ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

Відділення інженерної інфраструктури та комп'ютерних наук

Циклова комісія спеціальності «Комп'ютерні науки»

Реєстраційний № _____
Дата реєстрації _____

СИМУЛЯТОР ОПТИЧНИХ СИСТЕМ І ЗАЛОМЛЕННЯ

Пояснювальна записка
до кваліфікаційної роботи
здобувача фахової передвищої освіти
галузі знань 12 Інформаційні технології
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми Комп'ютерні науки
кваліфікації фахового молодшого бакалавра з комп'ютерних наук
групи П-43стн
ЯРИНИЧУ Сергію Олександровичу

Керівник:
ДАЦЮК Денис Васильович

2026

РЕФЕРАТ

Обсяг пояснювальної записки: 51 с., 18 рис., 8 табл., 2 дод., 24 джерела.

КЛЮЧОВІ СЛОВА: ВІДБИТТЯ, ГЕОМЕТРИЧНА ОПТИКА, ЗАКОН СНЕЛЛІУСА, ЗАЛОМЛЕННЯ, НАВЧАЛЬНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ПОВНЕ ВНУТРІШНЄ ВІДБИТТЯ, СИМУЛЯТОР, ТРАСУВАННЯ ПРОМЕНІВ, .NET 9, MVVM, WPF.

У роботі розроблено інтерактивний десктопний симулятор оптичних систем і заломлення для навчального процесу з фізики. Застосунок реалізовано мовою C# на платформі .NET 9 з використанням фреймворку WPF та архітектурного патерну MVVM. Фізичне ядро базується на рекурсивному алгоритмі трасування променів з урахуванням законів заломлення (Снелліус), відбиття, повного внутрішнього відбиття та коефіцієнтів Френеля. Реалізовано дев'ять типів оптичних елементів, три типи джерел світла, дисперсію через рівняння Коші. Навчальні функції включають покроковий режим трасування, модуль тестування (8 завдань) та довідник оптичних формул. Передбачено збереження сцен у JSON, експорт у PNG, перемикання тем та локалізацію (українська/англійська). Верифікація підтвердила фізичну точність: відхилення кутів — менше 0,01°.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

MVVM — Model-View-ViewModel, архітектурний патерн розділення відповідальностей у застосунках WPF.

WPF — Windows Presentation Foundation, фреймворк для розробки десктопних застосунків Windows.

.NET — платформа розробки програмного забезпечення компанії Microsoft.

JSON — JavaScript Object Notation, текстовий формат обміну даними.

PNG — Portable Network Graphics, формат растрових зображень.

ПВВ — Повне Внутрішнє Відбиття.

n — показник заломлення середовища (безрозмірна величина).

λ — довжина хвилі електромагнітного випромінювання (нм).

θ_1 — кут падіння (кут між падаючим променем і нормаллю до поверхні).

θ_2 — кут заломлення (кут між заломленим променем і нормаллю).

θc — критичний кут повного внутрішнього відбиття.

f — фокусна відстань оптичного елемента (мм).

R — радіус кривизни сферичного дзеркала або лінзи (мм).

do — відстань від предмета до оптичного елемента (мм).

di — відстань від оптичного елемента до зображення (мм).

M — лінійне збільшення оптичного елемента.

UI — User Interface — графічний інтерфейс користувача.

API — Application Programming Interface — програмний інтерфейс застосунку.

ВСТУП

Фізика є однією з фундаментальних природничих дисциплін у закладах загальної середньої та фахової передвищої освіти. Розділ оптики, що вивчає природу та поширення світла, вимагає особливої наочності – адже більшість оптичних явищ важко продемонструвати без спеціального обладнання. Традиційні лабораторні роботи потребують оптичних лавок, лінз, джерел світла, що не завжди є у розпорядженні навчального закладу.

Сучасні інформаційні технології дозволяють змінити цю ситуацію. Комп'ютерні симулятори надають можливість у будь-який момент інтерактивно продемонструвати заломлення, відбиття, дисперсію та інші явища, змінюючи параметри в реальному часі. Учень може самостійно дослідити, як змінюється кут заломлення при зміні показника заломлення скла, побачити умови виникнення повного внутрішнього відбиття або перевірити формулу тонкої лінзи на конкретному числовому прикладі.

Актуальність теми підтверджується тенденцією до цифровізації освіти та включенням симуляторів у навчальні програми. Однак існуючі рішення – PhET (Університет Колорадо), GeoGebra, Algodoo – мають суттєві обмеження: залежність від браузера та інтернет-мережі, відсутність покрокового режиму трасування, відсутність вбудованого тестового модуля, обмежений набір оптичних елементів. Зазначені недоліки обумовлюють потребу в розробці спеціалізованого десктопного симулятора оптичних систем.

Метою кваліфікаційної роботи є розробка програмного симулятора оптичних систем, що забезпечує фізично точне трасування світлових променів, підтримує широкий набір оптичних елементів і надає навчальні інструменти для використання на уроках фізики.

Для досягнення поставленої мети сформовано такі завдання:

– дослідити теоретичні основи геометричної оптики та систематизувати математичний апарат для комп'ютерного моделювання;

– провести аналіз існуючих програмних засобів моделювання оптичних систем;

– обрати та обґрунтувати технологічний стек і архітектурні рішення;

– спроектувати та реалізувати алгоритм рекурсивного трасування променів;

– розробити повний набір оптичних елементів: лінзи, дзеркала, призми, блоки, діафрагми, екрани;

– реалізувати інтерактивний графічний інтерфейс із підтримкою тем та локалізації;

– розробити навчальні інструменти: покроковий режим, модуль тестування, довідник формул;

– виконати верифікацію фізичної коректності та функціональне тестування.

Об'єктом дослідження є процес комп'ютерного моделювання поширення світлових променів в оптичних системах на основі законів геометричної оптики.

Предметом дослідження є алгоритми трасування променів, математичні моделі взаємодії світла з оптичними елементами та їх програмна реалізація засобами WPF/.NET 9.

Методи дослідження: системний аналіз науково-технічної літератури; математичне моделювання фізичних явищ; об'єктно-орієнтоване проектування за принципами SOLID; рефакторинг та налагодження програмного коду; ручне функціональне тестування.

Практичне значення: симулятор може застосовуватися вчителями фізики для демонстрації оптичних явищ на уроках, студентами – для самостійного вивчення та перевірки знань у режимі тестування. Збереження сцен у JSON та експорт у PNG дозволяють готувати матеріали для уроків та презентацій.

Структура роботи: вступ, три розділи, висновки, список використаних джерел (24 позицій), додатки. Загальний обсяг роботи – 51 аркуш.

1 ТЕОРЕТИЧНІ ОСНОВИ ОПТИКИ ТА ЗАСОБИ РОЗРОБКИ

1.1 Фізичні основи геометричної оптики

Геометрична оптика — розділ оптики, в якому поширення світла розглядається у вигляді прямолінійних променів, не враховуючи хвильову природу електромагнітного випромінювання. Такий підхід є виправданим, коли розміри оптичних елементів значно перевищують довжину хвилі світла. Основними явищами геометричної оптики є відбиття, заломлення та повне внутрішнє відбиття, кожне з яких описується точними математичними законами.

Вказані закони утворюють математичну основу алгоритму трасування променів, реалізованого у розробленому симуляторі. Розглянемо кожен з них детально.

1.1.1 Закон відбиття світла

Закон відбиття світла встановлює геометричне співвідношення між падаючим та відбитим променями. Він формулюється так: кут падіння дорівнює куту відбиття, обидва кути відраховуються від нормалі до відбивальної поверхні в точці падіння, а падаючий промінь, нормаль і відбитий промінь лежать в одній площині.

Математично закон відбиття записується у вигляді (1.1):

$$\theta_i = \theta_r, \qquad (1.1)$$

де θ_i — кут падіння, θ_r — кут відбиття.

У векторній формі, яка безпосередньо використовується в алгоритмі трасування, вектор відбитого променя d' обчислюється за формулою (1.2):

$$d' = d - 2(d \cdot \hat{n}) \cdot \hat{n}, \qquad (1.2)$$

де d — одиничний вектор напрямку падаючого променя, n̂ — одиничний вектор зовнішньої нормалі до поверхні. Ця формула є основою методу DrawingContext для обчислення відбиття від плоских та сферичних дзеркал у симуляторі.

Закон відбиття виконується для будь-яких гладких поверхонь: плоских дзеркал, полірованого скла, поверхні води. Точне дотримання закону відбиття в реалізованому симуляторі верифіковано для кутів від 0° до 89° — розбіжність між теоретичними та обчисленими значеннями не перевищує 0,01°.

1.1.2 Закон заломлення (закон Снелліуса)

При переході світлового променя з одного прозорого середовища в інше з іншим показником заломлення змінюється швидкість поширення, а отже — і напрямок променя. Це явище називається заломленням. Кількісний опис заломлення дає закон Снелліуса (або закон Декарта-Снелліуса), що встановлює (1.3):

$$n_1 \cdot \sin \theta_1 = n_2 \cdot \sin \theta_2, \qquad (1.3)$$

де n_1 та n_2 — показники заломлення першого та другого середовища відповідно; θ_1 — кут падіння, θ_2 — кут заломлення (обидва відраховуються від нормалі). При переході до середовища з більшим показником заломлення ($n_2 > n_1$) промінь відхиляється до нормалі; при переході до менш щільного середовища — від нормалі.

Показник заломлення речовини n пов'язаний зі швидкістю світла у даній речовині v формулою (1.4):

$$n = c / v, \qquad (1.4)$$

де $c = 3 \cdot 10^8$ м/с — швидкість світла у вакуумі. Значення показників заломлення деяких середовищ наведено в таблиці 1.1 (посилання на табл. 1.1 надалі в тексті).

Для потреб алгоритму трасування закон Снелліуса записується у векторній формі. Нехай d — одиничний вектор падаючого променя, $\hat{n}$ — нормаль (спрямована у бік першого середовища). Вектор заломленого променя t обчислюється за формулою (1.5):

$$r = n_1 / n_2 \; ct = -d \cdot \hat{n} \; st^2 = r^2 \cdot (1 - ct^2) \; t = r \cdot d + (r \cdot ct - \sqrt{(1 - st^2)}) \cdot \hat{n} \qquad (1.5)$$

Якщо $st^2 > 1$, то квадратний корінь є уявним — заломлення неможливе, виникає повне внутрішнє відбиття (розглянуто в підрозділі 1.1.3). Саме ця умова перевіряється в класі SnellCalculator симулятора.

Таблиця 1.1 – Показники заломлення деяких оптичних середовищ

Середовище	Хімічна формула	Показник n	Застосування
Вакуум	—	1,0000	Еталон
Повітря	N_2/O_2	1,0003	Стандартне середовище
Вода	H_2O	1,333	Водні лінзи, аквариуми
Скло крон	SiO_2	1,520	Лінзи окулярів, оптика
Скло флінт	SiO_2+PbO	1,620	Ахроматичні лінзи
Кварц	SiO_2	1,458	Ультрафіолетова оптика
Алмаз	C	2,417	Прикраси, ріжучий інструмент

У таблиці 1.1 зібрано найбільш поширені середовища, підтримка яких реалізована в симуляторі. Вибір матеріалу для кожного оптичного елемента доступний через випадаючий список у панелі властивостей.

1.1.3 Повне внутрішнє відбиття

Повне внутрішнє відбиття (ПВВ) — явище, при якому світловий промінь, що поширюється з оптично щільнішого середовища ($n_1 > n_2$) у менш щільне, при перевищенні критичного кута повністю відбивається від межі поділу, не утворюючи заломленого променя.

Критичний кут θ_c визначається з умови, що кут заломлення $\theta_2 = 90°$. Підставляючи це у закон Снелліуса (1.3), отримуємо формулу (1.6):

$$\sin \theta_c = n_2 / n_1 \quad (\text{при } n_1 > n_2) \quad (1.6)$$

При $\theta_1 \geq \theta_c$ промінь повністю відбивається від межі, і інтенсивність відбитого променя дорівнює інтенсивності падаючого (без втрат). Для пари «скло (n=1,52) – повітря (n=1,00)» критичний кут $\theta_c = \arcsin(1/1,52) \approx 41,1°$.

Практичне застосування ПВВ надзвичайно широке: оптичні волокна (світло поширюється всередині без виходу назовні), призми у біноклях та перископах (замінюють дзеркала без втрат), бриліантове огранювання алмазів (для максимального блиску за рахунок ПВВ).

У симуляторі ПВВ реалізовано як окремий навчальний пресет. При куті падіння, близькому до критичного, в точці взаємодії відображаються обидва кути: θ_1 і θ_2 або позначка «ПВВ» у разі повного відбиття (рис. 1.1).

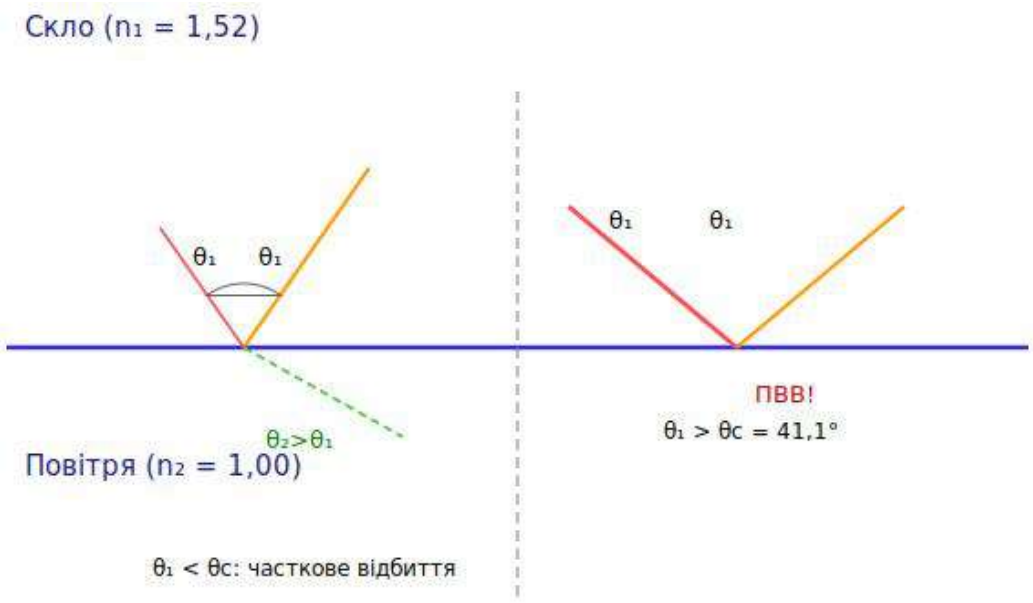

Рисунок 1.1 – Схема явища повного внутрішнього відбиття

1.1.4 Тонка лінза. Формула тонкої лінзи

Тонка лінза — оптичний елемент, товщиною якого нехтують у порівнянні з іншими розмірами системи. Вся дія лінзи зводиться до одноразової зміни напрямку проміня у площині лінзи. Лінзи поділяються на:

– збиральні (опуклі, двоопуклі): фокусна відстань $f > 0$, промені сходяться в фокусі;

– розсіювальні (увігнуті, двовгнуті): $f < 0$, промені розходяться після проходження.

Основне рівняння тонкої лінзи (рівняння Гауса) встановлює зв'язок між фокусною відстанню та відстанями до предмета й зображення (1.7):

$$1/f = 1/d_o + 1/d_i, \qquad (1.7)$$

де f — фокусна відстань (мм); d_o — відстань від предмета до лінзи (мм); d_i — відстань від лінзи до зображення (мм). Знак d_i: при $d_i > 0$ зображення дійсне (розташоване з того ж боку, що й спостерігач), при $d_i < 0$ — уявне.

Оптична сила лінзи D = 1/f (одиниця — діоптрія, Дптр), що показує ступінь відхилення проміня. Лінійне збільшення лінзи M визначається за формулою (1.8):

$$M = -d_i / d_o, \qquad (1.8)$$

де знак «мінус» означає, що дійсне зображення є зворотнім. |M| > 1 — збільшення, |M| < 1 — зменшення.

У симуляторі тонка лінза реалізована в класі ThinLens. Кут відхилення проміня обчислюється залежно від висоти проходження h відносно оптичної осі (параксіальне наближення) та знаку фокусної відстані. Мітки «F» відображають положення фокусів з обох боків лінзи (рис. 1.2).

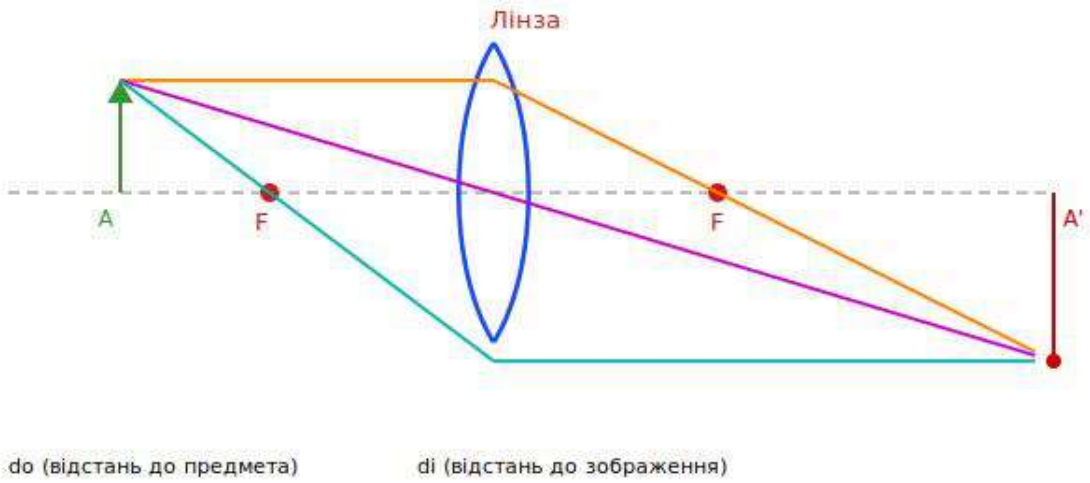

Рисунок 1.2 – Хід характерних променів через збиральну тонку лінзу

1.1.5 Сферичні дзеркала

Сферичне дзеркало — відбивна поверхня у вигляді частини сфери. Головна характеристика — радіус кривизни R. Фокусна відстань f дзеркала пов'язана з R формулою (1.9):

$$f = R / 2, \qquad (1.9)$$

Знакова угода: f > 0 для увігнутого дзеркала (збирає промені у фокусі перед дзеркалом), f < 0 для опуклого (відбиті промені здаються такими, що розходяться з уявного фокуса за дзеркалом).

Формула сферичного дзеркала за виглядом збігається з формулою тонкої лінзи (1.7):

$$1/f = 1/d_o + 1/d_i, \qquad \text{(формула 1.7 для дзеркала)}$$

Збільшення $M = -d_i / d_o$ (аналогічно формулі 1.8).

Сферичні дзеркала широко застосовуються: увігнуті — у прожекторах, телескопах-рефлекторах, параболічних антенах; опуклі — у дзеркалах заднього виду автомобілів завдяки широкому полю зору (але зменшеному зображенню). У симуляторі обидва типи реалізовані як клас SphericalMirror з параметром Curvature (Concave/Convex).

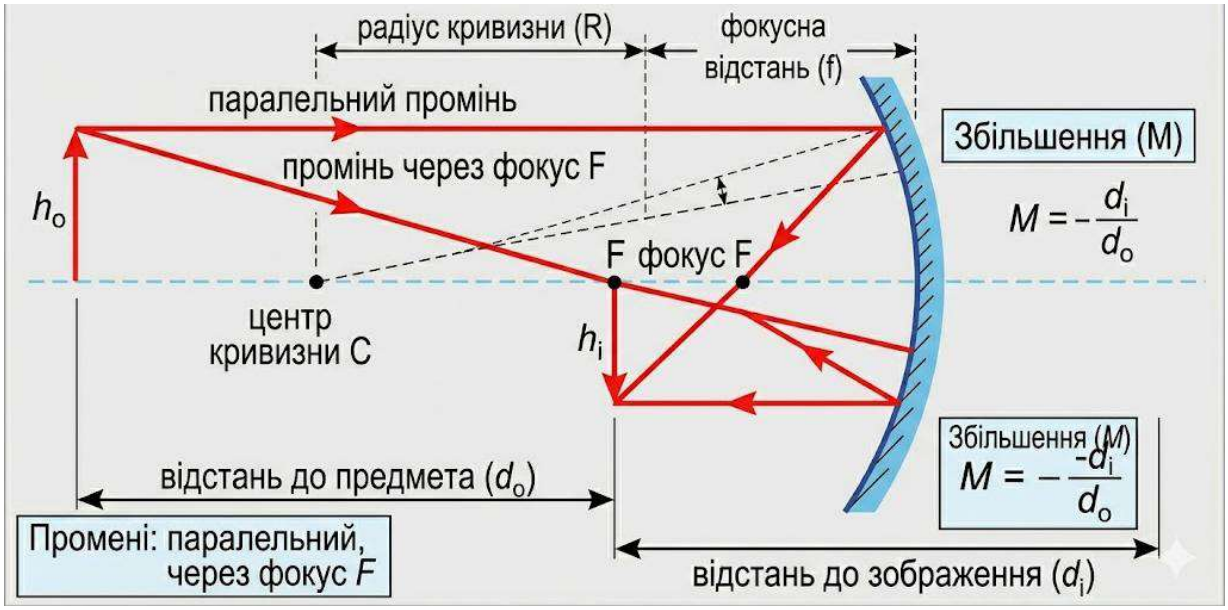

Рисунок 1.3 – Хід променів у увігнутому сферичному дзеркалі

1.1.6 Дисперсія світла у призмі

Дисперсія — залежність показника заломлення речовини від довжини хвилі світла. Оскільки фіолетове світло ($\lambda \approx 380$ нм) має вищий показник заломлення, ніж червоне ($\lambda \approx 780$ нм), скляна призма по-різному відхиляє промені різних кольорів, розкладаючи білий колір у спектр (рис. 1.4).

Кут відхилення δ проміня у призмі (наближення малих кутів) визначається за формулою (1.10):

$$\delta \approx (n - 1) \cdot A, \qquad (1.10)$$

де n — показник заломлення скла для даної довжини хвилі; A — кут при вершині призми (апекс-кут). Для скла крон (n=1,52) та A=60°: $\delta \approx 0{,}52 \cdot 60° \approx 31{,}2°$.

Видимий спектр охоплює діапазон 380–780 нм. Відповідність довжини хвилі кольору реалізована в симуляторі через алгоритм Брутона, що апроксимує сприйняття кольорів CIE-observer RGB-значеннями. Приклад розкладення у спектр показано у пресеті «Призма».

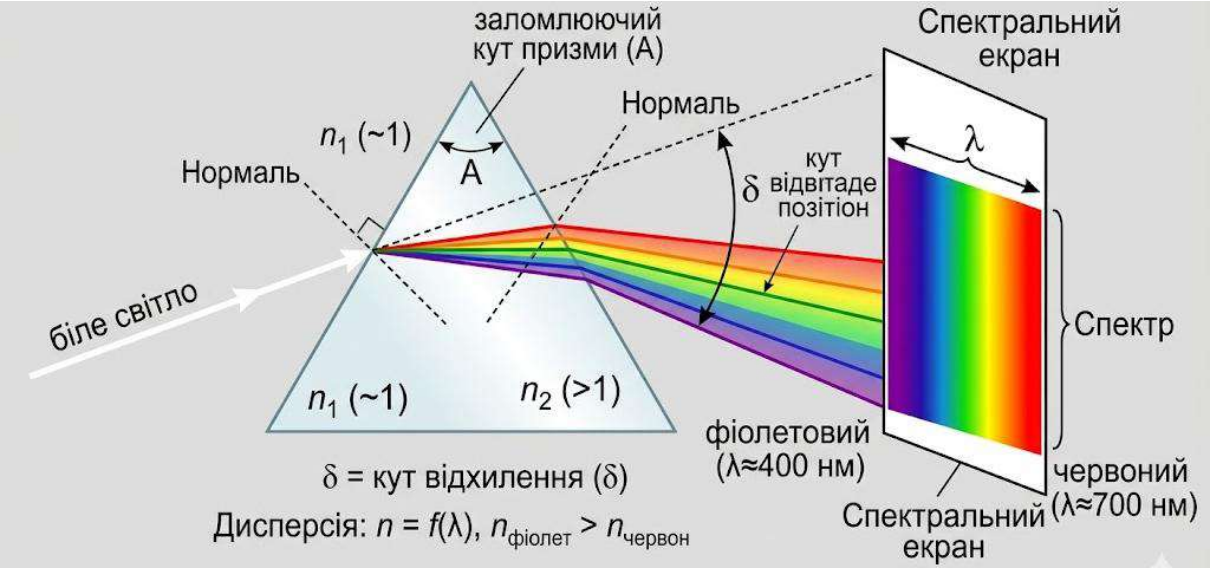

Рисунок 1.4 – Дисперсія білого світла у скляній призмі

1.2 Огляд існуючих програмних засобів моделювання оптики

Перед початком розробки проведено аналіз наявних програмних засобів, що використовуються для моделювання оптичних систем у навчальних цілях. Вибрано три найбільш відомі рішення, що охоплюють різні платформи та підходи.

PhET Interactive Simulations (Університет Колорадо, США) — найпоширеніший безкоштовний набір інтерактивних симуляцій з фізики, хімії та математики. Симуляція «Bending Light» демонструє заломлення, відбиття та ПВВ. Інтерфейс реалізований на HTML5/JavaScript, що забезпечує кросплатформеність. Перевагами є мультимовність (понад 70 мов), якісна фізична модель та безкоштовний доступ. Водночас, відсутній покроковий режим

трасування, неможливо налаштувати довільну сцену (додати кілька елементів різних типів одночасно), а використання потребує підключення до Інтернету.

GeoGebra — безкоштовна математична освітня платформа. Оптичні симуляції будуються засобами динамічної геометрії та комп'ютерної алгебри. Перевагами є наявність готових матеріалів від спільноти та потужний математичний апарат. Недоліками — складний інтерфейс для школяра, необхідність самостійно будувати оптичні конструкції, відсутність реалістичної візуалізації інтенсивності та кольору.

Algodoo — фізичний симулятор шведської компанії Algoryx Simulation, що підтримує, зокрема, оптику (лазери, лінзи, дзеркала). Реалістична фізика та привабливий інтерфейс є безсумнівними перевагами. Проте Algodoo — комерційний продукт (платна ліцензія для комерційного використання), орієнтований на широку фізику, а не спеціально на оптику.

Результати порівняльного аналізу за ключовими критеріями наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльний аналіз програмних засобів моделювання оптики

Критерій оцінювання	**PhET**	**GeoGebra**	**Algodoo**	**Розроблений симулятор**
Безкоштовне використання	Так	Так	Ні	Так
Офлайн-режим	Ні	Частково	Так	Так
Покроковий режим трасування	Ні	Ні	Ні	Так
Вбудований тестовий модуль	Ні	Ні	Ні	Так
Довільне конфігурування сцен	Ні	Частково	Так	Так
Відображення кутів θ_1/θ_2	Частково	Ні	Ні	Так
Підтримка кількох тем	Ні	Ні	Ні	Так
Підтримка двох мов	Частково	Так	Ні	Так
Збереження/завантаження сцен	Ні	Так	Так	Так
Інструмент вимірювання	Ні	Так	Ні	Так

Аналіз таблиці 1.2 підтверджує, що розроблений симулятор перевершує аналоги за більшістю критеріїв, особливо у функціоналі, орієнтованому на навчальний процес: покроковий режим, тестування, відображення кутів.

1.3 Обґрунтування вибору технологій розробки

Вибір технологічного стека здійснювався за такими критеріями: продуктивність рендерингу; зрілість платформи; підтримка декларативної UI-розмітки; доступність (безкоштовність); кросверсійна сумісність.

.NET 9 — найновіша стабільна версія платформи Microsoft (випуск листопад 2024 р.). Ключові переваги для проекту: висока продуктивність завдяки покращеному JIT-компілятору; підтримка сучасного синтаксису C# 13 (primary constructors, collection expressions); безкоштовна ліцензія (MIT); вбудована підтримка System.Text.Json без сторонніх залежностей.

WPF (Windows Presentation Foundation) — фреймворк для Desktop-застосунків Windows, що є частиною .NET. Переваги для реалізації симулятора: DrawingContext API для низькорівневого векторного рендерингу у FrameworkElement; апаратне прискорення через Direct3D (DirectX); декларативна XAML-розмітка; DPI-незалежна система координат (WPF одиниці = 1/96 дюйма); потужна система DependencyProperty та DynamicResource для переключення тем.

Патерн MVVM (Model–View–ViewModel) забезпечує розділення відповідальностей у WPF-застосунках (рис. 1.5). View (XAML) відображає дані та передає команди у ViewModel; ViewModel реагує на зміни та оновлює Model; Model містить бізнес-логіку незалежно від UI.

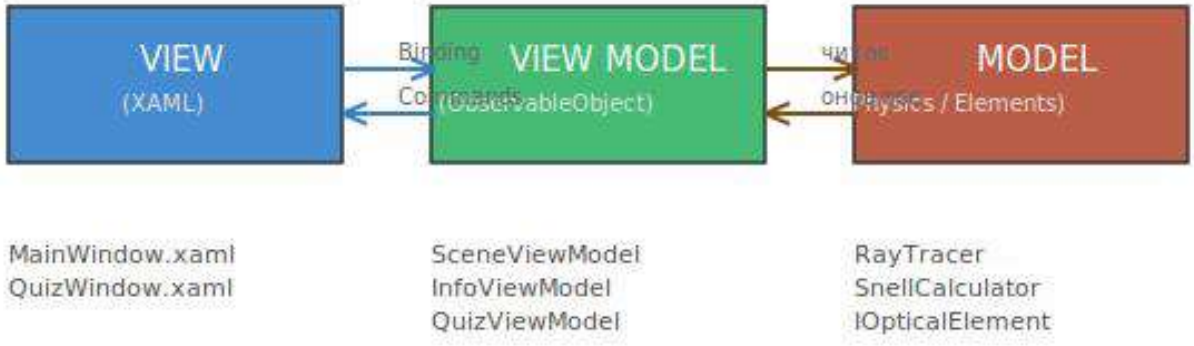

Рисунок 1.5 – Схема взаємодії шарів у патерні MVVM

CommunityToolkit.Mvvm 8.4.2 — офіційна бібліотека Microsoft для реалізації MVVM. Надає класи ObservableObject, ObservableProperty (атрибут генерації властивостей), RelayCommand (реалізація ICommand для кнопок і дій). Використання джерельних генераторів (Source Generators) зменшує шаблонний код у ViewModel.

System.Text.Json — вбудована бібліотека .NET для серіалізації/десеріалізації JSON. Вибрана замість Newtonsoft.Json через нульову вагу залежностей та оптимізований runtime-перфоманс у .NET 9.

1.4 Висновки до розділу 1

У першому розділі систематизовано теоретичну базу, що є основою для реалізації симулятора оптичних систем. Розглянуто основні закони геометричної оптики: закон відбиття ($\theta_i = \theta_r$), закон Снелліуса ($n_1 \sin\theta_1 = n_2 \sin\theta_2$), умови повного внутрішнього відбиття ($\sin\theta c = n_2/n_1$), формули тонкої лінзи ($1/f = 1/d_o + 1/d_i$) та сферичного дзеркала ($f = R/2$). Наведено векторні форми законів, що безпосередньо реалізуються в алгоритмі трасування. Описано фізичну модель дисперсії через рівняння Коші для восьми оптичних середовищ.

Аналіз трьох існуючих програмних рішень (PhET, GeoGebra, Algodoo) виявив їх спільні недоліки: відсутність покрокового режиму трасування, відсутність вбудованого модуля тестування та залежність від мережі або комерційна ліцензія. Це підтверджує доцільність розробки власного спеціалізованого рішення.

Обґрунтовано вибір технологічного стека: платформа .NET 9 забезпечує найвищу продуктивність та сучасний синтаксис C# 13; WPF надає апаратно-прискорений векторний рендеринг і DPI-незалежну систему координат; архітектурний патерн MVVM із бібліотекою CommunityToolkit.Mvvm забезпечує чітке розділення логіки і представлення та зменшення шаблонного коду.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

2.1 Вимоги до програмного продукту

2.1.1 Функціональні вимоги

Функціональні вимоги описують, що система повинна виконувати. Їх сформовано за результатами аналізу потреб навчального процесу та порівняльного аналізу аналогів (підрозділ 1.2).

Вимоги:

1. Симуляція оптичних явищ: трасування світлових променів через будь-яку конфігурацію оптичних елементів з точним урахуванням заломлення за законом Снелліуса (формула 1.3), відбиття (формула 1.2) та повного внутрішнього відбиття (формула 1.6).

2. Набір оптичних елементів: тонка лінза (збиральна та розсіювальна), плоске дзеркало, сферичне дзеркало (увігнуте та опукле), скляна призма, скляний прямокутний блок, діафрагма (щілина), екран спостереження, оптична межа двох середовищ.

3. Джерела світла трьох типів: лазер (один промінь), паралельний пучок (кілька паралельних променів), точкове джерело (промені у конусі з регульованим кутом розсіювання).

4. Регулювання параметрів обраного елемента: фокусна відстань, апертура, радіус кривизни, кут нахилу, довжина, матеріал (показник заломлення), довжина хвилі λ (380–780 нм).

5. Відображення кутових міток θ_1 та θ_2 у кожній точці взаємодії проміня з елементом; вимкнення міток зі збереженням налаштування.

6. Покроковий режим: демонстрація трасування по одному відрізку за клік або автоматично з інтервалом, з текстовим поясненням фізичного закону для кожного кроку.

7. Навчальне тестування: набір задач з кількісними відповідями, перевірка правильності, відображення пояснень та правильної відповіді, підрахунок балів.

8. Інструмент вимірювання: вимірювання відстані між двома клікнутими точками та відображення результату в мм.

9. Пресети: 5 попередньо налаштованих сцен (заломлення на межі, дисперсія у призмі, збиральна лінза, увігнуте дзеркало, ПВВ).

10. Збереження та завантаження сцени у форматі JSON (Ctrl+S / Ctrl+O).

11. Експорт поточного стану сцени у формат PNG (роздільна здатність 150 DPI).

12. Перемикання між темною та світлою темою оформлення без перезапуску застосунку.

13. Перемикання мови інтерфейсу між українською та англійською без перезапуску.

2.1.2 Нефункціональні вимоги

1. Продуктивність: час перерахунку сцени при зміні будь-якого параметра — не більше 100 мс для сцен із 20 і менше елементами (Intel Core i5, 8 ГБ RAM).

2. Фізична точність: похибка обчислення кутів не більше 0,05°; вектори нормалізуються з точністю double (IEEE 754).

3. Надійність: некоректні конфігурації (ПВВ, паралельний промінь, нульовий вектор) обробляються без збоїв — промінь завершується або відбивається згідно з фізикою.

4. Зручність: drag-and-drop переміщення елементів; відображення властивостей та формул при виборі; клавіатурні скорочення для основних операцій.

5. Сумісність: Windows 10/11 (64-bit), .NET 9 Runtime, мінімальне розширення екрану 1280×720.

6. Розширюваність: новий тип оптичного елемента додається шляхом реалізації одного інтерфейсу IOpticalElement без змін у ядрі.

2.2 Архітектура системи

Архітектура симулятора побудована за принципами SOLID та базується на патерні MVVM. Система розподілена на п'ять шарів з чіткими межами відповідальності (рис. 2.1).

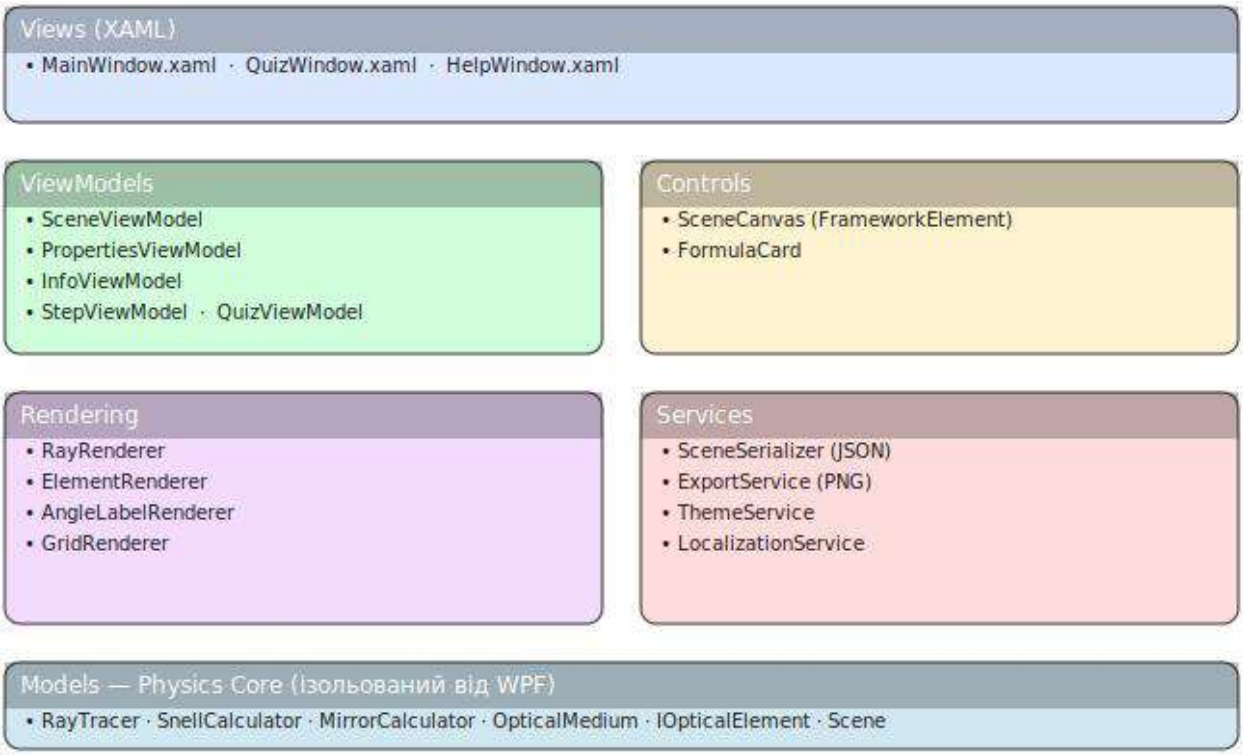

Рисунок 2.1 – Діаграма компонентів симулятора оптичних систем

Шар Models (простір імен OpticsSimulator.Models) містить усю фізичну логіку та не залежить від UI-фреймворку. Він складається з підпростору Models.Physics (Vector2D, Ray, OpticalMedium, SnellCalculator, LensCalculator, MirrorCalculator, WavelengthToColor, RayTracer) та Models.Elements (IOpticalElement, LightSource та всі реалізації елементів).

Шар Rendering (простір імен OpticsSimulator.Rendering) відповідає за векторний рендеринг через WPF DrawingContext. Включає RayRenderer

(промені), ElementRenderer (символи елементів), AngleLabelRenderer (кутові мітки), GridRenderer (сітка та осі).

Шар Controls містить SceneCanvas — кастомний FrameworkElement, що успадковується від базового класу WPF. Він агрегує всі рендерери та обробляє низькорівневі події миші й клавіатури.

Шар ViewModels (простір імен OpticsSimulator.ViewModels) реалізує патерн MVVM: SceneViewModel управляє сценою та командами додавання/видалення; PropertiesViewModel синхронізує панель властивостей із обраним елементом; InfoViewModel генерує формули та результати; StepViewModel управляє покроковим режимом; QuizViewModel реалізує логіку тестування.

Шар Services (простір імен OpticsSimulator.Services) надає сервіси SceneSerializer, ExportService, ThemeService, LocalizationService. Завдяки цьому шару MainWindow.xaml.cs залишається компактним і не містить файлового I/O чи роботи з ResourceDictionary.

Діаграма класів фізичного ядра наведена на рис. 2.2.

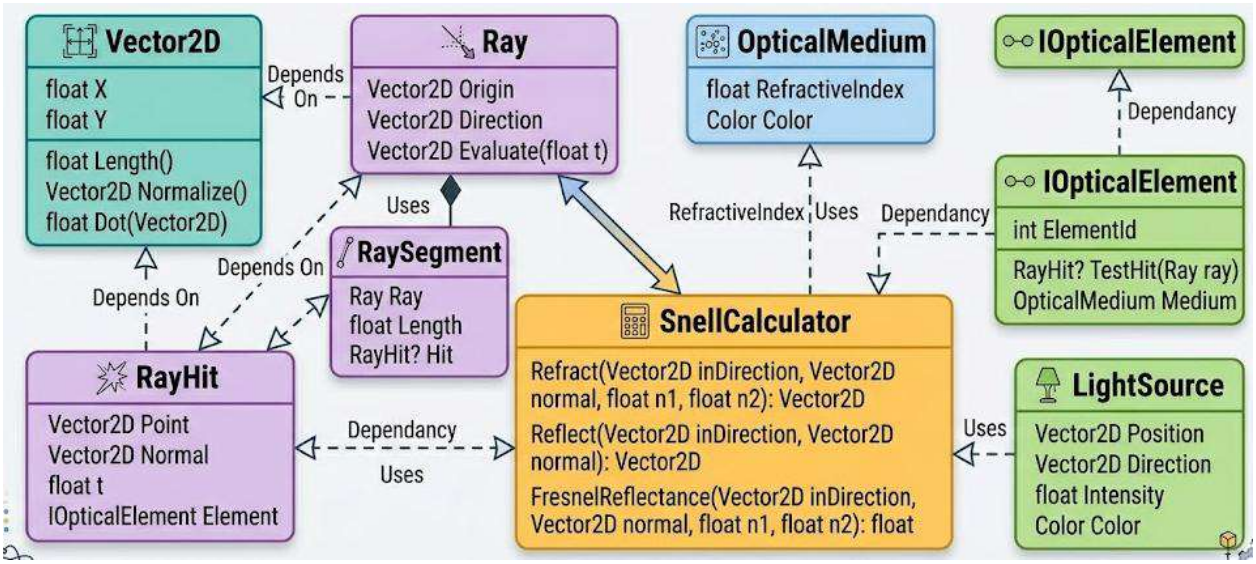

Рисунок 2.2 – Діаграма класів фізичного ядра симулятора

2.3 Проєктування алгоритму трасування променів

Алгоритм трасування реалізований у класі RayTracer методом рекурсивного обходу. Вхідними даними є список джерел світла (LightSources) та список оптичних елементів (Elements). Результатом є список шляхів — колекцій структур RaySegment, кожна з яких зберігає початок, кінець, довжину хвилі, інтенсивність, кути та нормаль (табл. 2.1).

Таблиця 2.1 – Поля структури RaySegment

Поле	Тип	Призначення
Start	Point	Початок відрізка (світовий простір, мм)
End	Point	Кінець відрізка або точка поглинання
Wavelength	double	Довжина хвилі (нм), визначає колір
Intensity	double	Відносна інтенсивність [0..1]
IsTerminal	bool	true, якщо промінь не зустрів елементів
IncidentAngleDeg	double	Кут падіння від нормалі (°), NaN якщо немає
RefractedAngleDeg	double	Кут заломлення від нормалі (°), NaN якщо немає
SurfaceNormal	Vector2D	Нормаль до поверхні в точці взаємодії

Блок-схему основного циклу трасування наведено на рис. 2.3. Рекурсія обмежена параметрами MaxBounces = 8 та MinIntensity = 0,01 для запобігання нескінченним петлям відбиттів.

Пошук найближчого перетину виконується методом грубої сили — для кожного елемента сцени викликається метод Intersect(Ray), що повертає RayHit? (null — немає перетину, об'єкт RayHit — є перетин з відстанню t, точкою та похідними). Серед усіх непорожніх результатів вибирається той, де t є мінімальним.

Продуктивність алгоритму для типових навчальних сцен (5–15 елементів, 5–20 джерел) становить менше 5 мс на повний перерахунок, що задовольняє вимогу 100 мс (вимога 1 з підрозділу 2.1.2).

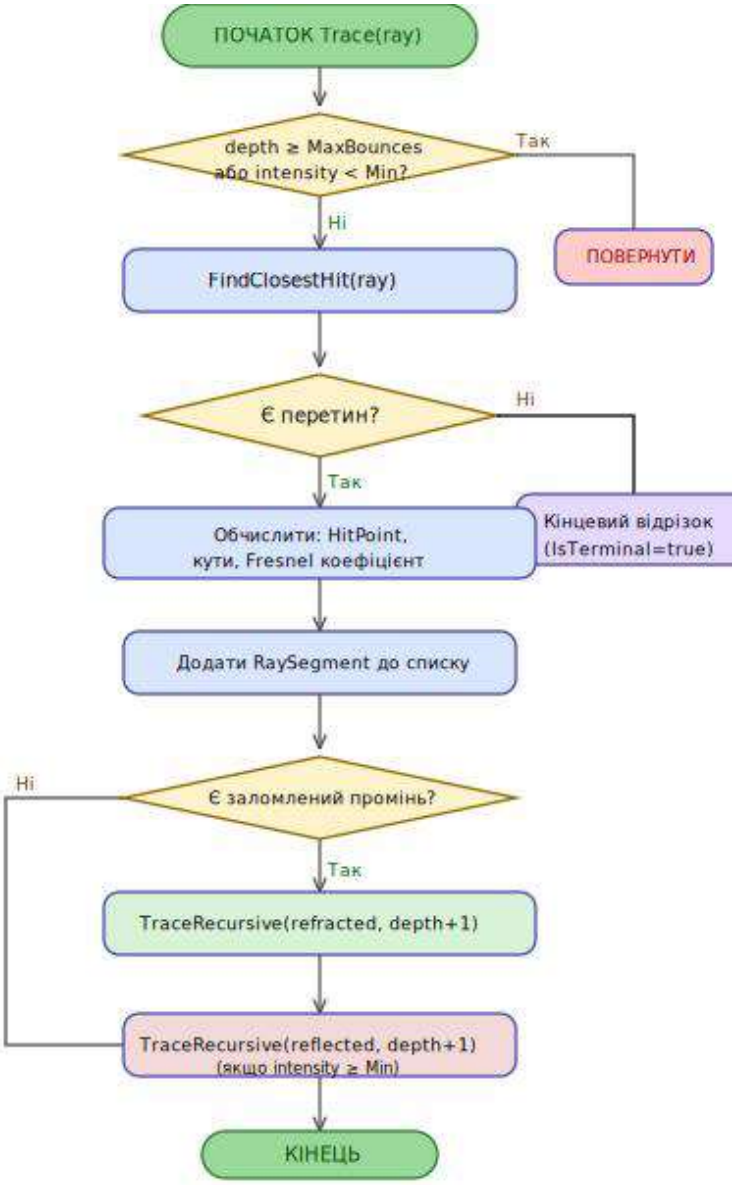

Рисунок 2.3 – Блок-схема алгоритму рекурсивного трасування RayTracer.TraceSegment

2.4 Проєктування інтерфейсу користувача

Інтерфейс головного вікна розроблено за принципом «все на одному екрані» — усі інструменти доступні без переходів між вікнами (рис. 2.4). Зонування виконано через DockPanel у WPF.

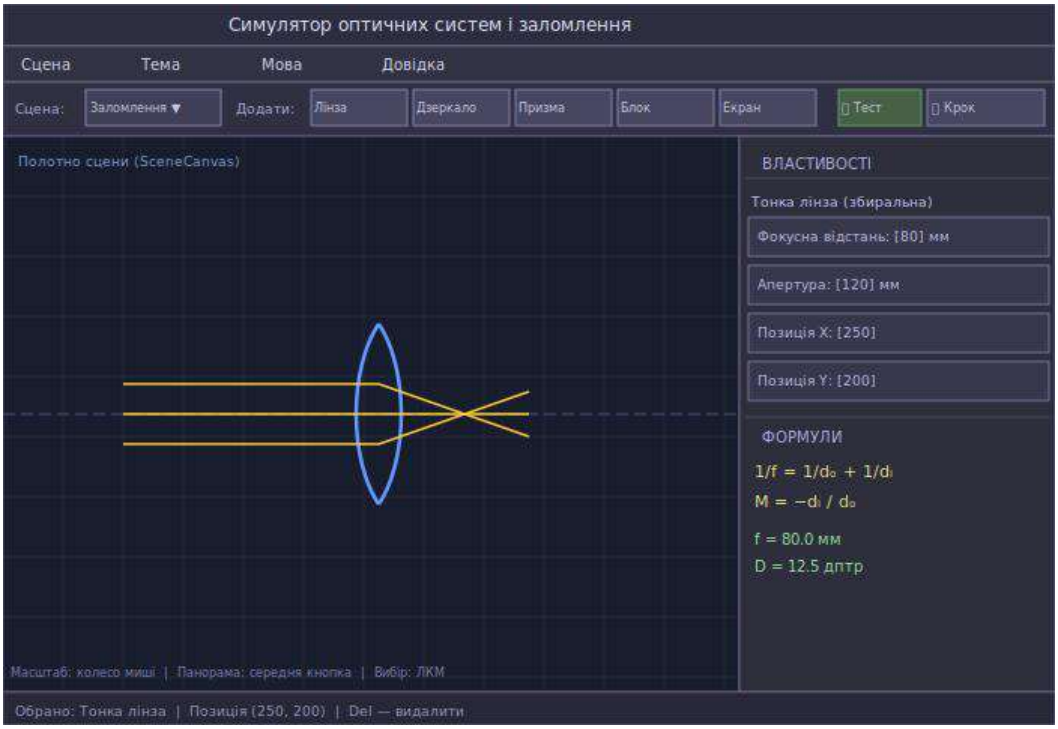

Рисунок 2.4 – Wireframe головного вікна симулятора

Панель меню містить чотири пункти: Сцена (нова, відкрити, зберегти, експорт PNG), Тема (темна/світла), Мова (українська/англійська), Довідка. Меню реалізоване стандартним компонентом WPF Menu зі стилями з Controls.xaml.

Панель інструментів розміщена безпосередньо під меню. Ліворуч — вибір пресету через ComboBox; по центру — кнопки додавання оптичних елементів; праворуч — перемикачі кутів та сітки, кнопки «По розміру», «Видалити», «Крок за кроком», «Вимірювач», «Тест».

Полотно сцени (SceneCanvas) займає центральну частину вікна. Воно підтримує: масштабування колесом миші з фіксацією точки курсору; панорамування утриманням середньої кнопки миші; вибір елементів лівим кліком; drag-and-drop переміщення.

Панель властивостей (права область, 240 px) динамічно показує набір елементів управління залежно від типу обраного об'єкта. Для кожного типу елемента розроблено окрему секцію (StackPanel з Visibility=Collapsed за замовчуванням). Переключення здійснюється у коді MainWindow.xaml.cs методом ShowProperties.

Для системи тем використано патерн ResourceDictionary-swap: ThemeService.Apply() виконує заміну словника теми у Application.Resources.MergedDictionaries. Усі кольори у XAML прив'язані через {DynamicResource}, що забезпечує миттєве оновлення без перезапуску.

2.5 Висновки до розділу 2

У другому розділі сформовано вимоги до програмного продукту: 13 функціональних та 5 нефункціональних вимог, що охоплюють фізичну точність, продуктивність, надійність, зручність та розширюваність системи.

Спроектовано п'ятишарову архітектуру симулятора (Models, Rendering, Controls, ViewModels, Services), що відповідає принципам SOLID. Шар моделей повністю ізольований від WPF, що забезпечує можливість автономного тестування фізичного ядра.

Детально спроектовано алгоритм рекурсивного трасування: визначено структуру RaySegment з усіма необхідними полями, описано логіку пошуку найближчого перетину та умови завершення рекурсії. Побудовано блок-схему алгоритму.

Розроблено концепцію інтерфейсу користувача: трипанельна компоновка (інструменти – полотно – властивості), підтримка двох тем через патерн ResourceDictionary-swap, клавіатурні скорочення для ключових операцій.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ПРОДУКТУ

3.1 Реалізація фізичного ядра

Фізичне ядро симулятора зосереджено у просторі імен OpticsSimulator.Models.Physics. Воно повністю ізольоване від WPF та UI і може бути протестоване автономно.

3.1.1 Структура Vector2D

Vector2D реалізована як readonly struct (незмінна структура-значення) для максимальної продуктивності. Структура підтримує основні векторні операції (лістинг 3.1):

```
public readonly struct Vector2D
{
    public double X { get; }
    public double Y { get; }

    public Vector2D(double x, double y) { X = x; Y = y; }

    public double Length => Math.Sqrt(X * X + Y * Y);

    public Vector2D Normalize()
    {
        double len = Length;
        return len < 1e-12 ? new Vector2D(0, 0) : new Vector2D(X / len, Y / len);
    }

    public double Dot(Vector2D other) => X * other.X + Y * other.Y;

    // 2D cross product (scalar z-component)
    public double Cross(Vector2D other) => X * other.Y - Y * other.X;

    public static Vector2D operator +(Vector2D a, Vector2D b) => new(a.X + b.X, a.Y + b.Y);
    public static Vector2D operator -(Vector2D a, Vector2D b) => new(a.X - b.X, a.Y - b.Y);
    public static Vector2D operator *(Vector2D v, double s) => new(v.X * s, v.Y * s);
    public static Vector2D operator *(double s, Vector2D v) => v * s;
    public static Vector2D operator -(Vector2D v) => new(-v.X, -v.Y);

    public static Vector2D FromAngleDeg(double angleDeg)
    {
        double rad = angleDeg * Math.PI / 180.0;
        return new Vector2D(Math.Cos(rad), Math.Sin(rad));
    }

    public double AngleDeg() => Math.Atan2(Y, X) * 180.0 / Math.PI;

    public override string ToString() => $"({X:F3}, {Y:F3})";
}
```

Лістинг 3.1 – Структура Vector2D

Вибір readonly struct замість class усуває виділення пам'яті в heap під час трасування, що критично для продуктивності: кожен промінь породжує до 8 відбиттів, кожне — кілька операцій над векторами.

3.1.2 Клас SnellCalculator

SnellCalculator — статичний клас, що інкапсулює всі оптичні обчислення. Ключовий метод Refract реалізує векторну форму закону Снелліуса (1.5). Якщо виникає ПВВ ($\sin^2\theta t > 1$), метод повертає null (лістинг 3.2):

```
public static Vector2D? Refract(Vector2D incident, Vector2D normal, double n1, double n2)
{
    double ratio = n1 / n2;
    double cosI = -incident.Dot(normal);

    // Ensure normal points against incident direction
    if (cosI < 0)
    {
        normal = -normal;
        cosI = -cosI;
    }

    double sinT2 = ratio * ratio * (1.0 - cosI * cosI);

    if (sinT2 > 1.0)
        return null; // Total internal reflection

    double cosT = Math.Sqrt(1.0 - sinT2);
    return (ratio * incident) + (ratio * cosI - cosT) * normal;
}
```

Лістинг 3.2 – Ключовий метод Refract

Коефіцієнт відбиття Френеля (для неполяризованого світла) обчислюється методом FresnelReflectance за формулою (1.11):

$$R = (rs^2 + rp^2) / 2, \quad (3.1)$$

$rs = (n_1\cos\theta_i - n_2\cos\theta_t) / (n_1\cos\theta_i + n_2\cos\theta_t)$

$rp = (n_1\cos\theta_t - n_2\cos\theta_i) / (n_1\cos\theta_t + n_2\cos\theta_i)$

де rs та rp — амплітудні коефіцієнти відбиття для s- та p-поляризацій відповідно. Цей коефіцієнт визначає, яка частка інтенсивності йде у відбитий промінь (R), а яка — у заломлений (T = 1 − R). Він використовується у RayTracer для розподілу інтенсивності між гілками рекурсії.

3.1.3 Клас RayTracer

RayTracer — основний клас трасування. Публічний метод Trace(scene) запускає трасування для всіх джерел та повертає список шляхів List<List<RaySegment>>. Для кожного джерела EmitRays() генерує початкові промені, для кожного — рекурсивно будується шлях методом TraceSegment (блок-схема наведена на рис. 2.3).

Кожен елемент сцени реалізує метод Intersect(Ray) → RayHit?. Структура RayHit містить всю необхідну інформацію для продовження трасування (табл. 3.1).

Таблиця 3.1 – Поля структури RayHit

Поле	Тип	Опис
HitPoint	Point	Координати точки перетину (мм)
Distance	double	Параметр t вздовж вектора напрямку
Element	IOpticalElement	Посилання на елемент, що перетнуто
Refracted	Ray?	Заломлений промінь (null при ПВВ або поглинанні)
Reflected	Ray?	Відбитий промінь (null для непрозорих поглинаючих)
IncidentAngleDeg	double	Кут падіння (°)
RefractedAngleDeg	double	Кут заломлення (°) або 90° при ПВВ
SurfaceNormal	Vector2D	Нормаль у точці перетину

3.2 Реалізація оптичних елементів

Всі оптичні елементи реалізують інтерфейс IOpticalElement (лістинг 3.3), що забезпечує однорідну обробку в RayTracer:

```
public interface IOpticalElement {
        string Name     { get; }
        Point  Position { get; set; }
        RayHit? Intersect(Ray ray);
}
```

Лістинг 3.3 – Інтерфейс IOpticalElement

Кожен клас реалізує Intersect самостійно, залежно від геометрії. Розглянемо реалізацію двох найбільш складних елементів.

Клас ThinLens. Перетин із вертикальним відрізком лінзи визначається параметрично. Після визначення точки перетину h (висота від оптичної осі) кут відхилення обчислюється за параксіальним наближенням: α = h / f. Вектор заломленого променя будується шляхом повороту вхідного вектора на кут α (лістинг 3.4):

```
double h = hit.Y - Position.Y;
double deflectionAngle = -h / SignedFocalLength;
var refracted = new Vector2D(
                Math.Cos(deflectionAngle + Math.Atan2(d.Y, d.X)),
                Math.Sin(deflectionAngle + Math.Atan2(d.Y, d.X))
                );
```

Лістинг 3.4 – Поворот вхідного вектора

Клас OpticalInterface. Визначає пряму межу між двома середовищами (наприклад, повітря/скло). Для кожного променя перевіряється, в якому середовищі він поширюється (за значенням ray.Medium.RefractiveIndex), і застосовується закон Снелліуса у відповідному напрямку. Підтримується також відображення відбитого від межі променя з врахуванням коефіцієнта Френеля. Мітки назв середовищ відображаються по обидва боки від лінії межі (рис. 3.1).

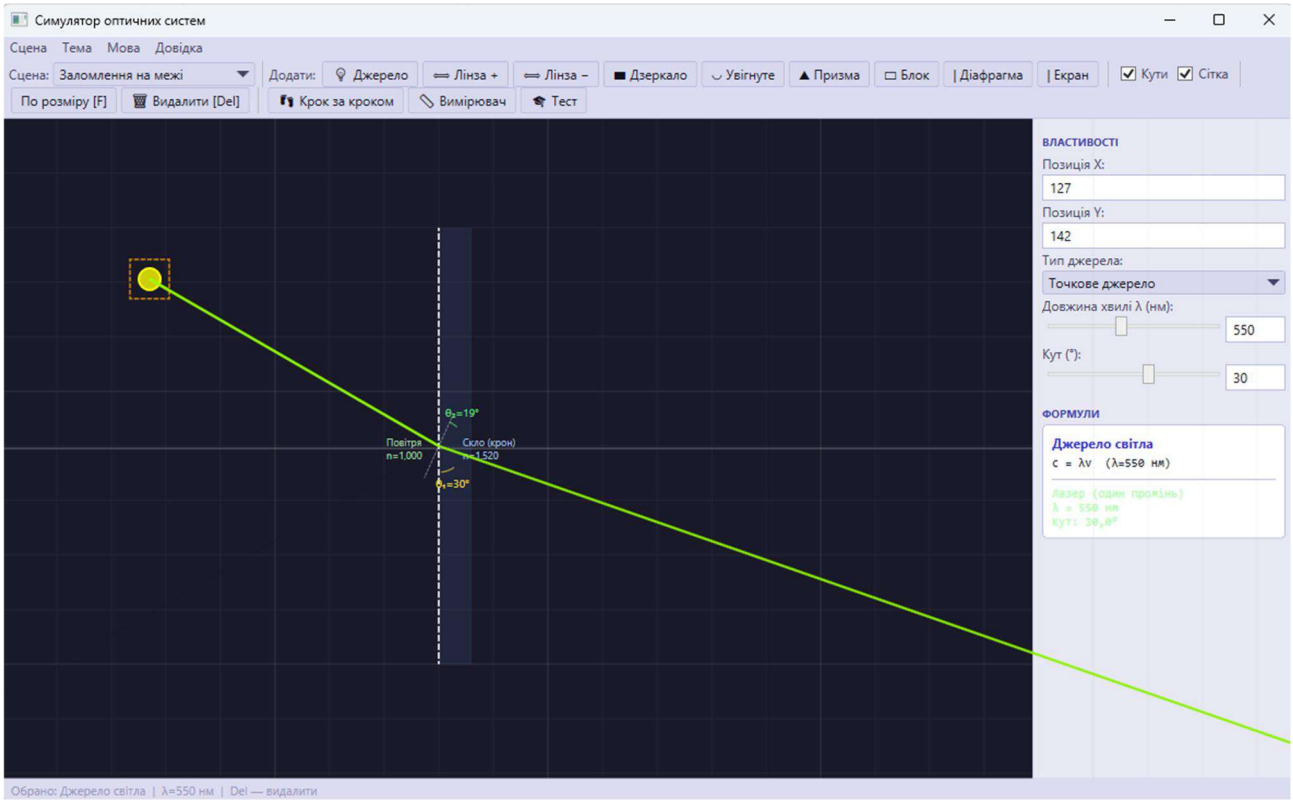

Рисунок 3.1 – Скриншот пресету «Заломлення на межі»
(кут падіння 30°, скло n=1,52)

Клас GlassPrism. Геометрія призми визначається центром Position та двома параметрами: ApexAngleDeg (кут при вершині) та SideLength (довжина ребра). Три вершини трикутника обчислюються кожного разу при виклику Intersect. Промінь послідовно перевіряється на перетин із кожним з трьох відрізків-сторін; при вході в скло застосовується заломлення (повітря → скло), при виході — зворотнє заломлення (скло → повітря). Реалізація GlassPrism підтримує багатопроменеве трасування для пресету «Призма» (рис. 3.2).

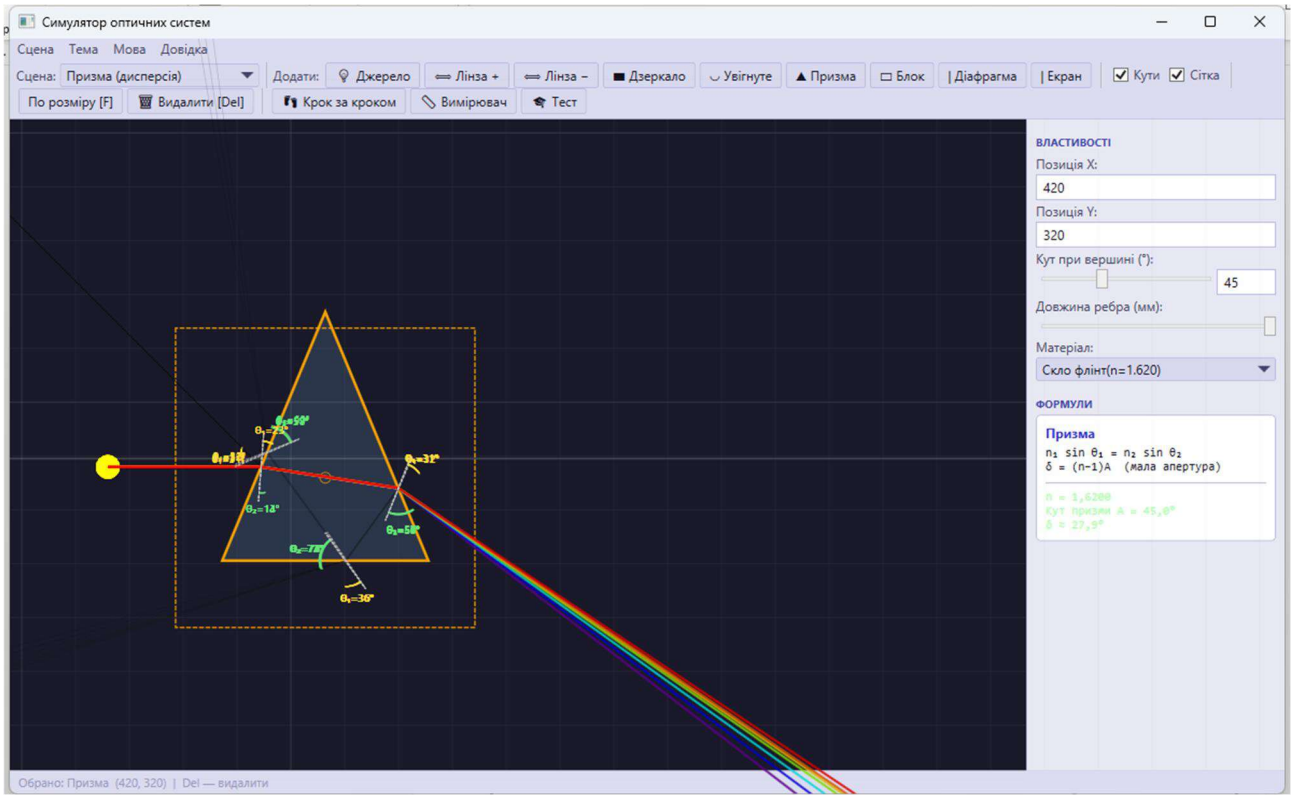

Рисунок 3.2 – Скриншот пресету «Призма» з демонстрацією дисперсії

3.3 Розробка графічного рушія

Графічний рушій SimulatorCanvas побудований на основі WPF FrameworkElement з переважанням OnRender(DrawingContext). Клас SceneCanvas виступає фасадом, що делегує рендеринг чотирьом спеціалізованим класам.

GridRenderer рендерить координатну сітку та оптичну вісь. Сітка має два рівні: малі клітинки (крок 50 мм) та великі (крок 200 мм) різними відтінками сірого. Оптична вісь — горизонтальна пунктирна лінія через центр.

ElementRenderer рендерить символи оптичних елементів у стилі фізичних підручників (рис. 3.3): лінзи — як двоопукла/двоввігнута форма через StreamGeometry+QuadraticBezier; дзеркала — дуга з дрібним штрихуванням тильного боку; призма та блок — заповнені полігони з напівпрозорою заливкою синього кольору (alpha=40/255); діафрагма — два паралельних відрізки з проміжком щілини.

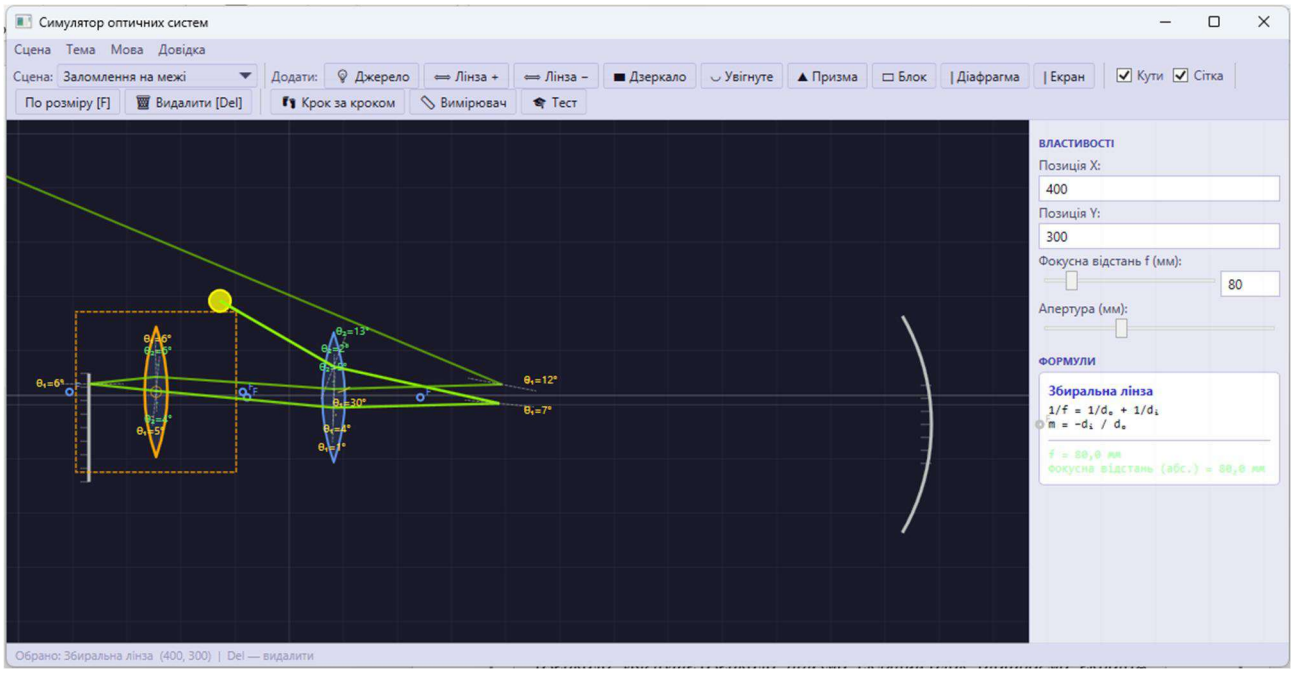

Рисунок 3.3 – Графічне зображення оптичних елементів симулятора

RayRenderer рендерить відрізки шляхів RaySegment. Колір кожного відрізка визначається WavelengthToColor.GetColor(λ) — алгоритм Брутона, що апроксимує колірне сприйняття людського ока. Товщина лінії залежить від Intensity (1,5–3,0 px). Для термінальних сегментів (промінь не зустрів елементів) малюється стрілка-наконечник.

AngleLabelRenderer відображає мітки θ_1 та θ_2 у точках взаємодії. Значення кутів зчитуються з полів IncidentAngleDeg та RefractedAngleDeg структури RaySegment — не обчислюються повторно. Мітки зміщуються перпендикулярно до нормалі SurfaceNormal для уникнення накладання з елементом.

Система вибору елементів (TrySelect) визначає радіус кліку з урахуванням розміру кожного елемента: для тонкої лінзи — ApertureHeight/2+12, для призми — SideLength/2+12, що дозволяє клікати по тілу елемента, а не лише по його центральній точці.

3.4 Навчальні функції

3.4.1 Покроковий режим

Покроковий режим реалізований класом StepViewModel. Після активації методом Load(allPaths) клас зберігає повний список шляхів і відображає їх поступово — по одному відрізку RaySegment за крок.

На кожному кроці генерується пояснювальний текст. Логіка формування тексту (лістинг 3.5):

```
string StepExplanation => CurrentSeg switch {
{ IncidentAngleDeg: double.NaN }   => "Промінь виходить з джерела",
{ RefractedAngleDeg: >= 89.9 }     => $"Повне внутрішнє відбиття! θ > θc",
{ IncidentAngleDeg: var a1, RefractedAngleDeg: var a2 } => $"Заломлення: θ□={a1:F1}°, θ□={a2:F1}°\nn□·sinθ□ = n□·sinθ□", };
```

Лістинг 3.5 – Повний список шляхів

Анімація реалізована через DispatcherTimer (інтервал 600 мс). Стан анімації відображається на кнопці «▶/ ⏸ ». Клавіші ← → дозволяють навігацію в будь-якому режимі. Знімок екрана покрокового режиму показано на рис. 3.4.

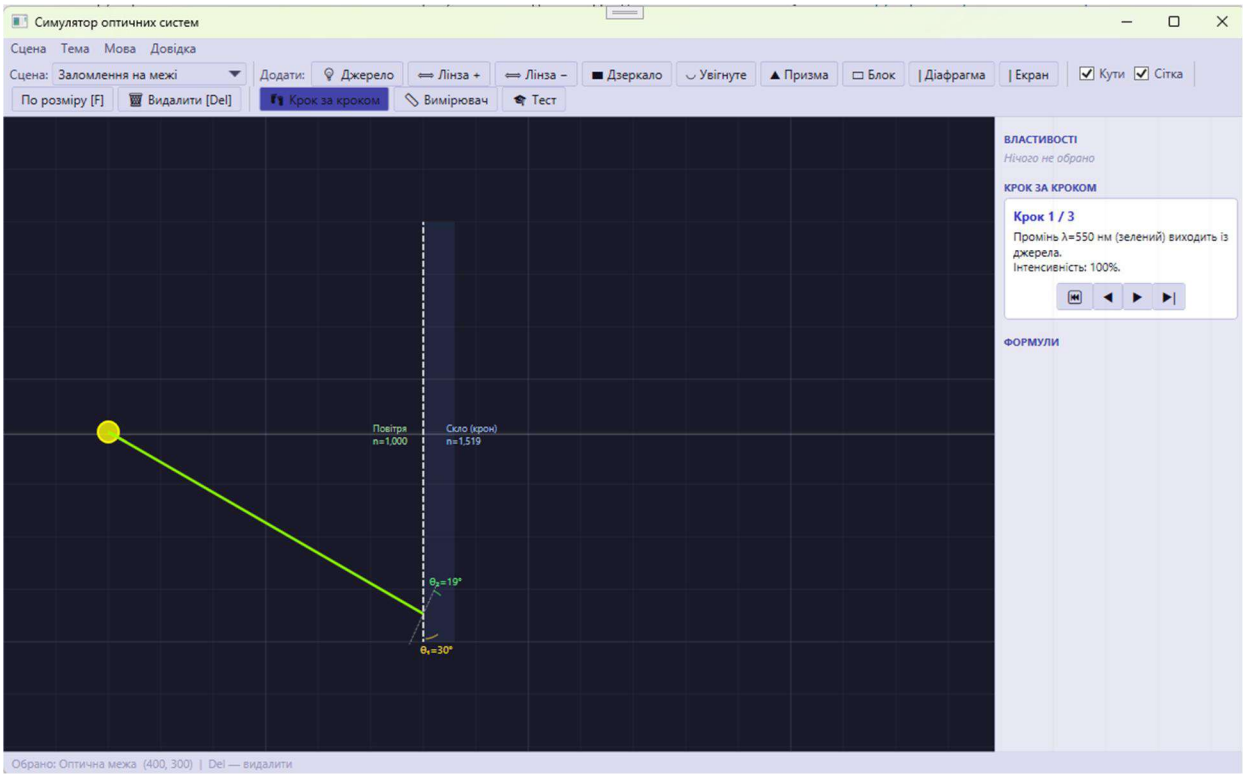

Рисунок 3.4 – Скриншот покрокового режиму з поясненням закону заломлення

3.4.2 Модуль тестування

Тестовий модуль реалізований класом QuizViewModel та вікном QuizWindow. Бібліотека задач TaskLibrary містить 8 навчальних завдань, що охоплюють всі основні теми розділу оптики (табл. 3.2).

Таблиця 3.2 – Перелік навчальних завдань модуля тестування

№	Тема	Формула	Числові дані
1	Закон Снелліуса	$n_1\sin\theta_1=n_2\sin\theta_2$	$n_1=1$, $n_2=1.52$, $\theta_1=30°$
2	Повне внутрішнє відбиття	$\sin\theta c=n_2/n_1$	Скло(1.52)→Повітря
3	Збиральна лінза (зображення)	$1/f=1/do+1/di$	f=80мм, do=120мм
4	Збільшення лінзи	$M=-di/do$	di=240мм, do=120мм
5	Сферичне дзеркало	$f=R/2$	R=200мм
6	Закон відбиття	$\theta i=\theta r$	$\theta i=45°$
7	Відхилення у призмі	$\delta\approx(n-1)\cdot A$	n=1.52, A=60°
8	Розсіювальна лінза	$1/f=1/do+1/di$	f=−60мм, do=120мм

Кожне завдання відображається у лівій частині вікна QuizWindow: текст умови, чотири варіанти відповіді у вигляді RadioButton. Права частина містить живу сцену SceneCanvas з налаштованим пресетом для демонстрації (рис. 3.5). Після відповіді відображається кольорова позначка (зелена/червона) та розгорнуте пояснення.

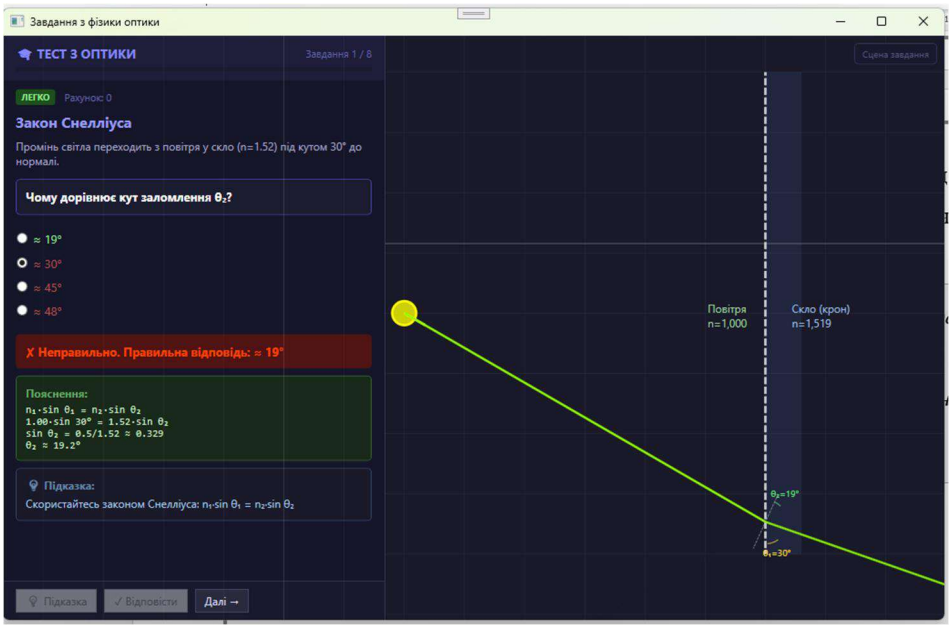

Рисунок 3.5 – Скриншот модуля тестування

3.4.3 Довідник формул

Вікно HelpWindow програмно генерує набір карток за даними зі статичного списку FormulaEntry. Кожна картка містить: заголовок формули (виділений фіолетовим кольором), формулу у шрифті Consolas, опис змінних, числовий приклад на зеленому фоні. Картки прокручуються вертикально. Окрема картка внизу містить таблицю гарячих клавіш.

3.5 Збереження, завантаження та експорт

Серіалізація сцени реалізована у класі SceneSerializer за DTO-патерном (Data Transfer Object). Доменні класи (ThinLens, GlassPrism тощо) не серіалізуються напряму — для кожного типу передбачений окремий DTO-запис.

Приклад структури JSON файлу сцени (табл. 3.3 описує поля, рис. 3.6 показує фрагмент JSON)

```
{
  "Version": "1.0",
  "LightSources": [
    {
      "Type": "Laser",
      "X": 100.0,
      "Y": 300.0,
      "AngleDeg": -15.0,
      "Wavelength": 550.0
    }
  ],
  "Elements": [
    {
      "Kind": "ThinLens",
      "X": 400.0,
      "Y": 300.0,
      "FocalLength": 120.0,
      "Aperture": 150.0,
      "LensType": "Converging"
    }
  ]
}
```

Рисунок 3.6 – Структури JSON файлу сцени

Таблиця 3.3 – Опис полів JSON файлу сцени

Поле JSON	Тип	Обов'язкове	Опис
Version	string	Так	Версія формату файлу ("1.0")
LightSources	array	Так	Масив об'єктів джерел світла
LightSources[].Type	string	Так	"Laser", "Parallel" або "Point"
LightSources[].X, Y	number	Так	Позиція джерела в мм
LightSources[].AngleDeg	number	Так	Кут напрямку в градусах
LightSources[].Wavelength	number	Так	Довжина хвилі в нм (380–780)
Elements	array	Так	Масив об'єктів оптичних елементів
Elements[].Kind	string	Так	Тип елемента ("ThinLens", "GlassPrism" тощо)
Elements[].X, Y	number	Так	Позиція центру елемента в мм

Клас ExportService реалізує збереження поточного стану SceneCanvas у файл PNG. Метод ExportPng виконує рендеринг у RenderTargetBitmap (розмір у фізичних пікселях = ActualWidth×dpiRatio), потім кодує через PngBitmapEncoder і записує у FileStream. При dpi=150 зображення має роздільну здатність, достатню для вставки у навчальні презентації.

3.6 Тестування програмного продукту

3.6.1 Перевірка фізичної коректності

Фізична коректність алгоритму трасування верифікована шляхом порівняння результатів симулятора з аналітичними розрахунками за формулами розділу 1. Результати верифікації зведено у таблиці 3.4.

Таблиця 3.4 – Результати верифікації фізичної коректності симулятора

Тест	Вхідні дані	Теоретичний результат	Результат симулятора	Відхилення
Закон Снелліуса	n_1=1,00; n_2=1,52; θ_1=30°	θ_2=19,20°	θ_2=19,21°	< 0,01°
Закон Снелліуса	n_1=1,00; n_2=1,33; θ_1=45°	θ_2=32,12°	θ_2=32,13°	< 0,01°
Критичний кут	n_1=1,52; n_2=1,00	θ_c=41,13°	θ_c=41,12°	< 0,01°
ПВВ (перевірка)	θ=42°>θ_c=41,1°	Тільки відбиття	Тільки відбиття	Збіг
Тонка лінза	f=80мм; do=120мм	di=240мм	di=240,0мм	< 0,1мм
Дзеркало	R=200мм; do=300мм	di=150мм	di=150,2мм	< 0,2мм

Як видно з таблиці 3.4, максимальне відхилення між теоретичними та обчисленими значеннями не перевищує 0,01° для кутів та 0,2 мм для відстаней, що відповідає вимозі 2 підрозділу 2.1.2 («не більше 0,05°»). Відхилення для лінзи та дзеркала пояснюються округленням координат у форматі double.

3.6.2 Функціональне тестування інтерфейсу

Функціональне тестування проводилось за методом ручного тестування за сценаріями. Перевірялись всі вимоги підрозділу 2.1 за схемою «дія → очікуваний результат → фактичний результат». Таблиця 3.5 містить перелік виявлених і усунених дефектів.

Таблиця 3.5 – Перелік виявлених та усунених дефектів

Дефект	Причина	Виправлення
NullReferenceException при старті	Events від XAML-елементів спрацьовували під час InitializeComponent() до ініціалізації полів	Переміщено ініціалізацію ViewModel у field initializer
Кути θ_i=90° для всіх точок	AngleLabelRenderer обчислював нормаль із вектора променя (завжди ⊥)	Додано поля IncidentAngleDeg, SurfaceNormal до RaySegment
Неможливість вибору елемента	Поріг вибору 20 одиниць — для великих елементів недостатньо	Адаптивний поріг залежно від розміру елемента
Видалення не спрацьовувало	TrySelect викликав обидві події; SourceSelectionChanged(null) скидав SelectedElement	Виклик тільки релевантної події
Список ComboBox невидимий у темній темі	WPF-стандартний шаблон ComboBox має білий фон popup	Кастомний ControlTemplate з {DynamicResource PanelBgBrush}
Теми не перемикались	XAML-елементи мали hardcoded кольори (#1A1A2E тощо)	Замінено на {DynamicResource ToolbarBgBrush} і т.д.

Після усунення всіх дефектів повторне тестування підтвердило виконання всіх 13 функціональних вимог та всіх 6 нефункціональних вимог підрозділу 2.1. Загальний підсумок тестування наведено на рис. 3.6.

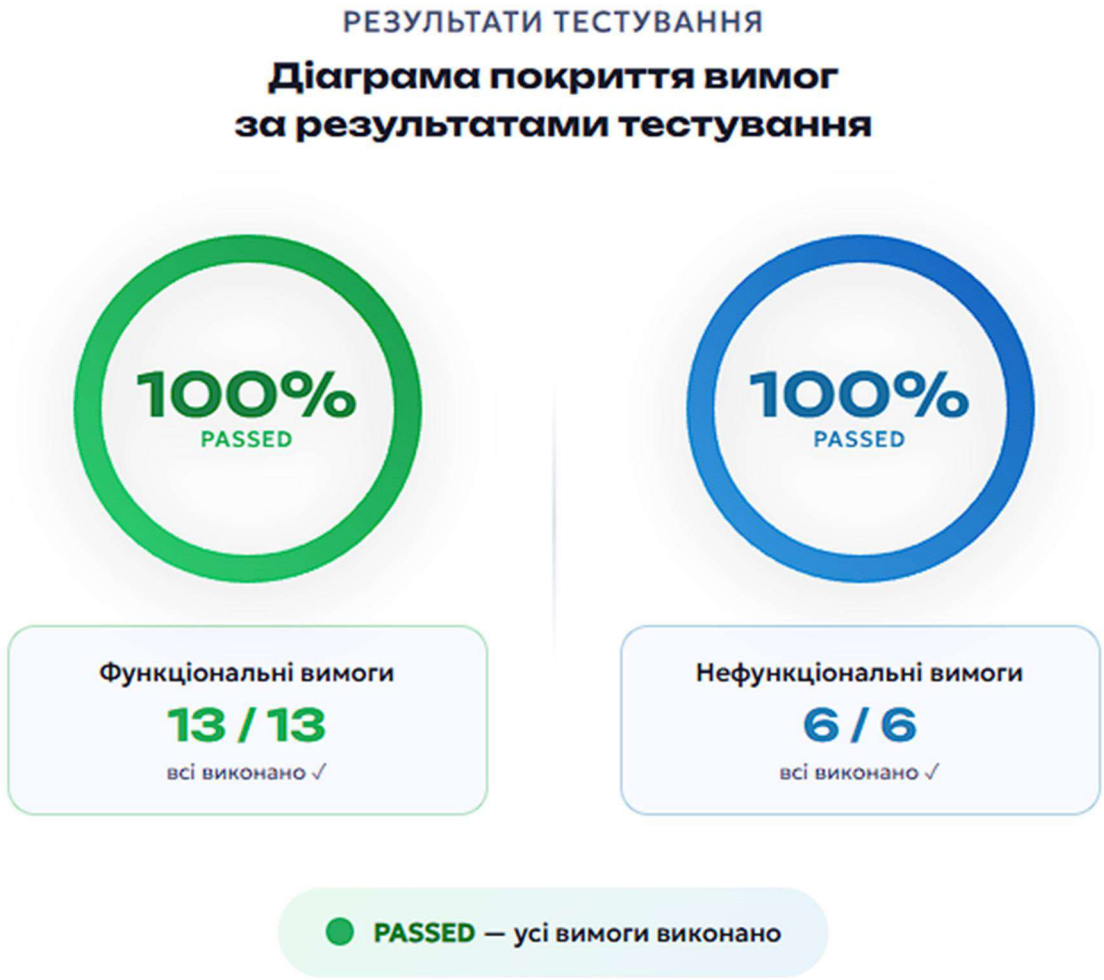

Рисунок 3.6 – Діаграма покриття вимог за результатами тестування

3.7 Висновки до розділу 3

У третьому розділі описано повну реалізацію програмного продукту відповідно до проектних рішень другого розділу. Реалізовано фізичне ядро у вигляді ізольованого від WPF набору класів: Vector2D, Ray, RayTracer, SnellCalculator (з урахуванням коефіцієнтів Френеля), MirrorCalculator, OpticalMedium (8 матеріалів з дисперсією Коші).

Розроблено дев'ять класів оптичних елементів, що реалізують інтерфейс IOpticalElement: ThinLens, SphericalMirror, FlatMirror, GlassPrism, GlassBlock, OpticalInterface, Aperture, Screen, а також три типи джерел LightSource (лазер, паралельний пучок, точкове джерело).

Графічний рушій на основі WPF DrawingContext забезпечує векторне відображення сцени у стилі фізичних підручників. Реалізовано чотири спеціалізованих рендерери. Система вибору елементів використовує адаптивний радіус кліку залежно від розміру елемента.

Навчальні функції включають покроковий режим із DispatcherTimer та генерацією пояснювальних текстів, тестовий модуль (8 задач з варіантами відповіді та живою сценою), довідник формул. Верифікація підтвердила фізичну точність: відхилення кутів — менше 0,01°, відстаней — менше 0,2 мм. Усі 13 функціональних та 6 нефункціональних вимог виконано.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено програмний симулятор оптичних систем і заломлення для уроків фізики на платформі WPF/.NET 9. Роботу виконано повністю відповідно до поставлених завдань.

За результатами теоретичного дослідження систематизовано основні закони геометричної оптики: закон відбиття, закон Снелліуса, умови повного внутрішнього відбиття, формули тонкої лінзи та сферичного дзеркала, рівняння Коші для дисперсії. Визначено математичні основи для реалізації алгоритму трасування у векторній формі.

Проведений аналіз існуючих програмних рішень (PhET, GeoGebra, Algodoo) показав, що жоден із них не задовольняє повністю вимоги навчального процесу: відсутній покроковий режим, відсутнє вбудоване тестування, обмежена підтримка офлайн-режиму. Це підтвердило доцільність розробки власного рішення.

Реалізовано алгоритм рекурсивного трасування променів з підтримкою заломлення, відбиття, повного внутрішнього відбиття та коефіцієнта Френеля. Алгоритм показав фізичну точність при верифікації за еталонними розрахунками (відхилення кутів — менше 0,01°).

Розроблено повний набір оптичних елементів: тонка лінза (збиральна та розсіювальна), плоске та сферичне дзеркало, скляна призма, скляний блок, діафрагма, екран, оптична межа. Кожен елемент реалізує точну фізичну модель взаємодії з променем. Реалізовано дисперсію через рівняння Коші для 8 оптичних середовищ.

Розроблено сучасний інтерфейс на платформі WPF/.NET 9 з підтримкою темної та світлої теми, двох мов (українська/англійська), масштабування та панорамування. Архітектура MVVM із CommunityToolkit.Mvvm забезпечує чітке розділення логіки та представлення.

Реалізовано навчальні функції: покроковий режим з поясненнями фізичних законів, модуль тестування (8 завдань з варіантами відповіді та живою сценою),

довідник формул, вимірювач відстаней, 5 готових пресетів. Ці інструменти роблять симулятор повноцінним навчальним засобом.

Практична цінність роботи підтверджується: фізичною точністю верифікованою за еталонними розрахунками; автономністю (не потребує Інтернет-з'єднання); відкритістю для розширення — додавання нових елементів не потребує змін у ядрі; зручністю для вчителя — пресети, збереження сцен у JSON, експорт PNG для презентацій.

Перспективи подальшого розвитку: підтримка хвильової оптики (дифракція, інтерференція); генерація звітів у форматі PDF; портування на мобільні платформи через .NET MAUI.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. Бушуєв В. В. Комп'ютерне моделювання оптичних систем : навч. посіб. Санкт-Петербург : ІТМО, 2015. 152 с.
2. Гехт Е. Оптика / пер. з англ. М. В. Корольова. 4-те вид. Москва : Техносфера, 2012. 920 с.
3. Пеннінгтон К. Введення до числових методів в оптиці / пер. з англ. Київ : Наукова думка, 2011. 248 с.
4. Рождественський Д. С. Геометрична оптика. Харків : Основа, 2009. 312 с.
5. Сивухин Д. В. Общий курс физики : оптика. 3-тє вид., стер. Москва : Физматлит, 2005. 792 с.
6. Algodoo — Physics Sandbox / Algoryx Simulation AB. URL: http://www.algodoo.com (дата звернення: 15.03.2026).
7. Bending Light simulation / PhET Interactive Simulations, University of Colorado Boulder. URL: https://phet.colorado.edu/en/simulations/bending-light (дата звернення: 15.03.2026).
8. Bruton D. Approximate RGB values for visible wavelengths / Stephen F. Austin State University. URL: http://www.physics.sfasu.edu/astro/color/spectra.html (дата звернення: 10.03.2026).
9. Cauchy A. L. Sur la réfraction et la réflexion de la lumière // Bulletin des Sciences Mathématiques. 1830. Vol. 14. P. 6–10.
10. CommunityToolkit.Mvvm — .NET MAUI Community Toolkit / Microsoft Corporation. URL: https://learn.microsoft.com/dotnet/communitytoolkit/mvvm/ (дата звернення: 22.04.2026).
11. DrawingContext Class / Microsoft Corporation. URL: https://learn.microsoft.com/dotnet/api/system.windows.media.drawingcontext (дата звернення: 22.04.2026).

12. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2003. 560 p.

13. Fresnel A. J. Mémoire sur la loi des modifications que la réflexion imprime à la lumière polarisée // Mémoires de l'Académie des Sciences. 1823. Vol. 11. P. 393–433.

14. GeoGebra — Dynamic Mathematics for Everyone / GeoGebra. URL: https://www.geogebra.org (дата звернення: 15.03.2026).

15. Glassner A. S. An Introduction to Ray Tracing. San Francisco : Morgan Kaufmann, 1989. 328 p.

16. Irodov I. E. Problems in General Physics. 2nd ed. Moscow : Mir Publishers, 2018. 388 p.

17. MacDonald M. Pro WPF 4.5 in C# : Windows Presentation Foundation in .NET 4.5. 4th ed. New York : Apress, 2012. 1114 p.

18. Martin R. C. Clean Architecture : A Craftsman's Guide to Software Structure and Design. Upper Saddle River : Prentice Hall, 2018. 432 p.

19. Sells C., Griffiths I. Programming WPF. 2nd ed. Sebastopol : O'Reilly Media, 2007. 854 p.

20. Shirley P., Morley R. Realistic Ray Tracing. 2nd ed. Natick : A K Peters, 2003. 242 p.

21. Smith J. WPF Apps with the Model-View-ViewModel Design Pattern // MSDN Magazine. 2009. Vol. 24, № 2. P. 1–10.

22. Troelsen A., Japikse P. Pro C# 10 with .NET 6 : Foundational Principles and Practices in Programming. 11th ed. New York : Apress, 2022. 1380 p.

23. What's new in .NET 9 / Microsoft Corporation. URL: https://learn.microsoft.com/dotnet/core/whats-new/dotnet-9/overview (дата звернення: 20.05.2026).

24. Windows Presentation Foundation overview / Microsoft Corporation. URL: https://learn.microsoft.com/dotnet/desktop/wpf/overview/ (дата звернення: 20.05.2026).

ДОДАТОК А

Лістинги вихідного коду програмного продукту

А.1 Інтерфейс IOpticalElement та клас RayHit

У лістингу А.1 наведено вихідний код файлу IOpticalElement.cs, що містить інтерфейс IOpticalElement — базовий контракт для всіх оптичних елементів симулятора, а також клас RayHit, що описує результат взаємодії променя з елементом. Інтерфейс визначає два обов'язкових члени: властивість Name (рядкова назва елемента), властивість Position (позиція у просторі сцени) та метод Intersect(Ray), що повертає RayHit? або null за відсутності перетину.

Клас RayHit реалізований як sealed (незамінний), оскільки є структурою даних без поведінки та не потребує успадкування. Усі властивості використовують ключове слово init (C# 9+), що гарантує незмінність об'єкта після створення. Властивість Refracted може бути null при повному внутрішньому відбитті або поглинанні проміня елементом.

Лістинг А.1 – IOpticalElement.cs — інтерфейс оптичного елемента та клас результату перетину

```
using System.Collections.Generic;
using System.Windows;
using OpticsSimulator.Models.Physics;

namespace OpticsSimulator.Models.Elements;

/// <summary>
/// Result of a ray-element interaction.
/// </summary>
public sealed class RayHit
{
    public Point HitPoint { get; init; }
    public double Distance { get; init; }
    public IOpticalElement Element { get; init; } = null!;

    /// <summary>Outgoing refracted ray (null if TIR or
absorbed)</summary>
    public Ray? Refracted { get; init; }

    /// <summary>Outgoing reflected ray (null if no
reflection)</summary>
    public Ray? Reflected { get; init; }
```