp_bool_Variable__pf)
				{
					__CurrentState = 126;
					break;
				}
			}
		case 122:
			{
			}
		case 123:
			{
				if(::IsValid(bpv__Camera__pf))
				{
					bpv__Camera__pf->Deactivate();
				}
			}
		case 124:
			{
				if(::IsValid(bpv__Camera1__pf))
				{
					bpv__Camera1__pf->Activate(false);
				}
			}
		case 125:
			{
```

```
                        bpv__Camera2_Active__pf = true;
                        __CurrentState = -1;
                        break;
                }
        case 126:
                {
                }
        case 127:
                {
                        if(::IsValid(bpv__Camera__pf))
                        {
                                bpv__Camera__pf->Activate(false);
                        }
                }
        case 128:
                {
                        if(::IsValid(bpv__Camera1__pf))
                        {
                                bpv__Camera1__pf->Deactivate();
                        }
                        __CurrentState = 1;
                        break;
                }
        default:
                break;
        }
    } while( __CurrentState != -1 );
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_3(int32
bpp__EntryPoint__pf)
{
    check(bpp__EntryPoint__pf == 220);
    // optimized KCST_UnconditionalGoto
    UKismetSystemLibrary::PrintString(this, FString(TEXT("Hit")), true, true,
FLinearColor(0.000000,0.660000,1.000000,1.000000), 2.000000);
    return; //KCST_EndOfThread
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_4(int32
bpp__EntryPoint__pf)
{
    FRotator bpfv__CallFunc_GetControlRotation_ReturnValue__pf(EForceInit::ForceInit);
    FRotator bpfv__CallFunc_MakeRotator_ReturnValue__pf(EForceInit::ForceInit);
    FVector bpfv__CallFunc_GetRightVector_ReturnValue__pf(EForceInit::ForceInit);
    check(bpp__EntryPoint__pf == 218);
    bpv__MoveRight__pf = b0l__K2Node_InputAxisEvent_AxisValue__pf;
    // optimized KCST_UnconditionalGoto
    bpfv__CallFunc_GetControlRotation_ReturnValue__pf = APawn::GetControlRotation();
    UKismetMathLibrary::BreakRotator(bpfv__CallFunc_GetControlRotation_ReturnValue__pf,
/*out*/ b0l__CallFunc_BreakRotator_Roll__pf, /*out*/ b0l__CallFunc_BreakRotator_Pitch__pf,
/*out*/ b0l__CallFunc_BreakRotator_Yaw__pf);
    bpfv__CallFunc_MakeRotator_ReturnValue__pf =
UKismetMathLibrary::MakeRotator(0.000000, 0.000000, b0l__CallFunc_BreakRotator_Yaw__pf);
    bpfv__CallFunc_GetRightVector_ReturnValue__pf =
UKismetMathLibrary::GetRightVector(bpfv__CallFunc_MakeRotator_ReturnValue__pf);
    AddMovementInput(bpfv__CallFunc_GetRightVector_ReturnValue__pf, bpv__MoveRight__pf,
false);
    return; //KCST_EndOfThread
}


void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_5(int32
bpp__EntryPoint__pf)
{
```

```
	FRotator bpfv__CallFunc_GetControlRotation_ReturnValue__pf(EForceInit::ForceInit);
	FRotator bpfv__CallFunc_MakeRotator_ReturnValue__pf(EForceInit::ForceInit);
	FVector bpfv__CallFunc_GetForwardVector_ReturnValue__pf(EForceInit::ForceInit);
	check(bpp__EntryPoint__pf == 216);
	bpv__MoveForward__pf = b0l__K2Node_InputAxisEvent_AxisValue_1__pf;
	// optimized KCST_UnconditionalGoto
	bpfv__CallFunc_GetControlRotation_ReturnValue__pf = APawn::GetControlRotation();
	UKismetMathLibrary::BreakRotator(bpfv__CallFunc_GetControlRotation_ReturnValue__pf,
/*out*/ b0l__CallFunc_BreakRotator_Roll__pf, /*out*/ b0l__CallFunc_BreakRotator_Pitch__pf,
/*out*/ b0l__CallFunc_BreakRotator_Yaw__pf);
	bpfv__CallFunc_MakeRotator_ReturnValue__pf =
UKismetMathLibrary::MakeRotator(0.000000, 0.000000, b0l__CallFunc_BreakRotator_Yaw__pf);
	bpfv__CallFunc_GetForwardVector_ReturnValue__pf =
UKismetMathLibrary::GetForwardVector(bpfv__CallFunc_MakeRotator_ReturnValue__pf);
	AddMovementInput(bpfv__CallFunc_GetForwardVector_ReturnValue__pf,
bpv__MoveForward__pf, false);
	return; //KCST_EndOfThread
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_6(int32
bpp__EntryPoint__pf)
{
	FRotator bpfv__CallFunc_GetControlRotation_ReturnValue_1__pf(EForceInit::ForceInit);
	FRotator bpfv__CallFunc_MakeRotator_ReturnValue_1__pf(EForceInit::ForceInit);
	FVector bpfv__CallFunc_GetUpVector_ReturnValue__pf(EForceInit::ForceInit);
	check(bpp__EntryPoint__pf == 214);
	bpv__UpxDown__pfWy = b0l__K2Node_InputAxisEvent_AxisValue_2__pf;
	// optimized KCST_UnconditionalGoto
	bpfv__CallFunc_GetControlRotation_ReturnValue_1__pf = APawn::GetControlRotation();
	UKismetMathLibrary::BreakRotator(bpfv__CallFunc_GetControlRotation_ReturnValue_1__pf,
/*out*/ b0l__CallFunc_BreakRotator_Roll_1__pf, /*out*/
b0l__CallFunc_BreakRotator_Pitch_1__pf, /*out*/ b0l__CallFunc_BreakRotator_Yaw_1__pf);
	bpfv__CallFunc_MakeRotator_ReturnValue_1__pf =
UKismetMathLibrary::MakeRotator(0.000000, 0.000000, b0l__CallFunc_BreakRotator_Yaw_1__pf);
	bpfv__CallFunc_GetUpVector_ReturnValue__pf =
UKismetMathLibrary::GetUpVector(bpfv__CallFunc_MakeRotator_ReturnValue_1__pf);
	AddMovementInput(bpfv__CallFunc_GetUpVector_ReturnValue__pf, bpv__UpxDown__pfWy,
false);
	return; //KCST_EndOfThread
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_13(int32
bpp__EntryPoint__pf)
{
	int32 __CurrentState = bpp__EntryPoint__pf;
	do
	{
		switch( __CurrentState )
		{
		case 132:
			{
				if (!bpv__E__pf)
				{
					__CurrentState = 133;
					break;
				}
				__CurrentState = -1;
				break;
			}
		case 133:
			{
				bpv__QxE_Speed__pfWy = 0.000000;
				__CurrentState = -1;
				break;
			}
```

```
		case 201:
			{
			}
		case 202:
			{
				b0l__Temp_struct_Variable_13__pf =
b0l__K2Node_InputKeyEvent_Key_27__pf;
				__CurrentState = 132;
				break;
			}
		default:
			break;
		}
	} while( __CurrentState != -1 );
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_14(int32
bpp__EntryPoint__pf)
{
	check(bpp__EntryPoint__pf == 199);
	b0l__Temp_struct_Variable_13__pf = b0l__K2Node_InputKeyEvent_Key_26__pf;
	// optimized KCST_UnconditionalGoto
	bpv__QxE_Speed__pfWy = -1.000000;
	return; //KCST_EndOfThread
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_17(int32
bpp__EntryPoint__pf)
{
	FRotator
bpfv__CallFunc_K2_GetActorRotation_ReturnValue_13__pf(EForceInit::ForceInit);
	FRotator
bpfv__CallFunc_K2_GetActorRotation_ReturnValue_14__pf(EForceInit::ForceInit);
	FRotator bpfv__CallFunc_MakeRotator_ReturnValue_11__pf(EForceInit::ForceInit);
	FRotator bpfv__CallFunc_RLerp_ReturnValue_8__pf(EForceInit::ForceInit);
	bool bpfv__CallFunc_K2_SetActorRotation_ReturnValue_9__pf{};
	check(bpp__EntryPoint__pf == 134);
	bpfv__CallFunc_K2_GetActorRotation_ReturnValue_13__pf =
AActor::K2_GetActorRotation();
	bpfv__CallFunc_K2_GetActorRotation_ReturnValue_14__pf =
AActor::K2_GetActorRotation();
	UKismetMathLibrary::BreakRotator(bpfv__CallFunc_K2_GetActorRotation_ReturnValue_13__p
f, /*out*/ b0l__CallFunc_BreakRotator_Roll_11__pf, /*out*/
b0l__CallFunc_BreakRotator_Pitch_11__pf, /*out*/ b0l__CallFunc_BreakRotator_Yaw_11__pf);
	bpfv__CallFunc_MakeRotator_ReturnValue_11__pf =
UKismetMathLibrary::MakeRotator(0.000000, 0.000000, b0l__CallFunc_BreakRotator_Yaw_11__pf);
	bpfv__CallFunc_RLerp_ReturnValue_8__pf =
UKismetMathLibrary::RLerp(bpfv__CallFunc_K2_GetActorRotation_ReturnValue_14__pf,
bpfv__CallFunc_MakeRotator_ReturnValue_11__pf,
bpv__Timeline_3_Alpha_6BD90F43451E2F64B75C368EDA890558__pf, false);
	bpfv__CallFunc_K2_SetActorRotation_ReturnValue_9__pf =
AActor::K2_SetActorRotation(bpfv__CallFunc_RLerp_ReturnValue_8__pf, false);
	return; //KCST_EndOfThread
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_18(int32
bpp__EntryPoint__pf)
{
	int32 __CurrentState = bpp__EntryPoint__pf;
	do
	{
		switch( __CurrentState )
		{
		case 137:
			{
```

```
                        bpv__QxE_Speed__pfWy = 0.000000;
                        __CurrentState = -1;
                        break;
                }
        case 138:
                {
                        if (!bpv__Q__pf)
                        {
                                __CurrentState = 137;
                                break;
                        }
                        __CurrentState = -1;
                        break;
                }
        case 184:
                {
                }
        case 185:
                {
                        b0l__Temp_struct_Variable_12__pf =
b0l__K2Node_InputKeyEvent_Key_25__pf;
                        __CurrentState = 138;
                        break;
                }
        default:
                break;
        }
    } while( __CurrentState != -1 );
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_25(int32
bpp__EntryPoint__pf)
{
    TArray< int32, TInlineAllocator<8> > __StateStack;

    int32 __CurrentState = bpp__EntryPoint__pf;
    do
    {
        switch( __CurrentState )
        {
        case 153:
                {
                        if(::IsValid(bpv__Timeline_1__pf))
                        {
                                bpv__Timeline_1__pf->UTimelineComponent::Stop();
                        }
                        __CurrentState = (__StateStack.Num() > 0) ?
__StateStack.Pop(/*bAllowShrinking=*/ false) : -1;
                        break;
                }
        case 166:
                {
                }
        case 167:
                {
                        __StateStack.Push(168);
                        __CurrentState = 169;
                        break;
                }
        case 168:
                {
                        __CurrentState = 153;
                        break;
                }
        case 169:
```

```
                        {
                                if(::IsValid(bpv__Timeline_0__pf))
                                {
                                        bpv__Timeline_0__pf->UTimelineComponent::Stop();
                                }
                                __CurrentState = (__StateStack.Num() > 0) ?
__StateStack.Pop(/*bAllowShrinking=*/ false) : -1;
                                break;
                        }
                default:
                        check(false); // Invalid state
                        break;
                }
        } while( __CurrentState != -1 );
}

void ADroneThor_C__pf332852796::bpf__ExecuteUbergraph_DroneThor__pf_28(int32
bpp__EntryPoint__pf)
{
        check(bpp__EntryPoint__pf == 85);
        b0l__Temp_struct_Variable_7__pf = b0l__K2Node_InputKeyEvent_Key_9__pf;
        bpv__LeftShifht__pf = false;
        return; //KCST_EndOfThread
}

void
ADroneThor_C__pf332852796::bpf__BndEvt__DroneThor_BoxCollision_K2Node_ComponentBoundEvent_1_
ComponentHitSignature__DelegateSignature__pf(UPrimitiveComponent* bpp__HitComponent__pf,
AActor* bpp__OtherActor__pf, UPrimitiveComponent* bpp__OtherComp__pf, FVector
bpp__NormalImpulse__pf, FHitResult const& bpp__Hit__pf__const)
{
        typedef FHitResult  T__Local__130;
        T__Local__130& bpp__Hit__pf = *const_cast<T__Local__130 *>(&bpp__Hit__pf__const);
        b0l__K2Node_ComponentBoundEvent_HitComponent__pf = bpp__HitComponent__pf;
        b0l__K2Node_ComponentBoundEvent_OtherActor__pf = bpp__OtherActor__pf;
        b0l__K2Node_ComponentBoundEvent_OtherComp__pf = bpp__OtherComp__pf;
        b0l__K2Node_ComponentBoundEvent_NormalImpulse__pf = bpp__NormalImpulse__pf;
        b0l__K2Node_ComponentBoundEvent_Hit__pf = bpp__Hit__pf;
        bpf__ExecuteUbergraph_DroneThor__pf_3(220);
}

void
ADroneThor_C__pf332852796::__StaticDependencies_DirectlyUsedAssets(TArray<FBlueprintDependen
cyData>& AssetsToLoad)
{
        const FCompactBlueprintDependencyData LocCompactBlueprintDependencyData[] =
        {
                {0, FBlueprintDependencyType(false, true, false, false),
FBlueprintDependencyType(false, false, false, false)},  //  StaticMesh /Game/TARAS/body.body
                {1, FBlueprintDependencyType(false, true, false, false),
FBlueprintDependencyType(false, false, false, false)},  //  StaticMesh /Game/TARAS/vint.vint
        };
        for(const FCompactBlueprintDependencyData& CompactData :
LocCompactBlueprintDependencyData)
        {

        AssetsToLoad.Add(FBlueprintDependencyData(F__NativeDependencies::Get(CompactData.Obje
ctRefIndex), CompactData));
        }
}
```