**ДНІпровський національний університет**

**імені Олеся Гончара**

**ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ**

**КАФЕДРА КОМП’ЮТЕРНИХ ТЕХНОЛОГІЙ**

**КУРСОВА РОБОТА**

**ЗА ФАХОВИМ СПРЯМУВАННЯМ**

на тему: \_\_\_Моделювання і візуалізація роботи радарної системи\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Освітньо–професійна програма

Комп’ютерне моделювання та технології програмування

Спеціальність 113 Прикладна математика

Галузь знань 11 Математика і статистика

Рівень вищої освіти перший (бакалаврський)

Студента 4 курсу групи ПА-19-2

\_\_\_\_\_\_Ільяшенко Є.В.\_\_\_\_\_\_\_\_\_\_\_\_\_ (прізвище та ініціали)

Керівник \_\_Степанова Н.І.\_\_\_\_\_\_\_\_

\_\_\_доц., канд. фіз.-мат. наук\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів \_\_\_\_\_\_\_\_\_\_\_

Національна шкала \_\_\_\_\_\_\_

Члени комісії:

Зайцев В.Г.

(підпис) (прізвище та ініціали)

Степанова Н.І.

(підпис) (прізвище та ініціали)

Золотько К.Є.

(підпис) (прізвище та ініціали)

м. Дніпро, 2022 р.

**ЗМІСТ**

[**ВСТУП** 3](#_Toc122436077)

[**ПОСТАНОВКА ЗАДАЧІ** 6](#_Toc122436078)

[**1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ** 7](#_Toc122436079)

[1.1 Хвиля 7](#_Toc122436080)

[1.2 Різновиди хвиль 7](#_Toc122436081)

[1.3 Електромагнітне випромінювання 8](#_Toc122436082)

[1.4 Довжина хвилі 10](#_Toc122436083)

[1.5 Період хвилі 10](#_Toc122436084)

[1.6 Частота хвилі 11](#_Toc122436085)

[1.7 Фаза хвилі 11](#_Toc122436086)

[1.8 Математичний опис хвилі 11](#_Toc122436087)

[1.9 Інтерференція 12](#_Toc122436088)

[1.10 Зсув фаз у ФАР 13](#_Toc122436089)

[**2. Математична модель** 15](#_Toc122436090)

[3. **ПРОГРАМНА РЕАЛІЗАЦІЯ** 17](#_Toc122436091)

[**3.1 Обрання інструментів для програмної реалізації** 17](#_Toc122436092)

[**3.2 Unity3D** 17](#_Toc122436093)

[**3.3 Скрипти програми** 19](#_Toc122436094)

[1.4 Код програми 21](#_Toc122436095)

[**4. Аналіз отриманих результатів** 24](#_Toc122436096)

[**СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТРИ** 26](#_Toc122436097)

## **ВСТУП**

Радіолокація – це виявлення об’єктів і визначення їх просторових координат та параметрів руху за допомогою радіотехнічних засобів і методів. Цей процес називається радіолокаційним спостереженням, а пристрої такого призначення – радіолокаційними станціями або радіолокаторами.

Радіолокація ґрунтується на багатьох наукових та інженерних досягненнях. Усе почалося у 1865 році, коли Шотландський фізик Джеймс Клерк Максвел представив свою «Теорію електромагнітного поля», яка містила дослідження електромагнітних хвиль та механізмів їх розповсюдження. Він показав, що електричні поля розповсюджуються у вигляді хвиль з постійною швидкістю. У вільному просторі ця швидкість дорівнює швидкості світла. Це була тільки теорія, яку тільки у 1886 році підтвердив німецький фізик Генріх Рудольф Герц. На пошану цьому чоловіку було названо одиницю вимірювання «герц» - тобто одиниця вимірювання частоти періодичних процесів. Потім у 1900 році, Ніколи Тесла припустив, що відбиття електромагнітних хвиль може бути використане для виявлення рухомих металевих об’єктів. Так почалася історія радіолокаційної технології. Вже через чотири роки після припущень Ніколи Тесли, було побудовано перший радіолокатор – так названий «Телемобілоскоп». Використовувався він для моніторингу руху на воді в умовах поганої видимості. Після цього, у 1922 та 1930 році, співробітниками Військово-морської лабораторії, було вперше виявлено спочатку дерев’яний корабель, а потім і літак у небі. Це був прорив у технології радіолокації. Лише за 10 років з цього моменту, було створено портативні радарні станції, у 1939 такі станції навіть з’явились на борті літаків B-17. У той рік почалася Друга Світова Війна, та ця технологія була дуже ефективна для виявлення німецьких підводних човнів, незалежно від доби. Внаслідок масштабних воєнних дій, де авіації відводилася дуже велика роль, розвиток радіолокаційної технології під час Другої світової війни відбувався швидкими темпами. На той час радіолокаційна технологія використовувалася тільки у військовій галузі. І тільки після закінчення війни, почала використовуватися у цивільній сфері життя.

На зараз, радіолокація застосовується у багатьох напрямках сучасних технологій. У цивільній авіації, для визначення можливих аварій та їх запобігання, у метеорології, для передбачення погоди, у картографії, дослідженні космосу і навіть у швидкісних радарах, які використовує поліція для виявлення швидкісних порушень під час водіння. Через сучасну ситуацію в Україні, технологія військової радіолокації знову стала та залишається дуже важливою. Вона використовується усюди: у системах протиповітряної оборони, які захищають нас щодня, у літаках-винищувачах, для викриття цілей на землі та у повітрі.

В цілому, переваги радіолокаційної технології над візуальним спостереженням колосальні. Завдяки цій розробці, ми можемо слідкувати за об’єктами 24 години на добу, при цьому погодні умови та час доби більше не грають ніякої ролі у якості спостережень.

Під час створення та в ході функціонування радіолокатора використовуються знання з багатьох дисциплін: від будівництва та важкого машинобудування до електротехніки та високочастотної техніки. Хоч теорія радіолокації і є складною, загальний принцип дії довільно простий. Як у каньйоні, ми кричимо, щоб почути ехо нашого голосу, так і тут, ми посилаємо електромагнітний імпульс, та слухаємо його ехо, щоб зрозуміти напрям та дистанцію до об’єкту. Антена радіолокатора опромінює ціль надвисокочастотними сигналами. Відбиті сигнали приймаються приймальною антеною. Антена може слугувати як для відправки, так і для прийому сигналу. Найбільша різниця у тому, що у випадку радіохвиль ми стикаємося із швидкістю світла, яка становить 300000000 метрів у секунду. Електромагнітні хвилі відбиваються, коли на їх шляху трапляється провідна поверхня, тобто, метали. І цей факт робить електромагнітні хвилі ідеальними для викриття літаків, кораблів та, навіть, підводних човнів.

Курсова робота складається з таких частин:

- вступ, який обгрунтовує актуальність роботи; визначає цілі проведення наукового дослідження; галузь дослідження; методи дослідження або розрахунків;

- постановка задачі;

- аналітичний огляд літературних джерел;

- математична модель;

- програмна реалізація;

- висновки;

- список використаної літератури;

- додатки

## **ПОСТАНОВКА ЗАДАЧІ**

Метою даної роботи є ознайомлення з особливостями хвильових процесів, принципами функціонування радарних установок, програмна реалізація візуалізації принципу роботи імпульсного радару.

Для виконання поставлених задач необхідно:

1. дослідити та провести аналіз предметної області, а саме: проаналізувати принципи функціонування радарних пристроїв, зокрема випромінювачів та приймачів сигналу, галузі використання радарів;
2. побудувати математичну модель роботи радарної системи;
3. визначити функціональні вимоги до програмного забезпечення, обрати мову програмування та платформу для програмної реалізації;
4. спроектувати, розробити та протестувати програму візуалізації роботи радарних пристроїв;
5. розробити інтерфейс для зручного користування додатком;
6. проаналізувати отримані результати;
7. зробити висновки, визначити способи подальшого поліпшення програми візуалізації роботи радарних пристроїв.

## **1. АНАЛІТИЧНИЙ ОГЛЯД ЛІТЕРАТУРНИХ ДЖЕРЕЛ**

## 1.1 Хвиля

Хвиля — це процес розповсюдження коливань у будь-якому фізичному середовищі. При цьому частинки середовища не рухаються разом з хвилею, а коливаються навколо своїх положень рівноваги.

Хвилі характеризують величиною збурення — амплітудою й напрямком поширення.

Поняття хвилі є фундаментальним поняттям фізики, розуміння змісту якого необхідне при аналізі широкого кола явищ в сучасному світі. Хвилі існують усюди навколо нас. Світло, звуки — це хвилі. У квантовому світі навіть частинки ведуть себе, як хвилі.

## 1.2 Різновиди хвиль

Хвилі поділяються на наступні різновиди:

**За характером розповсюдження:**

* **Біжучі хвилі -** це хвильовий рух, під час якого поверхня рівних фаз переміщується з кінцевою швидкістю. *Приклад: Пружні хвилі у стрижні.*
* **Стоячі хвилі**, які при будь-якій фазі коливань не поширюються в просторі. Характерною особливістю є наявність у ній вузлів та пучностей. У вузлах амплітуда хвилі дорівнює нулю, а у пучностях амплітуда максимальна. Така хвиля утворюється в результаті накладання двох біжучих хвиль, які поширюються назустріч одна одній. *Приклад: зафіксована мотузка на одному кінці, яка коливається вручну або поршнем на іншому. Генерує стоячі хвилі вздовж її довжини.*

**За типом коливань:**

* **Поперечні -** це хвилі, у яких коливання відбуваються в площині, перпендикулярній до напрямку поширення.

*Приклад: Електромагнітні хвилі у вакуумі.*

* **Повздовжні -** хвилі, у яких коливання в кожній точці простору паралельні напрямку розповсюдження.

*Приклад: Поздовжні звукові хвилі в газі.*

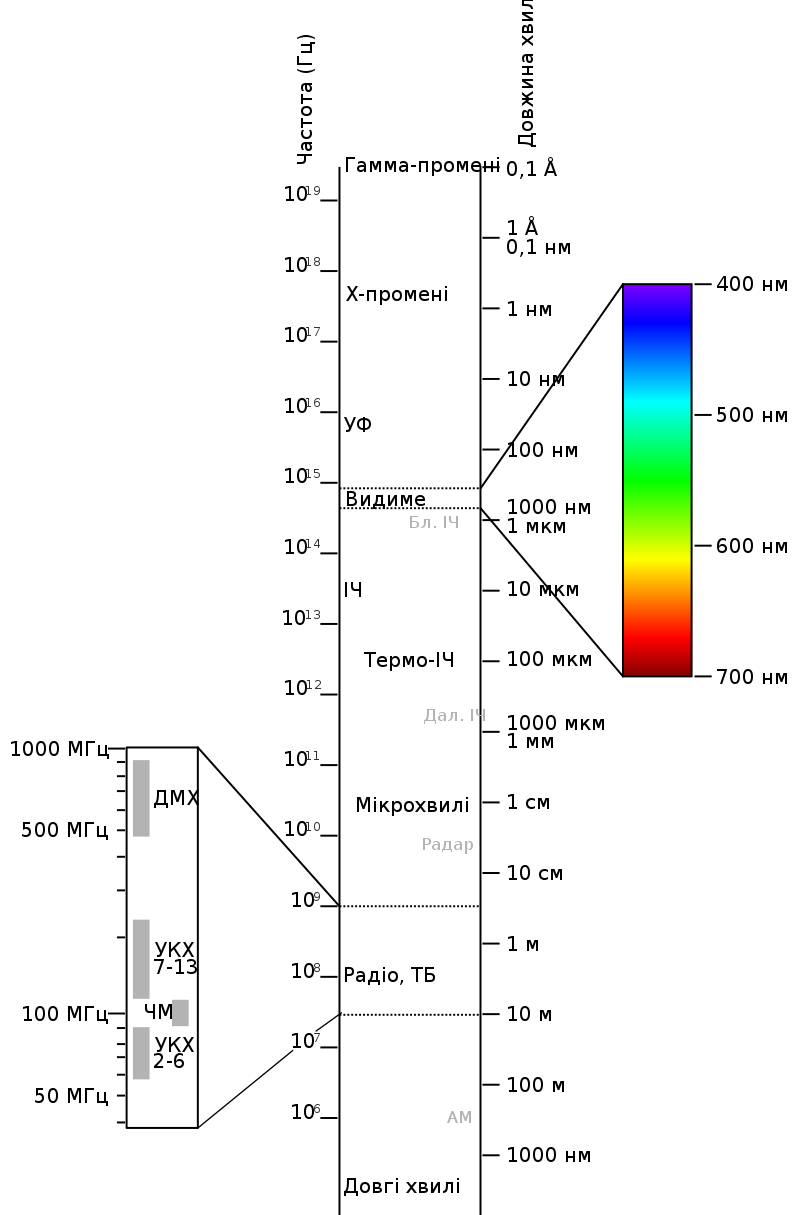
## 1.3 Електромагнітне випромінювання

Електромагнітне випромінювання — це взаємопов’язані коливання електричного і магнітного полів, що утворюють електромагнітне поле.

Електромагнітне випромінювання поділяється на:

* Радіохвилі
* Інфрачервоне випромінювання
* Видиме випромінювання
* Ультрафіолетове випромінювання
* Рентгенівське випромінювання
* Гамма випромінювання

Рисунок 1.1 - Типи електромагнітного випромінювання (1)

Рисунок 1.2 - Типи електромагнітного випромінювання (2)

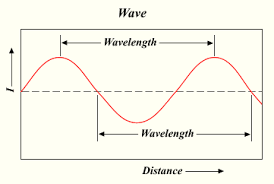
Електромагнітні хвилі мають декілька основних властивостей:

* Довжина хвилі
* Частота хвилі
* Період хвилі
* Фаза хвилі

## 1.4 Довжина хвилі

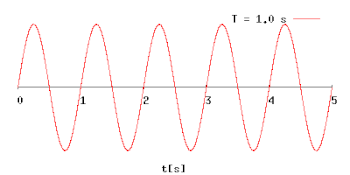
Довжина хвилі — це відстань, взята вздовж променя, між двома точками, які коливаються в одній фазі. Позначається, як λ (лямбда) і вимірюється в одиницях довжини. [(1.1)](#Object5|ole)

|  |  |
| --- | --- |
|  | (1.1) |

Рисунок 1.3 - Візуалізація довжини хвилі

## 1.5 Період хвилі

Період хвилі — це час, за який хвиля поширюється на відстань, що дорівнює довжині хвилі. Позначається, як T, та вимірюється в одиницях часу.

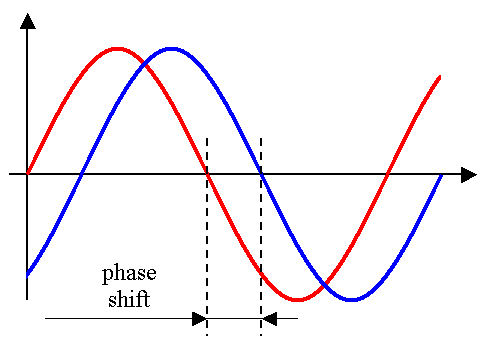
Рисунок 1.4 - Період хвилі

## 1.6 Частота хвилі

Частота — це кількість коливань, які здійснить частинка хвилі за одиницю часу. Позначається, як v (ню) і вимірюється в одиницях, обернених до одиниць часу.

## 1.7 Фаза хвилі

Фаза — це характеристика коливання, що визначає вадмінність між двома подібними коливаннями, які починаються в різні моменти часу.

Рисунок 1.5 - Візуалізація зсуву фази

## 1.8 Математичний опис хвилі

Описується хвиля за допомогою наступного рівняння [(1.2)](#Object6|ole), де х — просторова змінна, t – часова, A – амплітуда, v – частота хвилі.

|  |  |
| --- | --- |
|  | (1.2) |

Також хвильову функцію можна записати, як диференційне рівняння другого порядку, але у даній візуалізації було достатньо першого рівняння. [(1.3)](#Object7|ole)

|  |  |
| --- | --- |
|  | (1.3) |

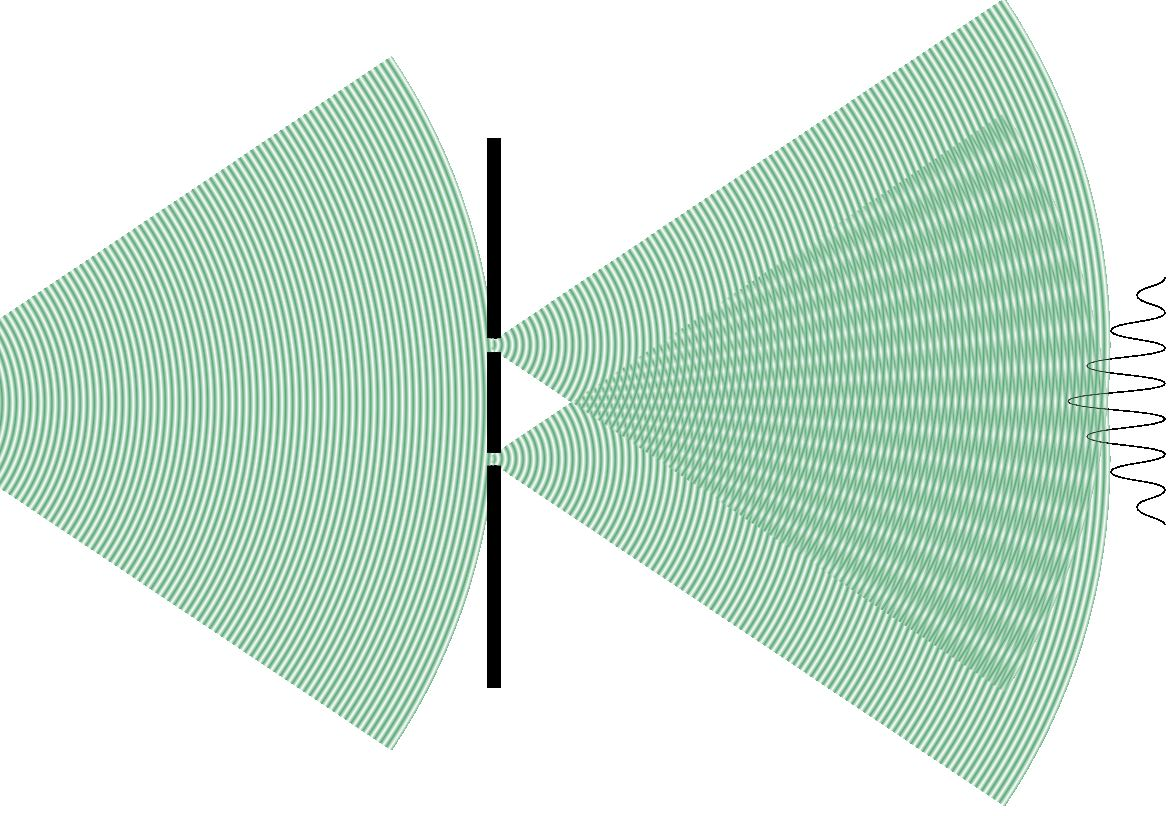
## 1.9 Інтерференція

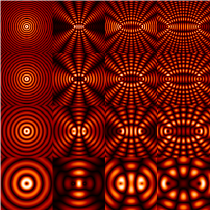
Інтерференція — це явище накладання двох або більше когерентних хвиль, в результаті чого в одних місцях спостерігається підсилення кінцевої хвилі, а в інших послаблення. Вона спостерігається у когерентних хвилях різноманітної природи — на поверхні води, у поперечних, поздовжніх звукових та електромагнітних.

Явище інтерференції використовується, наприклад, в радіотехніці і акустиці для створення складних антен. Особливо велике значення інтерференція має в оптиці, вона лежить в основі оптичної та акустичної голографії.

Інтерференція спостерігається у когерентних хвиль довільної природи— поверхневих (на воді), поперечних та поздовжніх звукових, електромагнітних (світло, радіохвилі).

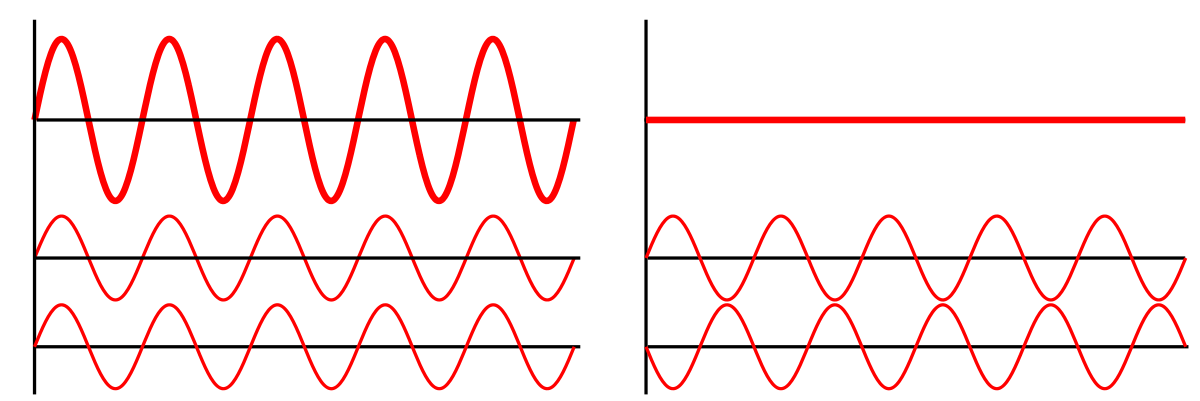
При інтерференції результативне коливання є геометричною сумою коливань обох хвиль у відповідних точках [1.]. Цей принцип суперпозиції як правило є точним і порушується у окремих випадках, в деяких середовищах, коли амплітуда коливань є дуже високою (нелінійна оптика, нелінійна акустика).

Рисунок 1.6 - Інтерференція від двох щілин

Рисунок 1.7 - Картина інтерференції двох кругових хвиль, у залежності від довжини хвилі та відстані між джерелами

При зіткненні, звилі поділяються на конструктивні, та деструктивні:

* Конструктивні хвилі накладаються одне на одного.
* Деструктивні знищують одне одного.

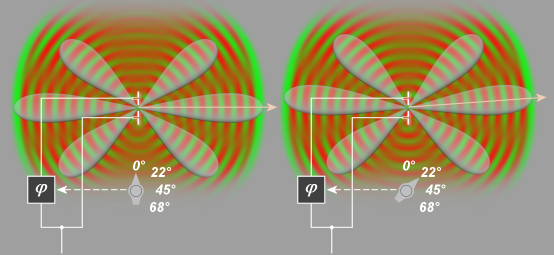
Рисунок 1.8 – Конструктивні (ліворуч) та деструктивні (праворуч) хвилі

## 1.10 Зсув фаз у ФАР

Фазована антенна решітка базується на принципу зсуву фаз. За допомогою цього ФАР повертає сигнал. Оскільки відстань між сусідніми випромінювачами у ФАР завжди однакова, зсув фази між ними теж буде однаковим. Таким чином зсув фази між першим і останнім випромінювачем буде у n раз більше, ніж між сусідніми випромінювачами. Через це, при виходу з ладу одного з випромінювачей, ФАР не втратить роботоздатносні, але її ефективність зменшиться.

Відношення кута зсуву променя та зсуву фази задає формула [(1.4)](#Object8|ole), де d – відстань між сусідніми випромінювачами, λ — довжина хвилі, φ — фазовий зсув, Θ — кут нахилу променя.

|  |  |
| --- | --- |
|  | (1.4) |

Рисунок 1.9 - Ліворуч: два випромінювача з однаковою фазой. Праворуч: два випромінювача з зсувом фази

## **2. Математична модель**

Для симуляції роботи радарної установки, нам потрібна константа швидкості світла. Після цього ми визначаємо час, за який хвиля долетить до об’єкту, відіб’ється та повернеться назад до радару. Робиться це за формулою (2.1)

|  |  |
| --- | --- |
|  | (2.1) |

Після цього ми отримуємо час, за який хвиля повернеться до радару. За оберненою формулою (2.2), ми зможемо знайти відстань до об’єкта, але вже виміряну за допомогою часу затримки нашого сигналу.

|  |  |
| --- | --- |
|  | (2.2) |

де -відстань від антени до цілі, -виміряне значення часу проходження імпульсу, -швидкість світла.

|  |  |
| --- | --- |
|  | (2.3) |

Зміна частоти сигналу через ефект Допплера (2.4)

|  |  |
| --- | --- |
|  | (2.4) |

де початкова частота.

Потім по різниці частоти відправлення та отримання сигналу ми можемо знайти радіальну швидкість об’єкта.

Изображение выглядит как текст, антенна

Автоматически созданное описание

Рисунок 2.1 – Візуалізація принципу визначення координат об’єкта

|  |  |
| --- | --- |
|  | (2.5) |

Де A, B та P – точки граничних приймачів сигналу у декартовій системі координат, .

|  |  |
| --- | --- |
|  | (2.6) |

## 3. **ПРОГРАМНА РЕАЛІЗАЦІЯ**

## **3.1 Обрання інструментів для програмної реалізації**

Для реалізації програми я обрав платформу Unity3D. За допомогою нього можна одразу після створення проекту перейти до реалізації задумки. Це дозволяє значно скоротити час розробки та написати більш оптимізовану програму. Саму візуалізацію я вирішив робити за допомогою шейдерів, це дозволить зробити симуляцію у реальному часі.

Unity3D підтримує мову програмування C# і шейдерну мову програмування CG. Отже я використав їх.

## **3.2 Unity3D**

У Unity все починається зі сцени. Це наш навколишній простір, який зберігає всі об’єкти і у якому виконується уся програма.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.1 - Hierarchy window

На сцені кожен об’єкт — це об’єкт типу gameObject. Це може бути будь що:

1) Камера, через яку ми бачимо простір.

2) Площина, на якій ми малюємо розповсюдження хвиль.

3) UI, який дозволяє керувати програмою.

4) І навіть об’єкти, які ми не бачимо, контролери та освітлення.

Ці об’єкти можуть об’єднуватися у складну ієрархію, що допомагає нам їх групувати.

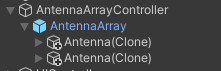


Рисунок 3.2 - Parent-child hierarchy

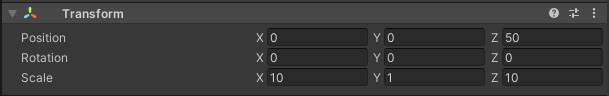
Кожен gameObject може зберігати у собі компоненти, в тому числі і скрипти, де ми і пишемо логіку програми. Кожен gameObject обов’язково має компонент Transform, який дозволяє орієнтувати його у просторі.

Все це ми бачимо через вікно з назвою: Inspector. Воно доступне нам тільки у редакторі й дозвоняє зручно налаштовувати програму. Саме тут ми можемо керувати нашими компонентами, додавати скрипти, або, наприклад, рендерер поверхні, щоб візуально бачити об’єкт.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.3 - Inspector menu

Рисунок 3.4 - Transform component

## **3.3 Скрипти програми**

В цілому в програмі існує 12 різних скриптів та один шейдер.

1. Antenna.cs
2. BayractarMover.cs
3. RadarDetailNode.cs
4. RadarListNode.cs
5. RadarPlane.cs
6. RadarScutter.cs
7. RadarScutterGUI.cs
8. ScutterCollider.cs
9. ScutterRayMeshBuilder.cs
10. ScutterRayMeshBuilderGUI.cs
11. UIController.cs
12. CamToBairaktar.cs
13. RadarImageShader.shader

У віртуальному просторі вже літають 3 об’єкти (байрактари), які ми і будемо відслідковуємо.

Изображение выглядит как пол

Автоматически созданное описание

Рисунок 3.5 - Bairaktar

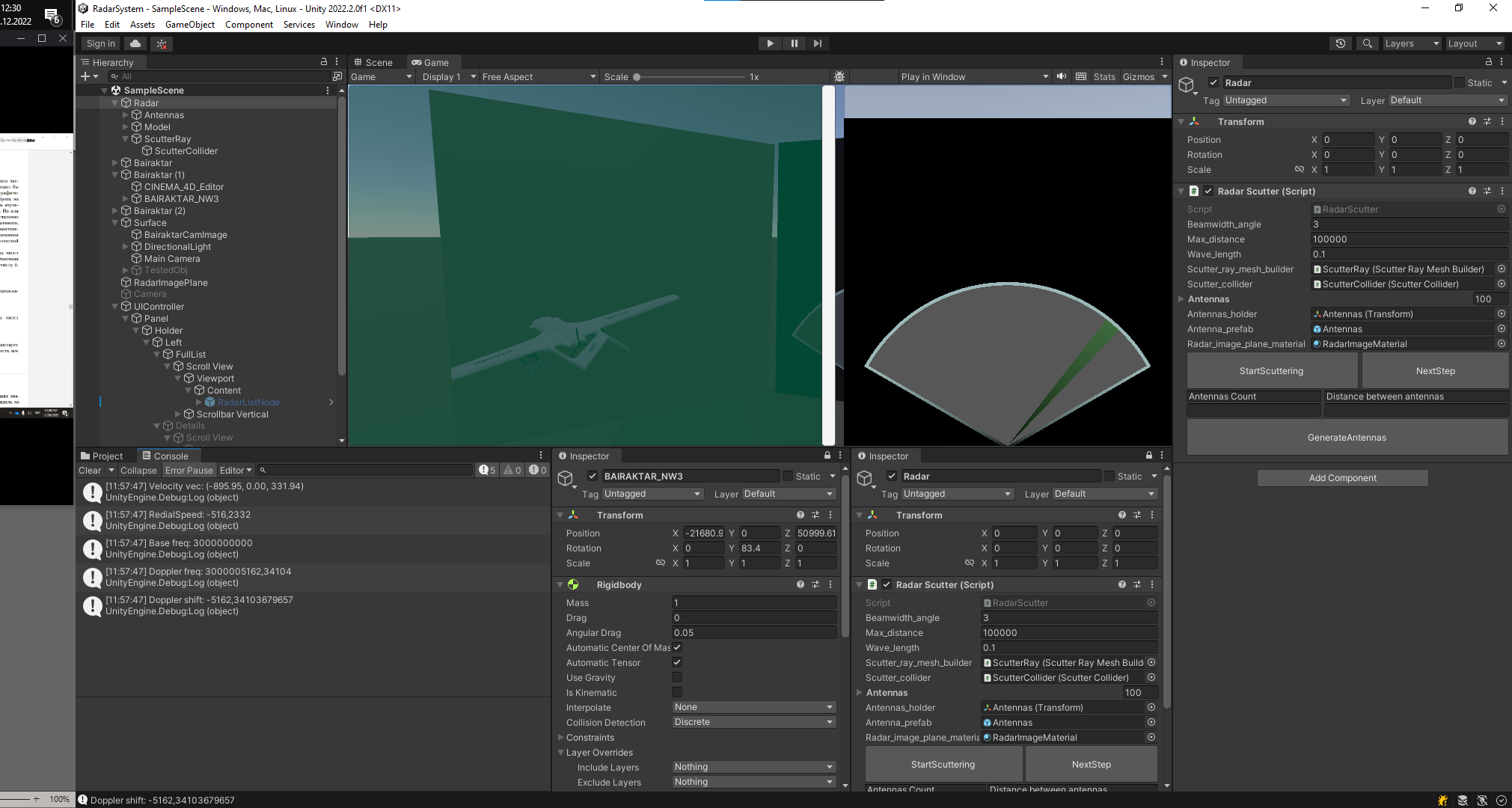


Рисунок 3.6 – Вигляд програми

У програму задані значення максимальної дальності установки – 100км, а також кут розсіювання імпульсу, який я знайшов із своєї курсової роботи за минулий семестр.

Изображение выглядит как текст, электроника, темный

Автоматически созданное описание

Рисунок 3.7 – Кут розсіювання імпульсу

Фазована антенна решітка складається зі 100 антен, відстань між сусідами – 1 см.

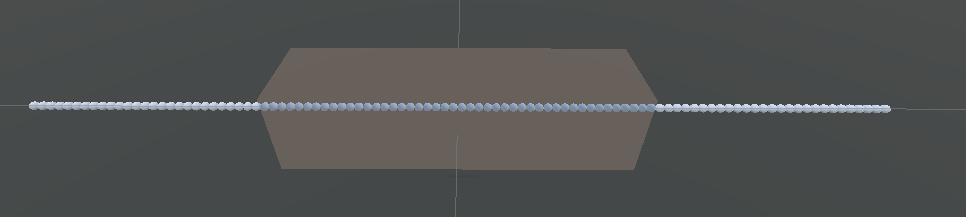


Рисунок 3.8 – Фазована антенна решітка

Після початку сканування, байрактари починають рухатися, я радар починає сканувати простір у спробі знайти ці байрактари.

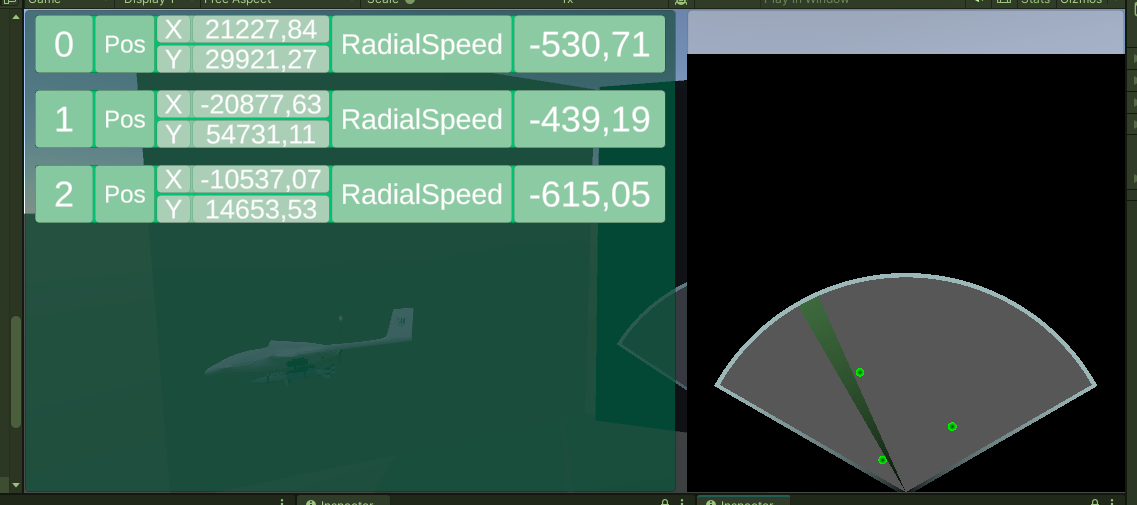


Рисунок 3.9 – Робота програми

Також можна вибрати одну з точок на радарі, щоби подивитися на історію позицій об’єкту разом із швидкостями на той момент.

## Код програми

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.10

Початок сканування об’єкту, визначення часу затримки сигнала, рахування скорості по ефекту Допплера.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.11

Визначення позиції об’єкта по знайденому куту, позиції між приймачами та затримками сигналу на кожному приймачі.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3.12

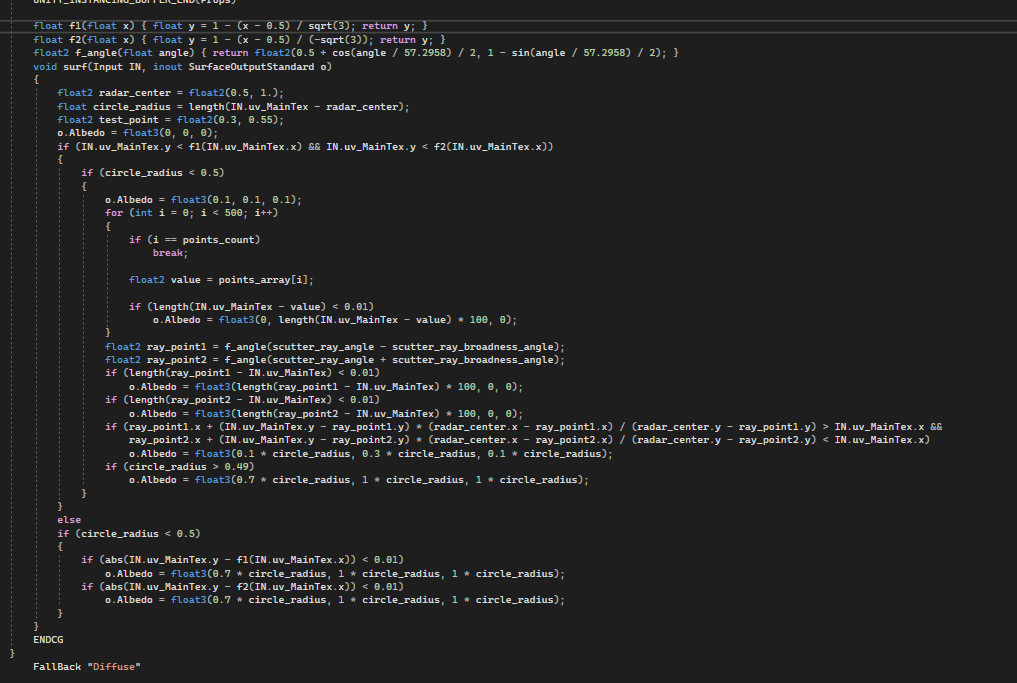


Рисунок 3.13 - шейдер

Шейдер для рендерінгу радарного індикатору кутом у 120 градусів, рендер самого півкола, променя сигналу та позицій знайдених об’єктів.

## **4. Аналіз отриманих результатів**

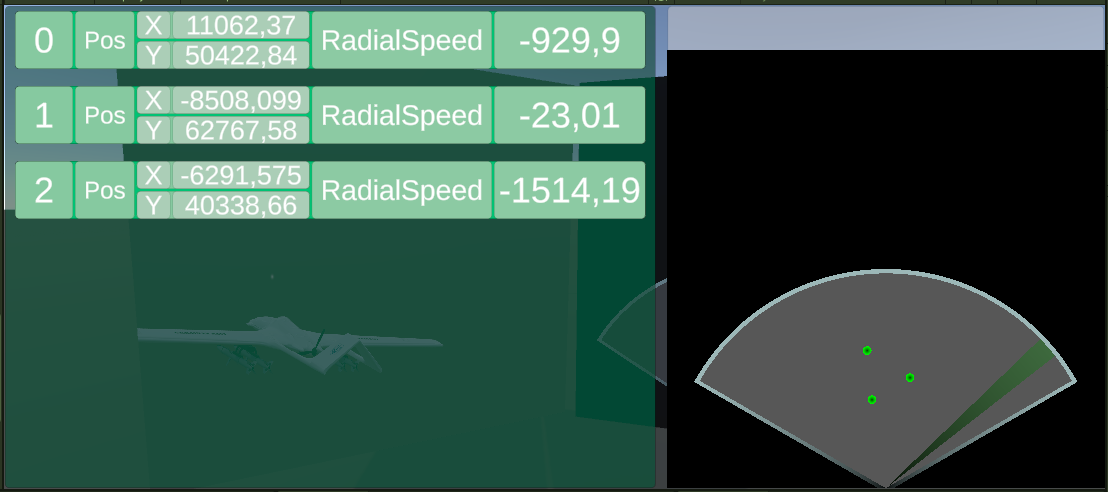


Рисунок 4.1 – виконання програми

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4.1 – погрішність визначення координат

З рисунку (4.1), ми бачимо, що точність знайдених координат достатньо висока. Значення виведені у метрах, а отже погрішність становить до пів-метра і викликана тільки недостатньою точністю змінних.

При використанні метода тріангуляції, точність радарної установки залежить тільки від точності вимірювання часу сигналу на приймачах, і, в теорії, якщо ця точність буде дуже великою (наприклад, до нано-секунд), то результати будуть максимально точними.

Також, для зменшення необхідної точності вимірювальних пристроїв, ми можемо збільшити відстань між краєвими приймачами сигналу.

**ВИСНОВКИ**

За результатом цієї курсової роботи, я:

- дослідив та провів аналіз предметної області, а саме: проаналізував принципи функціонування радарних пристроїв, зокрема випромінювачів та приймачів сигналу, галузі використання радарів;

- побудував математичну модель роботи радарної системи;

- визначив функціональні вимоги до програмного забезпечення, обрав мову програмування та платформу для програмної реалізації;

- спроектував та розробив програму візуалізації роботи радарних пристроїв;

- проаналізував отримані результати;

- зробив висновки, визначив способи подальшого поліпшення програми

візуалізації роботи радарних пристроїв.

Таким чином, використовуючи метод триангуляції з двох відомих точок та значень затримки сигналів, ми можемо помічати об’єкти у повітрі з надвисокою точністю. Це можливо зробити через застосування фазованої антенної решітки, яку можна використовувати не тільки для випромінювання сигналу, а і для прийому. Окрім високої точності, ми отримуємо ще декілька великих переваг при використанні ФАР. Наш радар не має рухомих деталей, що, в теорії, тільки збільшить час експлуатації установки. Також радар на системі ФАР зможе миттєво перемикати промінь випромінювання на інший кут, що неможливо зробити при використанні рухомих тарілкових антен.

# **СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТРИ**

1. Хвильові вимірювання. Навчальний посібник з дисципліни “ Оптичні

вимірювання ”, частина IIІ: [по напрямку підготовки “Оптико-електронне

приладобудування”] / Л.А. Міхеєнко. – Київ, 2011. – 4 ст.

1. Mathematics of waves — Режим доступу: [https://phys.libretexts.org](https://phys.libretexts.org/Bookshelves/University_Physics/Book%3A_University_Physics_(OpenStax)/Book%3A_University_Physics_I_-_Mechanics_Sound_Oscillations_and_Waves_(OpenStax)/16%3A_Waves/16.03%3A_Mathematics_of_Waves#:~:text=To%20find%20the%20amplitude%2C%20wavelength,T%3D2πω).)
2. SurfaceShaders – Режим доступу: [SL-SurfaceShaders](https://docs.unity3d.com/Manual/SL-SurfaceShaders.html)
3. Фазованні антенні решітки - Режим доступу: [https://habr.com](https://habr.com/ru/post/159109/)
4. Фазованні антенні решітки - Режим доступу: [https://www.radartutorial.eu](https://www.radartutorial.eu/06.antennas/an29.ru.html)
5. Проектування фазованних антенних решіток - Режим доступу: [https://ru.wikipedia.org](https://ru.wikipedia.org/wiki/Проектирование_фазированных_антенных_решёток)
6. ShaderLab: defining material properties - Режим доступу: [https://docs.unity3d.com](https://docs.unity3d.com/Manual/SL-Properties.html)
7. Nvidia Cg Toolkit Documentation - Режим доступу: [https://developer.download.nvidia.com](https://developer.download.nvidia.com/cg/)
8. Радарні системи - Режим доступу: [https://radartutorial.eu](https://www.radartutorial.eu/01.basics/rb09.uk.html)
9. Radar ang electronic warfare, illustrated collection of definitions and formulas / Dipl.-Ing. Christian Wolff

**ДОДАТКИ**

Shader "Custom/RadarImageShader"

{

Properties

{

\_Color("Color", Color) = (1,1,1,1)

\_MainTex("RenderTexture", 2D) = "" {}

scutter\_ray\_angle("Scutter ray angle", Range(30, 150)) = 90

scutter\_ray\_broadness\_angle("Scutter ray broadness angle", Range(0, 30)) = 3

}

SubShader

{

//Tags { "RenderType"="Opaque" "RenderType" = "Transparent" }

Tags {"Queue" = "Transparent" "IgnoreProjector" = "True" "RenderType" = "Transparent"}

ZWrite Off

Blend SrcAlpha OneMinusSrcAlpha

LOD 200

CGPROGRAM

// Physically based Standard lighting model, and enable shadows on all light types

#pragma surface surf Standard fullforwardshadows

// Use shader model 3.0 target, to get nicer looking lighting

#pragma target 3.0

sampler2D \_MainTex;

float scutter\_ray\_angle;

float scutter\_ray\_broadness\_angle;

int points\_count;

float2 points\_array[500];

struct Input

{

float2 uv\_MainTex;

};

fixed4 \_Color;

UNITY\_INSTANCING\_BUFFER\_START(Props)

UNITY\_INSTANCING\_BUFFER\_END(Props)

float f1(float x) { float y = 1 - (x - 0.5) / sqrt(3); return y; }

float f2(float x) { float y = 1 - (x - 0.5) / (-sqrt(3)); return y; }

float2 f\_angle(float angle) { return float2(0.5 + cos(angle / 57.2958) / 2, 1 - sin(angle / 57.2958) / 2); }

void surf(Input IN, inout SurfaceOutputStandard o)

{

float2 radar\_center = float2(0.5, 1.);

float circle\_radius = length(IN.uv\_MainTex - radar\_center);

float2 test\_point = float2(0.3, 0.55);

o.Albedo = float3(0, 0, 0);

if (IN.uv\_MainTex.y < f1(IN.uv\_MainTex.x) && IN.uv\_MainTex.y < f2(IN.uv\_MainTex.x))

{

if (circle\_radius < 0.5)

{

o.Albedo = float3(0.1, 0.1, 0.1);

for (int i = 0; i < 500; i++)

{

if (i == points\_count)

break;

float2 value = points\_array[i];

if (length(IN.uv\_MainTex - value) < 0.01)

o.Albedo = float3(0, length(IN.uv\_MainTex - value) \* 100, 0);

}

float2 ray\_point1 = f\_angle(scutter\_ray\_angle - scutter\_ray\_broadness\_angle);

float2 ray\_point2 = f\_angle(scutter\_ray\_angle + scutter\_ray\_broadness\_angle);

if (length(ray\_point1 - IN.uv\_MainTex) < 0.01)

o.Albedo = float3(length(ray\_point1 - IN.uv\_MainTex) \* 100, 0, 0);

if (length(ray\_point2 - IN.uv\_MainTex) < 0.01)

o.Albedo = float3(length(ray\_point2 - IN.uv\_MainTex) \* 100, 0, 0);

if (ray\_point1.x + (IN.uv\_MainTex.y - ray\_point1.y) \* (radar\_center.x - ray\_point1.x) / (radar\_center.y - ray\_point1.y) > IN.uv\_MainTex.x &&

ray\_point2.x + (IN.uv\_MainTex.y - ray\_point2.y) \* (radar\_center.x - ray\_point2.x) / (radar\_center.y - ray\_point2.y) < IN.uv\_MainTex.x)

o.Albedo = float3(0.1 \* circle\_radius, 0.3 \* circle\_radius, 0.1 \* circle\_radius);

if (circle\_radius > 0.49)

o.Albedo = float3(0.7 \* circle\_radius, 1 \* circle\_radius, 1 \* circle\_radius);

}

}

else

if (circle\_radius < 0.5)

{

if (abs(IN.uv\_MainTex.y - f1(IN.uv\_MainTex.x)) < 0.01)

o.Albedo = float3(0.7 \* circle\_radius, 1 \* circle\_radius, 1 \* circle\_radius);

if (abs(IN.uv\_MainTex.y - f2(IN.uv\_MainTex.x)) < 0.01)

o.Albedo = float3(0.7 \* circle\_radius, 1 \* circle\_radius, 1 \* circle\_radius);

}

}

ENDCG

}

FallBack "Diffuse"

}

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using static UnityEditor.PlayerSettings;

public class RadarScutter : MonoBehaviour

{

[SerializeField] float beamwidth\_angle;

[SerializeField] float max\_distance;

[SerializeField] double wave\_length;

[SerializeField] ScutterRayMeshBuilder scutter\_ray\_mesh\_builder;

[SerializeField] ScutterCollider scutter\_collider;

[SerializeField] List<Antenna> antennas = new List<Antenna>();

[SerializeField] Transform antennas\_holder;

[SerializeField] GameObject antenna\_prefab;

[SerializeField] Material radar\_image\_plane\_material;

List<(List<(Vector4, double)>, GameObject)> tracked\_points = new List<(List<(Vector4, double)>, GameObject)>();

public List<(List<(Vector4, double)>, GameObject)> TrackedPoints => tracked\_points;

static RadarScutter instance;

public static RadarScutter Instance => instance;

private void Awake()

{

if(instance == null)

instance = this;

}

private void Start()

{

if (tracked\_points == null)

tracked\_points = new List<(List<(Vector4, double)>, GameObject)>();

tracked\_points.Clear();

SetElements();

}

public void SetElements()

{

List<Vector4> only\_points = new List<Vector4>();

for (int i = 0; i < tracked\_points.Count; i++)

only\_points.Add(new Vector2(0.5f - (float)(tracked\_points[i].Item1[tracked\_points[i].Item1.Count - 1].Item1.x / max\_distance / 2), 1.0f - (float)(tracked\_points[i].Item1[tracked\_points[i].Item1.Count - 1].Item1.y / max\_distance / 2)));

if (only\_points.Count < 500)

while (only\_points.Count < 500)

only\_points.Add(Vector2.zero);

radar\_image\_plane\_material.SetVectorArray("points\_array", only\_points);

radar\_image\_plane\_material.SetInt("points\_count", tracked\_points.Count);

}

float current\_angle = 0;

const double LightSpeed = 300000000; // Meters per second

public void GenerateAntennas(int count, float distance)

{

if (antennas == null)

antennas = new List<Antenna>();

for (int i = 0; i < antennas.Count; i++)

Destroy(antennas[i].gameObject);

antennas.Clear();

for (int i = 0; i < count; i++)

{

Antenna buff = Instantiate(antenna\_prefab, antennas\_holder).GetComponent<Antenna>();

buff.transform.localPosition = new Vector3(distance \* (1 + 2 \* i - count) / 2, 0, 0);

antennas.Add(buff);

}

}

public void OnScutterColliderHit(Collider other)

{

Debug.Log("OnScutterColliderHitCallback");

ScutterRayToObject(other);

}

void ScutterRayToObject(Collider hit\_collider)

{

double time\_to\_first\_receiver;

double time\_to\_second\_receiver;

double time\_to\_tracked\_object;

double doppler\_frequency;

double base\_frequency = LightSpeed / wave\_length;

Vector3d pos;

Vector3d vec;

Vector3d first\_receiver\_position = new Vector3d(antennas[0].transform.position.x, 0, antennas[0].transform.position.z);

Vector3d second\_receiver\_position = new Vector3d(antennas[antennas.Count - 1].transform.position.x, 0, antennas[antennas.Count - 1].transform.position.z);

Vector3d radar\_pos = new Vector3d(transform.position.x, 0, transform.position.z);

{ //Variables, defined in this local space are forbidden to use in position calculations.

Vector3d tracked\_obj\_pos = new Vector3d(hit\_collider.transform.position.x, 0, hit\_collider.transform.position.z);

double distance\_to\_obj = Vector3d.Distance(radar\_pos, tracked\_obj\_pos);

double distance\_to\_first\_receiver = Vector3d.Distance(tracked\_obj\_pos, first\_receiver\_position);

double distance\_to\_second\_receiver = Vector3d.Distance(tracked\_obj\_pos, second\_receiver\_position);

//Need to be signed

Vector3 speed\_projection = Vector3.Project(hit\_collider.GetComponent<Rigidbody>().velocity, transform.position - hit\_collider.transform.position);

float radial\_speed = Vector3.Magnitude(speed\_projection);

float speed\_vec\_angle = Vector3.Angle(hit\_collider.transform.position, speed\_projection);

if (speed\_vec\_angle < 90)

radial\_speed \*= -1;

doppler\_frequency = base\_frequency \* (LightSpeed / (LightSpeed + radial\_speed));

//ClearLog();

//Debug.Log($"Velocity vec: {hit\_collider.GetComponent<Rigidbody>().velocity}");

//Debug.Log($"RedialSpeed: {radial\_speed}");

//Debug.Log($"Base freq: {base\_frequency}");

//Debug.Log($"Doppler freq: {doppler\_frequency}");

//Debug.Log($"Doppler shift: {base\_frequency - doppler\_frequency}");

time\_to\_tracked\_object = distance\_to\_obj / LightSpeed;

time\_to\_first\_receiver = distance\_to\_first\_receiver / LightSpeed;

time\_to\_second\_receiver = distance\_to\_second\_receiver / LightSpeed;

}

// Katet, that represent distance to second receiver, after it has touch first receiver.

double delay\_distance\_between\_receivers;

double dist\_between\_receivers = Vector3d.Distance(first\_receiver\_position, second\_receiver\_position);

double angle;

Vector3d obj\_pos;

if (time\_to\_first\_receiver < time\_to\_second\_receiver)

{

delay\_distance\_between\_receivers = LightSpeed \* (time\_to\_second\_receiver - time\_to\_first\_receiver);

angle = Mathd.Acos(delay\_distance\_between\_receivers / dist\_between\_receivers) \* Mathd.Rad2Deg;

pos = first\_receiver\_position;

vec = new Vector3d(Quaternion.AngleAxis((float)angle + 180, Vector3.up) \* Vector3.right);

obj\_pos = LightSpeed \* time\_to\_tracked\_object \* vec + first\_receiver\_position;

}

else// if (time\_to\_first\_receiver > time\_to\_second\_receiver)

{

delay\_distance\_between\_receivers = LightSpeed \* (time\_to\_first\_receiver - time\_to\_second\_receiver);

angle = -Mathd.Acos(delay\_distance\_between\_receivers / dist\_between\_receivers) \* Mathd.Rad2Deg;

pos = second\_receiver\_position;

vec = new Vector3d(Quaternion.AngleAxis((float)angle, Vector3.up) \* Vector3.right);

obj\_pos = LightSpeed \* time\_to\_tracked\_object \* vec + second\_receiver\_position;

}

vec += (first\_receiver\_position + second\_receiver\_position) / 2;

bool is\_has\_obj = false;

int obj\_index = -1;

for (int i = 0; i < tracked\_points.Count; i++)

{

if (tracked\_points[i].Item2 == hit\_collider.gameObject)

{

is\_has\_obj = true;

obj\_index = i;

break;

}

}

double founded\_radial\_speed = LightSpeed \* base\_frequency / doppler\_frequency - LightSpeed;

if (!is\_has\_obj)

{

tracked\_points.Add((new List<(Vector4, double)>(), hit\_collider.gameObject));

tracked\_points[tracked\_points.Count - 1].Item1.Add((new Vector2((float)obj\_pos.x, (float)obj\_pos.z), Time.timeAsDouble));

UIController.Instance.AddNode(tracked\_points.Count - 1, new Vector2((float)obj\_pos.x, (float)obj\_pos.z), founded\_radial\_speed);

}

else

{

tracked\_points[obj\_index].Item1.Add((new Vector2((float)obj\_pos.x, (float)obj\_pos.z), Time.timeAsDouble));

UIController.Instance.ChangeNode(obj\_index, new Vector2((float)obj\_pos.x, (float)obj\_pos.z), founded\_radial\_speed);

}

SetElements();

ClearLog();

Debug.Log(new Vector2(0.5f - (float)(obj\_pos.x / max\_distance / 2), 1.0f - (float)(obj\_pos.z / max\_distance / 2)));

Debug.Log($"Points count: {tracked\_points.Count}");

Debug.Log($"Angle: {angle}");

Debug.Log($"Impulse delay time: {time\_to\_tracked\_object + time\_to\_first\_receiver}");

Debug.Log($"Time to tracked object: {time\_to\_tracked\_object}");

Debug.Log($"Time to echo to reach first receiver: {time\_to\_first\_receiver}");

Debug.Log($"Time to echo to reach second receiver: {time\_to\_second\_receiver}");

Debug.Log($"Real Obj Pos: {hit\_collider.transform.position}");

Debug.Log($"Found Obj Pos: {obj\_pos}");

}

public void StartScuttering()

{

if (antennas.Count < 2)

return;

scutter\_ray\_mesh\_builder.BuildMesh(Vector3.Distance(antennas[0].transform.position, antennas[antennas.Count - 1].transform.position), beamwidth\_angle, max\_distance);

scutter\_collider.RegisterCallback(OnScutterColliderHit);

current\_angle = 60;

radar\_image\_plane\_material.SetFloat("scutter\_ray\_broadness\_angle", beamwidth\_angle);

max\_time = Time.timeAsDouble + max\_distance / LightSpeed;

StartCoroutine(TimeCounter());

}

public void StopScuttering()

{

StopCoroutine(TimeCounter());

}

IEnumerator TimeCounter()

{

NextStep();

//Debug.Log(Time.timeAsDouble);

yield return new WaitUntil(() => Time.timeAsDouble > max\_time);

StartCoroutine(TimeCounter());

}

public void NextStep()

{

ScutterToAngle(current\_angle);

current\_angle -= beamwidth\_angle / 3;

if (current\_angle < -60)

current\_angle = 60;

}

double time;

double max\_time;

void ScutterToAngle(float angle)

{

radar\_image\_plane\_material.SetFloat("scutter\_ray\_angle", current\_angle + 90);

scutter\_collider.transform.rotation = Quaternion.Euler(0, current\_angle, 0);

}

public void ClearLog()

{

var assembly = System.Reflection.Assembly.GetAssembly(typeof(UnityEditor.Editor));

var type = assembly.GetType("UnityEditor.LogEntries");

var method = type.GetMethod("Clear");

method.Invoke(new object(), null);

}

}