Державний вищий навчальний заклад
Прикарпатський національний університет імені Василя Стефаника

Кафедра алгебри та геометрії

Дипломна робота

на здобуття першого (бакалаврського) рівня вищої освіти
на тему: «Методи інтерполяції функцій і їх використання у програмуванні»

Виконав: студент IV курсу, групи М-41
спеціальності 111 Математика

Прокопів Денис Васильович

Керівник : к.ф.- м.н. Копорх К.М

Рецензент : к.ф.- м.н. Глушак І.Д.

м. Івано-Франківськ – 2025 рік

ЗМІСТ

ВСТУП

Інтерполяція функцій - один із основних підходів у сучасній математиці, який дозволяє знаходити значення функції в проміжних точках, спираючись на обмежений набір відомих даних. Цей метод активно застосовується в чисельному аналізі, моделюванні процесів та обробці інформації, забезпечуючи ефективні рішення для задач, що виникають у науці, техніці та інженерії.

Об'єктом цього дослідження виступають математичні методи інтерполяції, їх теоретичне обґрунтування і можливості практичного використання.
Предметом аналізу є особливості ключових підходів до інтерполяції, їхні властивості та переваги у вирішенні прикладних задач.

Мета роботи полягає у систематизації теоретичних знань про інтерполяцію функцій, дослідженні методів її реалізації та оцінці ефективності для різних практичних завдань.

Швидкий розвиток інформаційних технологій створює нові виклики та можливості для реалізації методів інтерполяції. В рамках дослідження буде приділено увагу сучасним підходам, які спираються на класичні математичні основи, але адаптовані до потреб цифрової епохи.

Методи дослідження включають аналіз наукових джерел, порівняння математичних моделей, а також розгляд прикладів їх реалізації в реальних задачах.

Практичне значення цієї роботи полягає у можливості використання отриманих висновків для розробки ефективних алгоритмів моделювання та аналізу даних, що є особливо важливим для сучасної техніки і науки.

Структура роботи складається зі вступу, двох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг становить … сторінок, включаючи … рисунків і … таблиць.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВІ ВИКОРИСТАННЯ ІНТЕРПОЛЯЦІЇ ФУНКЦІЙ.

Історія виникнення та застосування методів інтерполяції

Сер Едмунд Віттакер, професор чисельної математики в Единбурзькому університеті з 1913 по 1923 рік, зауважив, що «найпоширеніша форма інтерполяції виникає, коли ми шукаємо дані в таблиці, яка не містить точних значень, які нам потрібні».

Протягом всієї історії інтерполяція в тій чи іншій формі використовувалася практично для всіх цілей під сонцем. Говорячи про сонце, одні з перших збережених свідчень використання інтерполяції походять із стародавнього Вавилону та Греції. Близько 300 року до н.е. вони використовували не тільки лінійну, але й більш складні форми інтерполяції для прогнозування положень сонця, місяця і планет, про які вони знали. Фермери, які розраховували час посіву своїх культур, були основними користувачами цих прогнозів. Також у Греції близько 150 року до н.е. Гіппарх Родоський використовував лінійну інтерполяцію для побудови «хордової функції», схожої на синусоїду, для обчислення положень небесних тіл. Далі на схід, китайські свідчення про інтерполяцію датуються приблизно 600 роком нашої ери. Лю Чжуо використовував еквівалент інтерполяції другого порядку Грегорі-Ньютона для побудови «Імператорського стандартного календаря».

У 625 році нашої ери індійський астроном і математик Брахмагупта запровадив метод інтерполяції синусоїди другого порядку, а пізніше - метод інтерполяції даних з нерівними інтервалами. Протягом століть було знайдено багато подібних наземних цілей для інтерполяції, але однією з найважливіших сфер застосування протягом століть була океанська навігація. Таблиці значень спеціальних функцій були побудовані за допомогою чисельних методів, і моряки використовували певні з них для визначення значень широти і довготи. Французький уряд розпочав виробництво великого набору таких таблиць, коли було запроваджено метричну систему. В ідеалі, хотілося б, щоб математики створили великий набір таблиць завдяки своїй обізнаності в цьому питанні.

Однак основним джерелом роботи над проектом стали перукарі, які втратили своїх клієнтів з кричущими перуками на гільйотині. Сумна правда про таблиці спеціальних функцій полягає в тому, що більшість з них були плагіатом. З появою «комп'ютерів» працівники, які проводили і записували розрахунки, були схильні до плагіату , обчислення, були схильні робити багато помилок під час створення цих складних таблиць, плагіат лише поширював більше помилок. Чарльз Беббідж намагався вирішити цю проблему, винайшовши свій «різницевий двигун» - механічний комп'ютер, запрограмований за допомогою перфокарт. Паралельно Беббідж також намагався винайти систему, яка обирала б виграшні номери на кінних перегонах, сподіваючись заробити додаткові гроші. Хоча йому не бракувало коштів, його життя було коротким, і він так і не побачив завершення свого винаходу. Понад століття з чвертю потому, коли ми занурюємося в еру нанотехнологій, Беббідж вважається дідусем сучасних комп'ютерів.

Під час Великої депресії у Сполучених Штатах відбувся останній сплеск ручного виготовлення таблиць. Незадовго до Другої світової війни Адміністрація розвитку промисловості розпочала проект «Математичні таблиці». Як і у випадку з французьким проектом, бажані «математики» виявилися цього разу некваліфікованими настільки, що від'ємні числа викликали у них здивування. Рішення: чорні олівці для додатних чисел і червоні для від'ємних. Завдяки тому, що кожен розрахунок у цьому проекті повторювався двічі (кожен раз іншою людиною), а також завдяки ретельному вичитуванню, ці таблиці були «можливо, найточнішими з усіх, що коли-небудь створювалися». Багато з них були зібрані в книзі Мілтона Абромовіца та Ірен Стегун, яка досі використовується у всьому світі. Завдяки комп'ютерам (а не людям) таблиці більше не складаються вручну, але уряд Австралії випускає таблиці життя, які описують рівень смертності. Що стосується індустрії страхування життя і вивчення демографії, «ці таблиці розширюються за допомогою сучасних методів інтерполяції». Незалежно від того, наскільки досконалими чи обширними вони є, інтерполяція завжди буде потрібна для знаходження значень у сучасних таблицях через їхню природу. Оскільки вони не є безперервними функціями,

буде нескінченно багато пропущених значень. Два методи інтерполяції, які викладаються в HNMI, належать Ньютону і Лагранжу. Ньютон розпочав свою роботу над цим питанням у 1675 році, яка «заклала фундамент класичної теорії інтерполяції». У 1795 році Лагранж опублікував формулу інтерполяції, яка тепер відома під його ім'ям, незважаючи на те, що Ворінг вже вивів цю формулу шістнадцять років тому.

Інтерполяція - це фундаментальний інструмент чисельного аналізу, який застосовується в широкому спектрі галузей для оцінки невідомих значень між відомими точками даних. У цьому розділі розглядається практичне застосування методів інтерполяції в науці, інженерії та інших галузях. В інженерії інтерполяція відіграє вирішальну роль в аналізі даних і вирішенні реальних проблем. Наприклад: Термодинаміка: інженери використовують інтерполяцію для оцінки термодинамічних властивостей, таких як тиск, температура та ентропія, коли точні значення недоступні в стандартних таблицях. Структурний аналіз: інтерполяція використовується для визначення значень напружень і деформацій в матеріалах під впливом різних навантажень, що має вирішальне значення для проектування безпечних конструкцій. Обробка сигналів: Фур'є-інтерполяція допомагає реконструювати сигнали з вибіркових даних, забезпечуючи точне представлення вихідного сигналу.

Поняття про інтерполяцію: сутність , класифікація та термінологія

Інтерполяція - це фундаментальна техніка в математиці та статистиці, яка дозволяє оцінювати невідомі значення в діапазоні, визначеному набором відомих точок даних. Основною метою інтерполяції є побудова безперервної функції або кривої, яка або проходить через ці відомі точки, або наближається до них. Цей метод широко застосовується в різних галузях, включаючи інженерію, комп'ютерну графіку, аналіз даних і наукові дослідження, де важливо передбачити значення в невиміряних місцях на основі наявної інформації. По суті, інтерполяція спирається на взаємозв'язки між відомими точками даних для отримання значень у проміжних положеннях. Наприклад, якщо показники температури знімаються в певний час протягом дня, інтерполяція може бути використана для оцінки температури в будь-який час між цими показниками, забезпечуючи таким чином більш повне розуміння температурних коливань.

Методи інтерполяції можна класифікувати за кількома критеріями:

Поліноміальна інтерполяція: Цей метод передбачає підбір поліноміальної функції до відомих точок даних. Найпоширеніші методи включають інтерполяцію Лагранжа та інтерполяцію розділеними різницями Ньютона. Хоча поліноміальна інтерполяція є відносно простою, вона іноді може призвести до таких проблем, як коливання на краях інтервалу, відомі як феномен Рунге.

Кускова інтерполяція: У цьому підході діапазон даних розбивається на менші сегменти, а для з'єднання точок всередині кожного сегмента використовуються простіші функції (часто лінійні). Лінійна інтерполяція та сплайн-інтерполяція (наприклад, кубічні сплайни) є популярними прикладами, що забезпечують більш плавні переходи між точками даних.

Тригонометрична інтерполяція: Цей метод використовує тригонометричні функції для інтерполяції даних, що робить його особливо

корисним для періодичних функцій. Ряд Фур'є є відомим прикладом цього типу інтерполяції.

Інтерполяція радіально-базисних функцій: Цей метод використовує радіальні базисні функції для інтерполяції даних у декількох вимірах. Він особливо ефективний у просторах високої розмірності і знаходить застосування в таких галузях, як геостатистика і машинне навчання.

Крігінг: Складний метод геостатистичної інтерполяції, який не тільки надає оцінки, але й кількісно оцінює невизначеність, пов'язану з цими оцінками. Зазвичай використовується в аналізі просторових даних.

Ознайомлення з термінологією, пов'язаною з інтерполяцією, має важливе значення для ефективного застосування та комунікації цих методів. Ось деякі важливі терміни:

Точки даних - відомі значення в певних точках або інтервалах, які є основою для інтерполяції.

Функція інтерполяції - математична модель або функція, створена для оцінки значень між відомими точками даних.

Вузол - конкретна точка даних, що використовується в процесі інтерполяції.

Ступінь полінома - відноситься до найвищого степеня змінної в поліноміальній функції інтерполяції.

Похибка - розбіжність між фактичним та інтерпольованим значенням.

Екстраполяція - споріднене поняття, яке передбачає оцінку значень за межами діапазону відомих точок даних.

Згладжування - метод, який часто використовується разом з інтерполяцією для зменшення шуму в даних і створення більш візуально привабливої кривої.

Висновки до розділу І

Інтерполяція слугує важливим інструментом для оцінки невідомих значень на основі відомих даних, покращуючи аналіз та прийняття рішень у різних дисциплінах. Засвоївши її основні концепції, типи і термінологію, фахівці можуть вибрати найбільш підходящий метод інтерполяції для своїх конкретних застосувань, забезпечуючи точні і надійні результати.

РОЗДІЛ 2. МАТЕМАТИЧНІ МЕТОДИ ТА ЗАСТОСУВАННЯ ІНТЕРПОЛЯЦІЇ ФУНКЦІЙ

2.1 Поліноміальна інтерполяція та її математичні особливості

Поліноміальна інтерполяція — це метод апроксимації функцій, який передбачає побудову полінома, що точно проходить через задані точки. Якщо задано набір точок , то можна визначити поліном , який задовольняє умову для кожного .

Ключові характеристики методу :

1. **Ступінь побудованого полінома**: поліном, який інтерполює точку, має максимальний ступінь . Це дозволяє йому точно відтворювати значення у вузлах.
2. **Оцінка похибки**: Для функції , яка має безперервну похідну порядку , похибка в точці обчислюється за формулою:

$$R_n = \frac{f^{(n+1)}(\varepsilon)}{(n+1)!}\prod_{i=0}^{n}(x - x_i)$$

Де $R_n -$ похибка інтерполяції в точці x , $f^{(n+1)}(\varepsilon) -$ значення $(n+1) -$ ї похідної функції $f(x)$ у деякій точці $\varepsilon \in [\min(x_i), \max(x_i)]$, $x_i -$ вузли інтерполяції , $n -$ кількість вузлів мінус 1(ступінь полінома)

Таким чином, точність залежить як від розташування вузлів , так і від значень похідної .

3. **Ефект розташування вузлів**: Якщо вузли розташовані рівномірно на великому інтервалі, на краях можуть виникати значні відхилення. Це явище відоме як "ефект Рунге" і може погіршувати точність інтерполяції.
4. **Числова стійкість**: Зі збільшенням кількості вузлів обчислення можуть ставати нестійкими через накопичення похибок. Використання спеціальних стратегій вибору вузлів, наприклад, вузлів Чебишова, може зменшити цю проблему.

Висновки

Метод поліноміальної інтерполяції дозволяє будувати точні апроксимації для функцій, значення яких відомі у кількох точках. Однак для досягнення найкращих результатів важливо враховувати вибір вузлів, можливі похибки та числову стабільність обчислень.

2.2 Інші методи інперполяції: методи Лагранжа і Ньютона, кубічні сплайни

Враховуючи набір даних, **поліноміальна інтерполяція** - це метод пошуку поліноміальної функції, яка точно відповідає набору точок даних. Хоча існує кілька методів пошуку цього многочлена, сам поліном унікальний, що ми доведемо пізніше.

Многочлен Лагранжа

Одним з найпоширеніших способів виконання поліноміальної інтерполяції є використання многочлена Лагранжа. Щоб використовувати цей метод, ми починаємо з побудови полінома, який проходить через 2 точки (x_0, y_0)і (x_1, y_1). Ми використовуємо два рівняння з алгебри рівня коледжу.

$$y - y_1 = m(x - x_1) \text{ та } m = \frac{y_1 - y_0}{x_1 - x_0}$$

Поєднуючи їх, ми в кінцевому підсумку отримаємо :

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_1) + y_1$$

Тепер, щоб вивести формулу, подібну до тієї, що використовується для многочлена Лагранжа, ми виконуємо деяку алгебру. Ми починаємо з заміни умов $y_1 - y_0$ та $x_1 - x_0$

$$y = \frac{x - x_1}{x_1 - x_0}(y_1 - y_0) + y_1$$

Розкриючи дужки одержимо :

$$y = \frac{x - x_1}{x_1 - x_0}y_1 - \frac{x - x_1}{x_1 - x_0}y_0 + y_1$$

Домножимо останній доданок на вираз $\frac{x_1 - x_0}{x_1 - x_0} = 1$ і отримаємо :

$$y = \frac{x - x_1}{x_1 - x_0}y_1 - \frac{x - x_1}{x_1 - x_0}y_0 + \frac{x_1 - x_0}{x_1 - x_0}y_1$$

Зведемо подібні доданки :

$$y = -\frac{x - x_1}{x_1 - x_0}y_0 + \frac{x - x_0}{x_1 - x_0}y_1$$

Позбудемось від’ємного знака :

$$y = \frac{x - x_1}{x_0 - x_1}y_0 + \frac{x - x_0}{x_1 - x_0}y_1$$

З наведеного вище можна побачити, що якщо $x = x_1$, то перший доданок обнуляється, а другий стає $1 \cdot y_1 = y_1$, як і очікувалося. Аналогічно, якщо $x = x_0$, то другий доданок обнуляється, а перший стає $1 \cdot y_0 = y_0$, що теж відповідає очікуванням.

Для трьох точок (x_0 , y_0) , (x_1 , y_1) та (x_2 , y_2) можна записати поліном, який працює за тим самим принципом:

$$y = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} y_0 + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} y_1 + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} y_2$$

Якщо, наприклад, підставити $x = x_1$, то перший і третій доданки обнуляються, а середній стає $1 \cdot y_1 = y_1$, як і потрібно. Аналогічно можна перевірити інші випадки. У результаті цей підхід дає повний поліном Лагранжа.

$$y = \sum_{i=0}^{n} (y_i \prod_{j=0, j \neq i}^{n} \frac{x - x_i}{x_i - x_j})$$

У наведеному вище поліномі кожен чисельник записаний таким чином, що при $x = x_i$ всі коефіцієнти обнуляються, окрім коефіцієнта при y_i , який дорівнює y_i, . Таким чином, цей поліном проходить через усі задані точки, і, як можна перевірити, є поліномом степеня n .

Многочлен Ньютона

Інтерполяція Ньютона є альтернативою поліному Лагранжа. Хоча вона може здаватися складнішою, вона дозволяє виконувати поступову інтерполяцію та забезпечує ефективний спосіб знайти явну формулу $y = a_0 + a_1 x + \cdots + a_n x^n$.

Метод Ньютона полягає у знаходженні коефіцієнтів та використанні їх для обчислення наступних коефіцієнтів. Оскільки важливою частиною цього методу є можливість поступового додавання точок, почнемо з однієї точки та продемонструємо обчислення.

Для однієї точки (x_0 , y_0) розрахунок простий:

$$b_0 = y_0$$

А поліном буде :

$$y = b_0$$

Додамо нову точку (x_1 , y_1) .Наступний коефіцієнт b_1 , зазвичай позначається як $[y_0 , y_1]$:

$$b_1 = [y_0 , y_1] = \frac{y_1 - y_0}{x_1 - x_0}$$

Поліном у цьому випадку записується як:

$$y = b_0 + b_1(x - x_0)$$

Зверніть увагу, що нам не потрібно повторно обчислювати весь поліном, а лише додати новий коефіцієнт до $(x - x_0)$.

Додамо третю точку (x_2, y_2) . Тоді коефіцієнт $b_2 = [y_0, y_1, y_2]$ обчислюється за формулою:

$$b_2 = [y_0, y_1, y_2] = \frac{[y_1, y_2] - [y_0, y_1]}{x_2 - x_0} = \frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0}$$

Поліном у цьому випадку набуває вигляду:

$$y = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

Можна помітити, що попередні визначення мають рекурсивний характер. Це продовжується для загального випадку інтерполяції Ньютона.

$$y = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1) \ldots (x - x_{n-1})$$

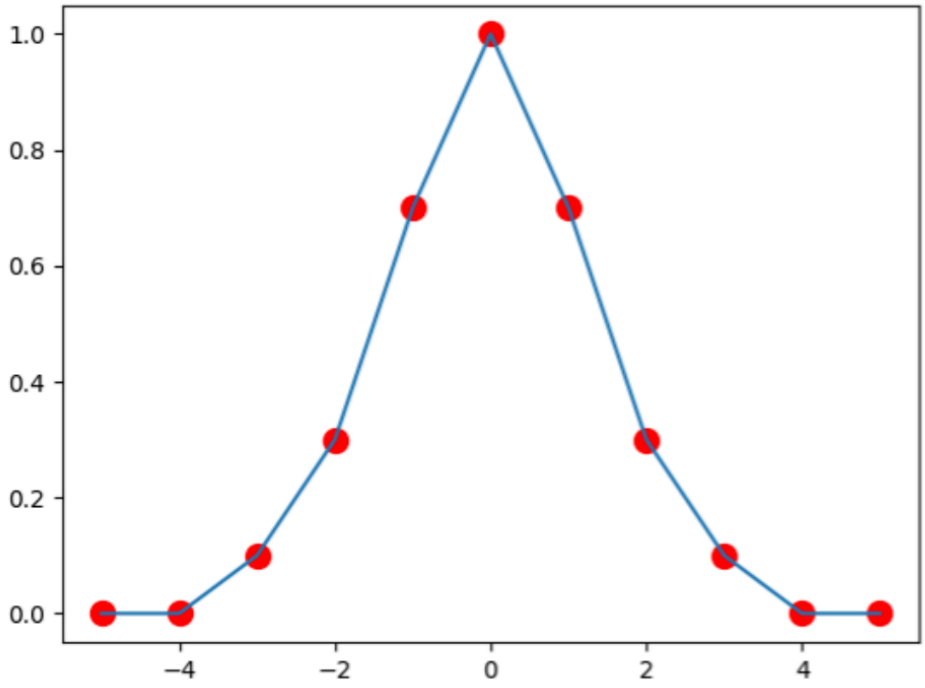
де

$$b_n = [y_0, y_2, \ldots, y_n] = \frac{[y_1, y_2, \ldots, y_n] - [y_0, y_1, \ldots, y_{n-1}]}{x_n - x_0}$$

Кубічні сплайни

Сплайн — це функція, визначена частинами за допомогою поліномів. Замість того, щоб використовувати один поліном, який проходить через усі точки даних, як ми робили раніше, ми визначаємо кілька поліномів між кожною парою точок. Хоча такий підхід потребує більше зусиль, він зазвичай кращий, оскільки дає точні результати та уникає явища Рунге.

Лінійний сплайн будується шляхом з'єднання точок прямими лініями. Використовуючи наші попередні дані, можна створити доволі точне наближення вихідної функції.



Хоч це й простий підхід, отримане наближення зазвичай є більш надійним порівняно з результатами, які дає метод Ньютона, завдяки відсутності явища Рунге. Щоб отримати ще точніше зображення, замість прямих ліній можна використовувати кубічні поліноми. У цьому випадку кількість таких поліномів дорівнюватиме кількості інтервалів між точками даних (на одиницю менше, ніж кількість самих точок). Отриману функцію позначимо як $s(x)$, і вона визначається для кожного інтервалу окремо.

2.3 Практичне застосування методів інтерполяції

Многочлен Лагранджа

Нехай маємо три точки (1,2) , (2,3) та (4,5)

Запишемо базові поліноми

$$L_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} = \frac{(x-2)(x-4)}{(1-2)(1-4)} = \frac{(x-2)(x-4)}{3}$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} = \frac{(x-1)(x-4)}{(2-1)(2-4)} = \frac{(x-1)(x-4)}{-2}$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)} = \frac{(x-1)(x-2)}{(4-1)(4-2)} = \frac{(x-1)(x-2)}{6}$$

Запишемо многочлен P(x) :

$$P(x) = y_0 \cdot L_0(x) + y_1 \cdot L_1(x) + y_2 \cdot L_2(x)$$

Підставляємо значення $y_0 = 2$, $y_1 = 3$, $y_2 = 5$

$$P(x) = 2 \cdot \frac{(x-2)(x-4)}{3} + 3 \cdot \frac{(x-1)(x-4)}{-2} + 5 \cdot \frac{(x-1)(x-2)}{6}$$

Порозкриваємо дужки , зведемо подібні доданки і отримаємо кінцевий результат

$$P(x) = x + \frac{11}{3}$$

Це і є інтерполяційний многочлен Лагранжа для заданих точок .

Многочлен Ньютона

Використовування методу Ньютона для обчислення многочлена через точки (1,3),(5,7) та (8,0).

Для цього прикладу ми використаємо таблицю, щоб упорядкувати наші дані.

x_0	y_0		
x_1	y_1	$[y_1, y_0]$	
x_2	y_2	$[y_2, y_1]$	$[y_0, y_1, y_2]$

Ми починаємо з внесення наших даних:

Зауважимо що x_i , y_i - координати заданих точок. Тобто перший рядок це координати першої точки , другий рядок (перші два стовпці) – координати другої точки , а третій (перші два стовпці) – третьої .

1	3		
5	7	$[y_1, y_0]$	
8	0	$[y_2, y_1]$	$[y_0, y_1, y_2]$

Потім ми робимо розрахунки і заповнюємо ці дані:

$$[y_1, y_0] = \frac{y_1 - y_0}{x_1 - x_0} = \frac{7-3}{5-1} = 1$$

$$[y_2, y_1] = \frac{y_2 - y_1}{x_2 - x_1} = \frac{0-7}{8-5} = -\frac{7}{3}$$

1	3		
5	7	1	
8	0	$-\frac{7}{3}$	$[y_0, y_1, y_2]$

Тепер обчислюємо залишковий елемент, використовуючи раніше обчислені значення:

$$[y_0, y_1, y_2] = \frac{[y_1, y_2] - [y_0, y_1]}{x_2 - x_0} = \frac{-\frac{7}{3} - 1}{8-1} = -\frac{10}{21}$$

1	3		
5	7	1	
8	0	$-\frac{7}{3}$	$-\frac{10}{21}$

Зверніть увагу, що b_0 , b_1 *і* b_2 записані у верхній діагоналі. Отже, наш остаточний многочлен має вигляд:

$$y = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

$$b_0 = y_0$$

$$b_1 = [y_1, y_0]$$

$$b_2 = [y_0, y_1, y_2]$$

Отже одержимо :

$$y = 3 + 1(x-1) - \frac{10}{21}(x-1)(x-5)$$

Кубічні сплайни

Маємо температурні вимірювання, зроблені кожні 6 годин:

Час (години)	**Температура (у градусах)**

0	10
6	8
12	15
18	14
24	10

Необхідно знайти температуру в проміжних точках, наприклад, о 9:00.

Задамо точки : x= [0,6,12,18,24] , y= [10,8,15,14,10]

Нагадаємо : кубічний сплайн – це набір кубічних поліномів $S_i(x)$, визначених на кожному інтервалі $[x_i , x_{i+1}]$:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Звернемо увагу на декілька умов :

1. Сплайн проходить через усі вузли : $S_i(x_i) = y_i , S_i(x_{i+1}) = y_{i+1}$
2. Гладкість : перші й другі похідні рівні на стиках інтервалів
3. Друга похідна на краях часто прирівнюється до нуля

Обчислимо коефіцієнти :

Стовпець коефіцієнтів a – це температура .

$$a_0 = 10 , a_1 = 8 , a_2 = 15 , a_3 = 14$$

Кроки між вузлами – рівномірні $h_0 = h_1 = h_2 = h_3 = 6$

Використовуючи умову гладкості другої похідної , запишемо систему рівнянь для c_i.

$$\boldsymbol{S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3}$$

Знайдемо другу похідну $S_i''(x) = 2c_i + 6d_i(x - x_i)$

Застосуємо умову неперервності другої похідної :

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$$

Одержемо

$$2c_i + 6d_i h_i = 2c_{i+1}$$

$$\boldsymbol{d_i = \frac{c_{i+1} - c_i}{3h_i}}$$

Застосуємо умову неперервності першої похідної :

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$$

Одержемо

$$b_i + 2c_i h_i + 3d_i h_i^2 = b_{i+1}$$

$$\boldsymbol{b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{h_i}{3}(2c_i + c_{i+1})}$$

$$b_{i+1} = \frac{y_{i+2} - y_{i+1}}{h_{i+1}} - \frac{h_{i+1}}{3}(2c_{i+1} + c_{i+2})$$

Підставляємо b_i , b_{i+1} та d_i в умову неперервності першої похідної . І одержуємо

$$\boldsymbol{h_i(c_{i-1} + 4c_i + c_{i+1}) = 3(\frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}})}$$

Для i=1,2,3.

Знайдемо коефіцієнти c_i :

i=1:

$$h_1(c_0 + 4c_1 + c_2) = 3(\frac{y_2 - y_1}{h_1} - \frac{y_1 - y_0}{h_0})$$

$$6(c_0 + 4c_1 + c_2) = 3(\frac{15 - 8}{6} - \frac{8 - 10}{6})$$

$$\boldsymbol{2(c_0 + 4c_1 + c_2) = \frac{3}{2}}$$

i=2:

$$h_2(c_1 + 4c_2 + c_3) = 3(\frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1})$$

$$6(c_1 + 4c_2 + c_3) = 3(\frac{14 - 15}{6} - \frac{15 - 8}{6})$$

$$\boldsymbol{2(c_1 + 4c_2 + c_3) = -\frac{4}{3}}$$

i=3:

$$h_3(c_2 + 4c_3 + c_4) = 3(\frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2})$$

$$6(c_2 + 4c_3 + c_4) = 3(\frac{10 - 14}{6} - \frac{14 - 15}{6})$$

$$\boldsymbol{2(c_2 + 4c_3 + c_4) = -\frac{1}{2}}$$

Зауважимо , що $c_0 = 0$ – крайова умова (друга похідна на кінцях дорівнює нулю)
Розв'яжимо систему і одержимо розв'язки :

$$c_1 = 0.125 \;\; c_2 = -0.625 \;\; c_3 = -0.25$$

Підставимо дані в b_i і знайдемо коефіцієнти b_i :

$$b_0 = -0.7 \;\; b_1 = 1.375 \;\; b_2 = -0.2083 \;\; b_3 = -0.1667$$

Аналогічно шукаємо d_i

Сформулюємо таблицю коефіцієнтів :

Інтервал	**a**	**b**	**c**	**d**
[0,6]	10	-0.7	0	0.0069
[6,12]	8	1.375	0.125	-0.0417
[12,18]	15	-0.2083	-0.625	0.0208
[18,24]	14	-0.1667	-0.25	0.0139

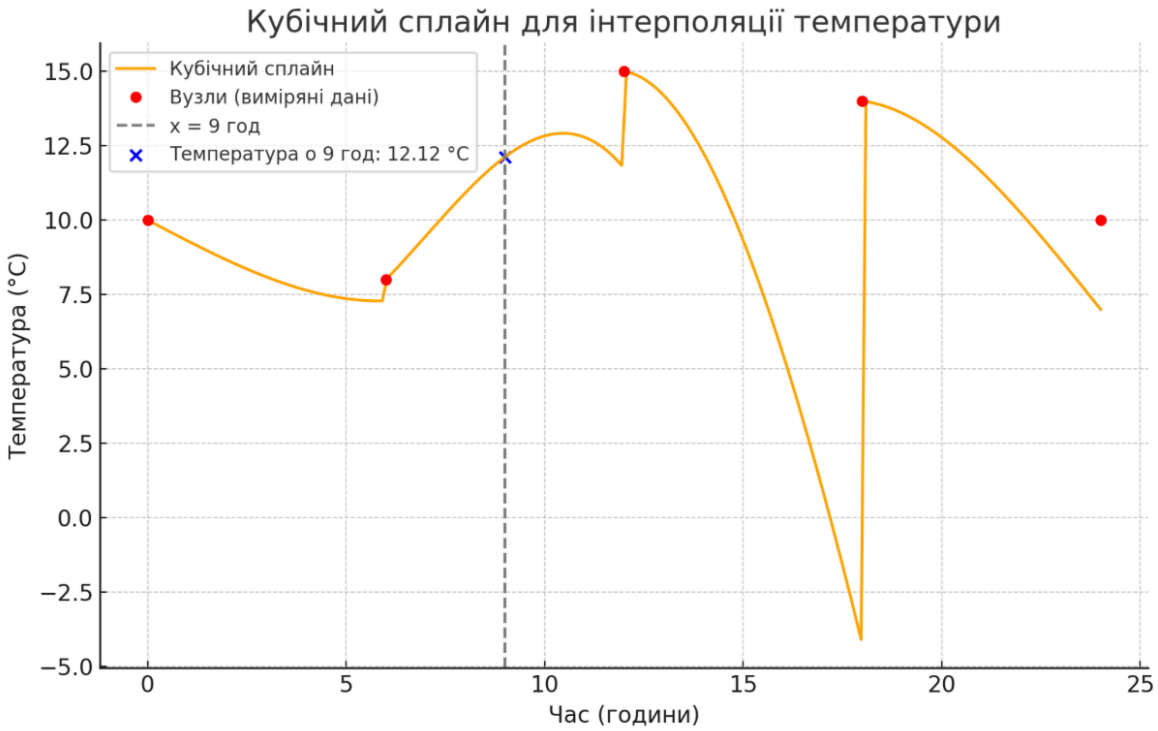
Для x=9 , шукаємо інтервал [6,12] і підставляємо значення в відповідний поліном .

Одержимо

$$\boldsymbol{S_1(x) = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3}$$

$$\boldsymbol{S_1(x) = 8 + 1.375(x - 6) + 0.125(x - 6)^2 - 0.0417(x - 6)^3}$$

Розрахунки за допомогою сплайнів дадуть температуру приблизно S(9)=12.12



Ось графік, який показує кубічний сплайн для інтерполяції температури. Червоні точки позначають виміряні значення, синя точка — інтерпольовану температуру о 9:00, а крива — плавну залежність температури від часу.

Висновки до розділу II

Цей розділ акцентує увагу на ключових математичних методах інтерполяції та їх практичному застосуванні. Поліноміальна інтерполяція, методи Лагранжа і Ньютона, а також сплайни пропонують широкі можливості для використання в наукових та технічних сферах. Вибір найкращого методу для конкретного завдання вимагає врахування їх переваг і обмежень.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРПОЛЯЦІЇ У ВІЗУАЛІЗАЦІЇ РУХУ ПЕРСОНАЖА

3.1 Постановка завдання

Програмна частина дипломного проекту спрямована на створення графічної демонстрації, яка ілюструє застосування методів інтерполяції функцій у комп’ютерній графіці. В рамках цього проекту розроблено анімацію персонажа, який рухається в заданому просторі, долаючи певні обмеження та унікальні перешкоди на своєму шляху. Для розробки графічної складової використано бібліотеку OpenGL, що надає потужні інструменти для побудови як двовимірної, так і тривимірної графіки, забезпечуючи високий рівень візуалізації та плавність анімації.

Основна концепція полягає у створенні просторової сцени, що складається зі звичайних математичних фігур, які у сукупності формують навколишнє середовище, через яке рухається персонаж. Відсутність складних елементів, таких як дороги або будівлі, дозволяє зосередитися на технічній складовій — застосуванні інтерполяційних алгоритмів для досягнення плавної і природної зміни положення об'єктів у часі.

Ключові завдання даного проекту включають:

- Розробку візуального середовища на основі базових геометричних фігур, що формують простір навколо анімованого персонажа.
- Реалізацію переміщення персонажа з дотриманням плавності руху за допомогою інтерполяційних методів, які забезпечують природню анімацію та уникають різких змін положення.
- Створення механізму обходу окремої перешкоди персонажем під час руху, що демонструє здатність системи реагувати на просторові обмеження.
- Забезпечення динамічної зміни координат об'єктів сцени, яка змінюється з часом відповідно до заданих траєкторій, що створює відчуття живого середовища і підкреслює особливості інтерполяції.

Результатом реалізації цього проекту є невелика, але технічно вдосконалена програма, яка ілюструє базові принципи комп’ютерної графіки та анімації, використовуючи методи математичної інтерполяції у контексті 2D-візуалізації з допомогою OpenGL. Такий підхід сприяє кращому розумінню алгоритмічних основ графічної анімації і може бути корисним для подальших досліджень та розробок у сфері інтерактивної графіки.

3.2 Використання інтерполяції в графіці

Інтерполяція є важливим математичним інструментом, що дозволяє генерувати нові дані на основі вже відомих точок у дискретному наборі. У контексті комп'ютерної графіки ця техніка відіграє ключову роль, оскільки часто виникає потреба в отриманні значень функцій для обмеженої кількості значень незалежної змінної. Це дозволяє створювати плавні криві та детальні поверхні, що значно підвищує якість візуалізації.

Основна мета інтерполяції полягає в знаходженні адекватного математичного виразу, який точно відображає задану криву. Цей процес є незамінним, коли потрібно зобразити криву, визначаючи проміжні точки між відомими даними, що дозволяє досягти високої точності та естетичності в графічних зображеннях.

Існує кілька основних методів інтерполяції, кожен з яких має свої особливості та сфери застосування:

1. **Зважена зворотна відстань** : Цей метод оцінює значення в кожній комірці шляхом усереднення значень сусідніх точок даних. Важливість кожної точки визначається її відстанню до центру комірки: чим ближча точка, тим більший її вплив на результат. Це дозволяє отримати більш точні оцінки в районах, де дані є розрідженими.
2. **Крігінг**: Крігінг є геостатистичним методом, який враховує як відстань, так і ступінь варіації між відомими точками даних. Цей підхід дозволяє створити розрахункову поверхню на основі розсіяного набору точок, що забезпечує точнішу інтерполяцію в невідомих областях.
3. **Природний сусід**: Метод Natural Neighbour визначає найближчу підмножину вхідних зразків для точки запиту. Він використовує ваги, які пропорційні площам, що оточують вхідні зразки, для обчислення інтерполяційного значення. Цей метод також відомий як інтерполяція "розкрадіжки" і забезпечує високу точність.
4. **Сплайн**: Сплайн-інтерполяція використовує математичні функції, які мінімізують загальну кривизну поверхні, створюючи гладку криву, що проходить через усі вхідні точки. Цей метод є особливо корисним для створення естетично привабливих графічних зображень.
5. **Сплайн з бар'єрами**: Цей метод є варіацією сплайн-інтерполяції, яка враховує розриви, закодовані в даних. Сплайн з бар'єрами дозволяє зберігати важливі особливості даних, що робить його корисним у випадках, коли потрібно врахувати зміни в структурі даних.
6. **Топо до растру**: Цей метод спеціально розроблений для створення поверхонь, які точно відображають природні дренажні системи. Він забезпечує кращу збереженість потокових мереж на основі вхідних контурних даних, що робить його важливим у географічних інформаційних системах.

7. **Тренд**: Глобальна поліноміальна інтерполяція, відома як тренд, відповідає гладкій поверхні, визначеній математичною функцією (поліномом), до вхідних точок. Цей метод дозволяє виявити масштабні закономірності в даних, що може бути корисним для аналізу та прогнозування.

Під час реалізації алгоритму руху персонажа було застосовано принципи, подібні до методу **зваженого оберненого врахування відстані (IDW)**. Зокрема, вибір напрямку руху базувався на аналізі відстаней до об'єктів середовища: що ближчим був об'єкт (наприклад, школа чи перешкода), то більшою була його вага у процесі прийняття рішення щодо руху. Для досягнення плавності переміщення персонажа використовувався підхід, схожий на **інтерполяцію за допомогою сплайнів** (пункт 4). Це дало змогу створити згладжену траєкторію руху, яка виглядає природно й естетично на візуальному рівні. Крім того, зважаючи на наявність об’єктів, які не можна перетинати (наприклад, паркани), логіка руху враховувала обмеження, що співвідносяться з принципами **сплайн-інтерполяції з бар'єрами** (пункт 5). Завдяки цьому персонаж міг обирати шлях, що оминає перешкоди, зберігаючи при цьому плавність маршруту.

Отже, при моделюванні руху було враховано елементи трьох методів: 1 (IDW), 4 (сплайн) та 5 (сплайн з бар’єрами). Таке поєднання дозволило досягти балансу між точністю навігації, візуальною плавністю траєкторії та адаптацією до особливостей віртуального середовища.

Завдяки різноманіттю методів інтерполяції, комп'ютерна графіка отримує можливість створювати більш реалістичні та естетично привабливі зображення, що значно підвищує якість візуалізації та взаємодії з користувачем. Використання інтерполяційних технік дозволяє не лише покращити візуальні ефекти, але й забезпечити плавність анімацій, що робить графічні проекти більш динамічними та інтерактивними. Це, в свою чергу, відкриває нові горизонти для творчості та інновацій у сфері комп'ютерної графіки, дозволяючи дизайнерам і розробникам реалізовувати свої ідеї з високим рівнем деталізації та точності.

3.3 Інструменти та середовище розробки

Для успішної реалізації графічного проекту було обрано набір сучасних інструментів та технологій, які забезпечують ефективну розробку та стабільне виконання програми.

В якості основної мови програмування використано **C++**, що є оптимальним вибором для графічних застосунків завдяки високій продуктивності та гнучкості у роботі з апаратними ресурсами.

Візуалізація та відтворення графіки реалізовані за допомогою бібліотеки **OpenGL**, котра надає потужний набір засобів для програмування 2D та 3D-графіки. Для організації роботи з вікнами, обробки подій та спрощення взаємодії з операційною системою застосовано **GLUT**. Важливо відзначити, що для підтримки кросплатформенності проекту та розробки під macOS була використана стандартна збірка **Xcode**, що дозволяє здійснювати компіляцію і запуск програми без підключення додаткових зовнішніх бібліотек.

Вибір **Xcode** як середовища розробки пов’язаний із зручністю налаштування, широкою підтримкою сучасних стандартів C++ і інтеграцією з графічними API на macOS, що значно покращує продуктивність процесу розробки і налагодження. Завдяки цьому забезпечується стислий цикл розробки без необхідності установки додаткових пакетів чи утиліт.

Таким чином, основні складові середовища розробки проекту включають:

- **Мову програмування C++** – для створення високоефективного та оптимізованого коду;
- **OpenGL** – графічний API, що реалізує рендеринг сцени;
- **GLUT** – бібліотека допоміжних функцій для роботи з вікнами і подіями;
- **Xcode** – комплексне середовище розробки для macOS, що забезпечує компіляцію, відлагодження та управління проєктом без необхідності зовнішніх доповнень.

Використання поєднання цих інструментів гарантує швидку, стабільну та зручну розробку графічної програми, адаптованої для роботи на різних платформах, зокрема в середовищі macOS.

3.4 Основні компоненти програми

Програма складається з кількох логічних блоків, кожен з яких виконує свою специфічну функцію, що забезпечує інтерактивність та динамічність графічного середовища. Основні компоненти програми включають:

1. **Модуль ініціалізації сцени**: Цей модуль відповідає за створення вікна програми та налаштування початкових параметрів сцени. Він встановлює основні елементи, такі як освітлення, яке визначає, як об'єкти будуть виглядати в різних умовах, а також налаштовує камеру для коректного відображення сцени з потрібного кута. Крім того, модуль створює координатну сітку, що допомагає візуалізувати простір і орієнтуватися в ньому.

```
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Man Going to School");

    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-5.0, 5.0, -3.0, 3.0);

    glutDisplayFunc(display);
    glutTimerFunc(0, update, 0);
    glutKeyboardFunc(keyboard);

    glutMainLoop();
    return 0;

}
```

2. **Модуль створення об’єктів**: У цьому модулі реалізовані функції для малювання основних елементів сцени, таких як прості геометричні фігури, які формують навколишнє середовище. Замість традиційних об'єктів, таких як дороги чи дерева, програма зосереджується на створенні абстрактних форм, які можуть включати різноманітні фігури, що представляють перешкоди або інші елементи середовища. Це дозволяє зосередитися на візуалізації та анімації без зайвих деталей.

```
// Функція для малювання сонця
void drawSun() {
    glColor3f(1.0f, 1.0f, 0.0f);// Жовтий колір для сонця
    float radius = 0.3f;
    float x = 4.0f, y = 2.5f;
    glBegin(GL_TRIANGLE_FAN);// Малюємо сонце як багато граней
    glVertex2f(x, y);
    for (int i = 0; i <= 100; i++) {
        float angle = 2.0f * 3.14159f * i / 100;
```

```
            glVertex2f(x + cos(angle) * radius, y + sin(angle) * radius);
        }
        glEnd();
    }


    // Функція для малювання квітки
    void drawFlower(float x, float y) {
        glColor3f(1.0f, 0.0f, 1.0f); // пелюстки
        for (int i = 0; i < 8; ++i) {
            float angle = i * 3.14159f / 4.0f;
            float dx = cos(angle) * 0.03f;
            float dy = sin(angle) * 0.03f;
            glBegin(GL_POLYGON);// Малюємо пелюстки
            for (int j = 0; j < 20; ++j) {
                float a = j * 2 * 3.14159f / 20;
                glVertex2f(x + dx + cos(a) * 0.01f, y + dy + sin(a) *
    0.01f);
            }
            glEnd();
        }

        glColor3f(1.0f, 1.0f, 0.0f); // серцевина
        glBegin(GL_POLYGON);
        for (int j = 0; j < 20; ++j) {
            float a = j * 2 * 3.14159f / 20;
            glVertex2f(x + cos(a) * 0.01f, y + sin(a) * 0.01f);
        }
        glEnd();
    } та інші…
```

3. **Модуль анімації персонажа**: Цей модуль відповідає за динаміку руху персонажа в сцені. Він реалізує оновлення координат персонажа за допомогою інтерполяції, що дозволяє досягти плавного та природного переміщення. Персонаж може підстрибувати, додаючи елемент динаміки до анімації, проте не зупиняється, що створює враження безперервного руху. Це дозволяє зосередитися на динаміці та емоційності анімації, підкреслюючи активність персонажа в середовищі.

```
    void update(int value) {
        // Якщо людина рухається до школи
        if (isMoving) {
            // Якщо людина ще не досягла школи, рухаємось
            if (manX < schoolX) {
                // Якщо на шляху перешкода, і людина не уникала її раніше
                if (avoidStage == 0 && isCollidingWithObstacle() &&
    !hasAvoidedObstacle) {
                    // Починаємо процес уникання перешкоди
                    avoidStage = 1;
                    avoidStartX = manX;
                    avoidStartY = manY;
                }
                // Якщо етап уникання 1, людина починає підніматися
                if (avoidStage == 1) {
                    manY += 0.02f;
                    // Піднімаємо людину
```

```
                                // Якщо досягли певної висоти,
переходимо до етапу 2
                if (manY >= 0.3f) {
                    avoidStage = 2;
                }
            }// Етап уникання 2, рухаємо людину вбік, щоб обійти
перешкоду
            else if (avoidStage == 2) {
                manX += 0.02f;// Переміщаємо людину вбік
                // Якщо обійшли перешкоду, переходимо до етапу 3
                if (manX > obstacleX + obstacleWidth) {
                    avoidStage = 3;
                    hasAvoidedObstacle = true; //  більше не обробляти
паркан
                }
            }
            // Етап уникання 3, опускаємо людину назад
            else if (avoidStage == 3) {
                manY -= 0.02f;// Опускаємо людину
                // Якщо досягли початкової висоти, повертаємось до
етапу 0
                if (manY <= avoidStartY) {
                    manY = avoidStartY;// Встановлюємо людину назад на
початкову висоту
                    avoidStage = 0;
                }
            }
            // Якщо уникнення не потрібно, просто рухаємось вперед
            else {
                manX += 0.02f;
            }
            // Якщо людина досягла школи, зупиняємо рух
            if (manX >= schoolX - 0.5f) {
                isMoving = false;// Зупиняємо рух
                hasReachedSchool = true;// Відмічаємо, що людина
досягла школи
            }
        }
    }

    // Якщо людина стрибає, підвищуємо висоту стрибка
    if (isJumping) {
        jumpHeight += jumpSpeed;// Підвищуємо висоту стрибка
        // Якщо досягли максимальної висоти, припиняємо стрибок
        if (jumpHeight >= 0.5f) {
            isJumping = false;// Закінчуємо стрибок
        }
    }
    // Якщо стрибок завершено, знижуємо висоту
    else if (jumpHeight > 0.0f) {
        jumpHeight -= jumpSpeed;// Зменшуємо висоту стрибка
    }

    glutPostRedisplay();
    // Викликаємо цю функцію кожні 16 мілісекунд (близько 60 FPS)
    glutTimerFunc(16, update, 0);
}
```

4. **Модуль обробки подій**: Цей модуль відповідає за взаємодію користувача з програмою. Він реагує на натискання клавіш, що дозволяє запускати або зупиняти анімацію, а також виконувати інші дії, пов'язані з управлінням персонажем. Завдяки цьому модулю, користувач може активно взаємодіяти з програмою, що робить досвід більш інтерактивним і захоплюючим.

```
void keyboard(unsigned char key, int x, int y) {
    if (key == ' ') {
        isMoving = true;
    }
    if (key == 'j' && !isJumping) {
        isJumping = true;
    }

}
```

Кожен з цих модулів працює в тісній взаємодії з іншими, створюючи цілісну та динамічну графічну програму, що дозволяє користувачеві насолоджуватися плавною анімацією та інтерактивним середовищем.

3.5 Результати виконання програми

У результаті реалізації проєкту було створено програму з використанням бібліотеки OpenGL, яка демонструє рух персонажа (учня) до школи з можливістю уникнення перешкод. Програма моделює процес взаємодії користувача з візуальним середовищем, у якому персонаж автоматично рухається вперед, реагуючи на перешкоди на своєму шляху.

Основна логіка реалізована у функції update(), яка періодично викликається таймером glutTimerFunc() та забезпечує анімацію.

```
void update(int value) {
    // Якщо людина рухається до школи
    if (isMoving) {
        // Якщо людина ще не досягла школи, рухаємось
        if (manX < schoolX) {
            // Якщо на шляху перешкода, і людина не уникала її раніше
            if (avoidStage == 0 && isCollidingWithObstacle() &&
!hasAvoidedObstacle) {
                // Починаємо процес уникання перешкоди
                avoidStage = 1;
                avoidStartX = manX;
                avoidStartY = manY;
            }
            // Якщо етап уникання 1, людина починає підніматися
            if (avoidStage == 1) {
                manY += 0.02f;
                // Піднімаємо людину
                                // Якщо досягли певної висоти, переходимо до
етапу 2
                if (manY >= 0.3f) {
                    avoidStage = 2;
                }
            }// Етап уникання 2, рухаємо людину вбік, щоб обійти перешкоду
            else if (avoidStage == 2) {
                manX += 0.02f;// Переміщаємо людину вбік
                // Якщо обійшли перешкоду, переходимо до етапу 3
                if (manX > obstacleX + obstacleWidth) {
                    avoidStage = 3;
                    hasAvoidedObstacle = true; //  більше не обробляти паркан
                }
            }
            // Етап уникання 3, опускаємо людину назад
            else if (avoidStage == 3) {
                manY -= 0.02f;// Опускаємо людину
                // Якщо досягли початкової висоти, повертаємось до етапу 0
                if (manY <= avoidStartY) {
                    manY = avoidStartY;// Встановлюємо людину назад на початкову
висоту
                    avoidStage = 0;
                }
            }
            // Якщо уникнення не потрібно, просто рухаємось вперед
            else {
                manX += 0.02f;
            }
            // Якщо людина досягла школи, зупиняємо рух
            if (manX >= schoolX - 0.5f) {
                isMoving = false;// Зупиняємо рух
                hasReachedSchool = true;// Відмічаємо, що людина досягла школи
```

```
            }
        }
    }

    // Якщо людина стрибає, підвищуємо висоту стрибка
    if (isJumping) {
        jumpHeight += jumpSpeed;// Підвищуємо висоту стрибка
        // Якщо досягли максимальної висоти, припиняємо стрибок
        if (jumpHeight >= 0.5f) {
            isJumping = false;// Закінчуємо стрибок
        }
    }
    // Якщо стрибок завершено, знижуємо висоту
    else if (jumpHeight > 0.0f) {
        jumpHeight -= jumpSpeed;// Зменшуємо висоту стрибка
    }

    glutPostRedisplay();
    // Викликаємо цю функцію кожні 16 мілісекунд (близько 60 FPS)
    glutTimerFunc(16, update, 0);
}

void keyboard(unsigned char key, int x, int y) {
    if (key == ' ') {
        isMoving = true;
    }
    if (key == 'j' && !isJumping) {
        isJumping = true;
    }

}
```

Ця функція керує логікою руху персонажа. Якщо на шляху перешкода, активується процедура обходу, яка включає три етапи: підйом, горизонтальний обхід та спуск.

Результати роботи програми

У процесі виконання програми користувач бачить анімовану сцену:

- персонаж рухається по шляху до школи;
- на його шляху перешкода;
- персонаж автоматично здійснює обхід;
- при досягненні цілі (школи) рух зупиняється.

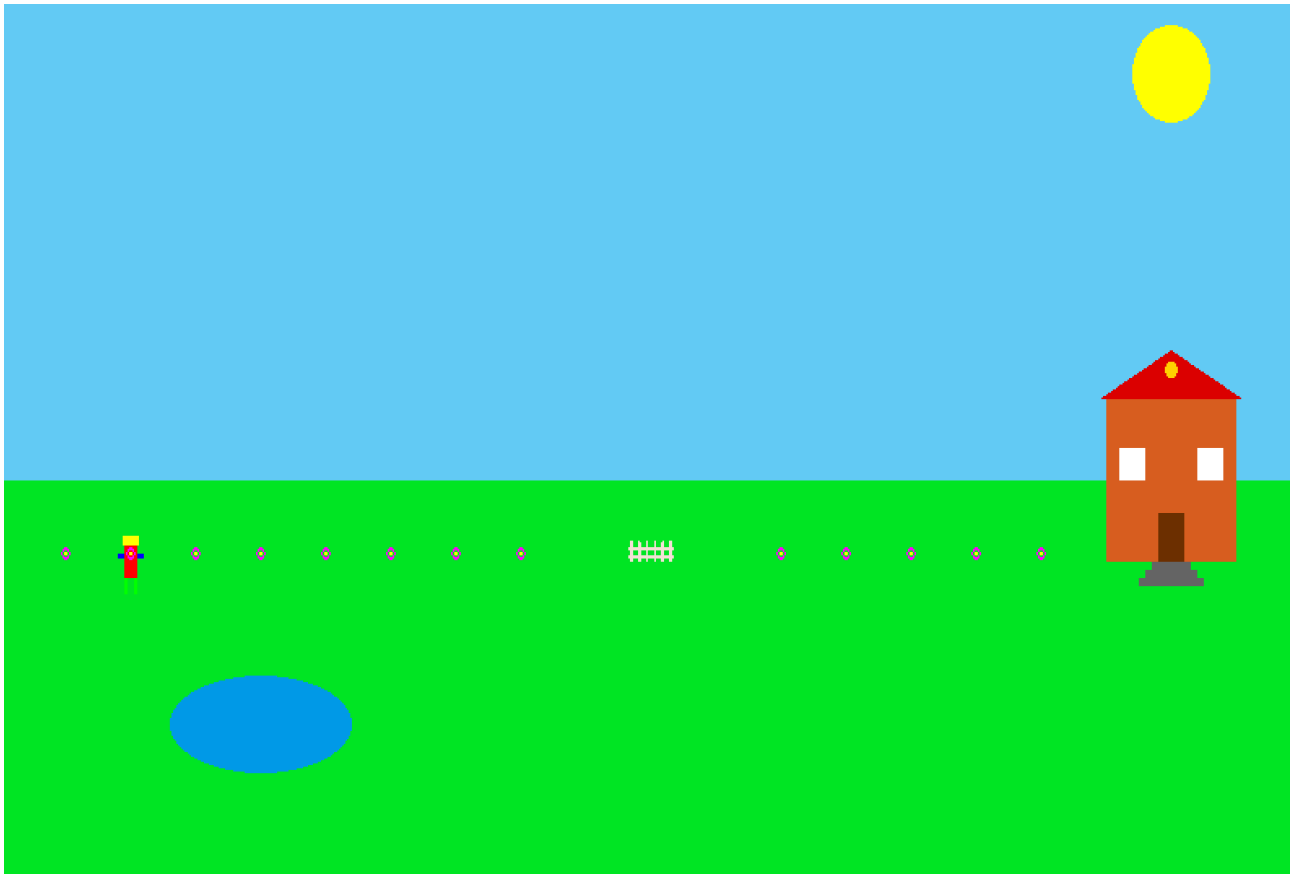


Рисунок 3.5.1 – Початкове положення персонажа

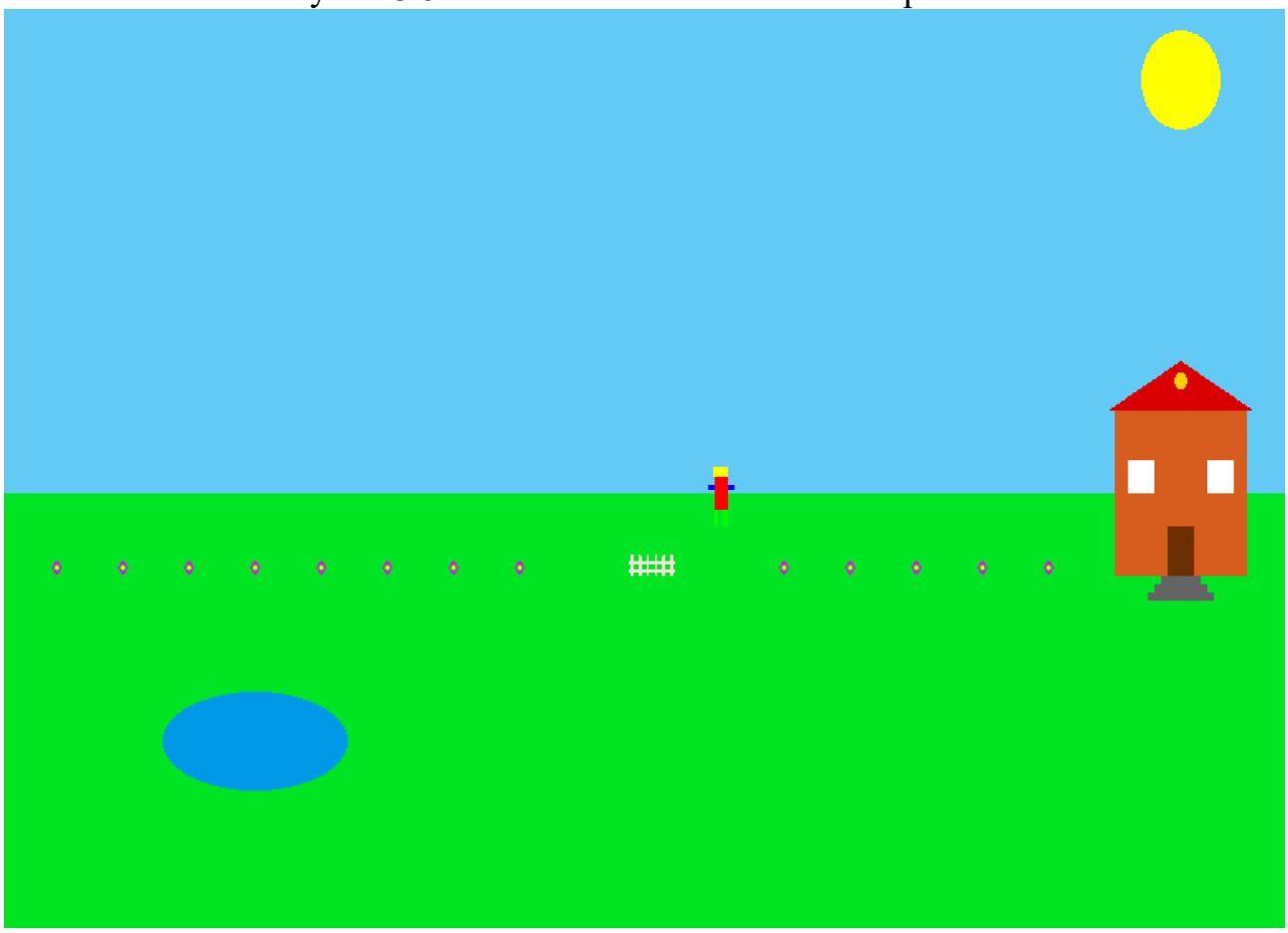


Рисунок 3.5.2 – Процес уникнення перешкоди

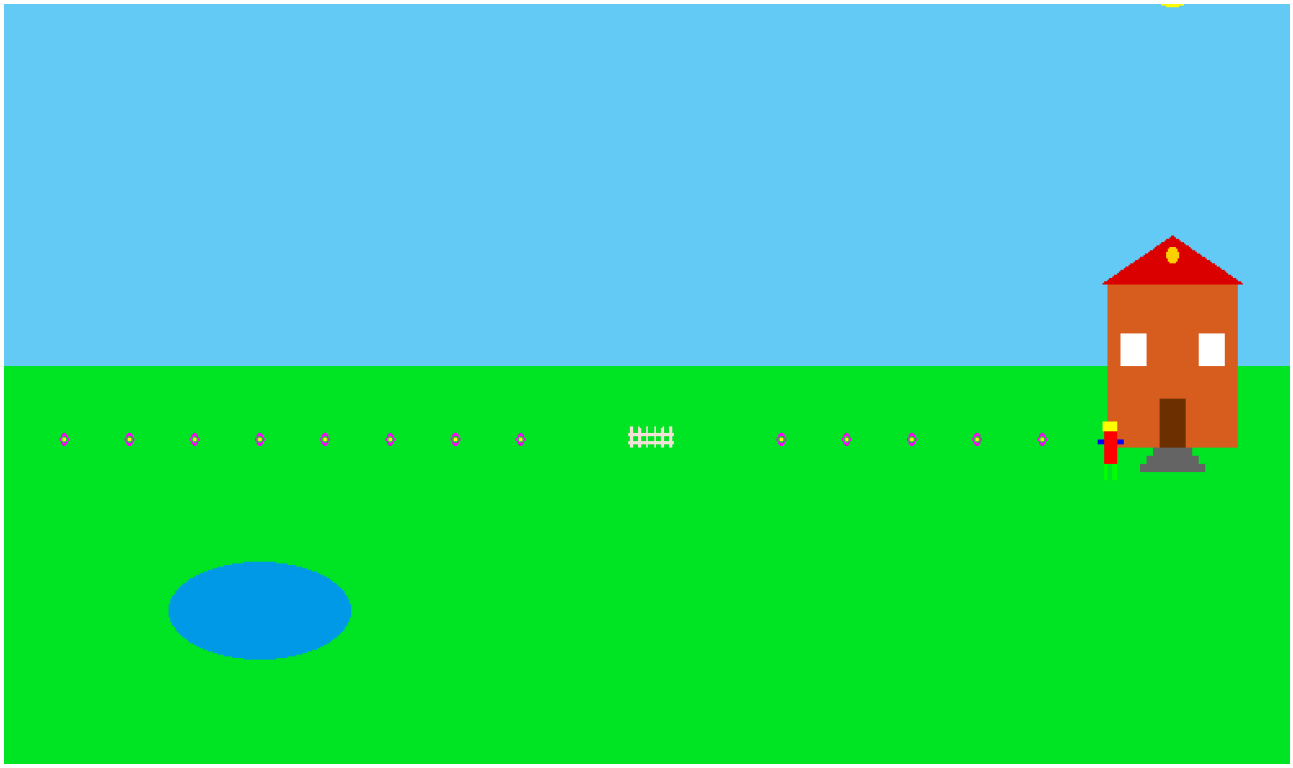


Рисунок 3.5.3 – Досягнення школи

3.6 Висновки до розділу 3

У третьому розділі було реалізовано приклад практичного втілення математичних методів інтерполяції у сфері комп’ютерної графіки. Створена за допомогою бібліотеки OpenGL анімована сцена демонструє, як засоби інтерполяції дозволяють досягнути високої реалістичності та плавності руху віртуального персонажа у динамічному середовищі.

Застосування поліноміальної інтерполяції, зокрема методу Лагранжа, дозволило точно моделювати траєкторію руху, забезпечуючи логічну послідовність зміни координат у часі. Такий підхід не лише покращує візуальне сприйняття, а й відкриває широкі можливості для застосування в інженерному моделюванні, візуалізації даних, створенні інтерактивних освітніх програм та симуляторів.

Отже, інтерполяція у поєднанні з графічними технологіями стає не лише інструментом розрахунків, а й потужним засобом творчої візуалізації складних математичних процесів у програмному середовищі.

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання дипломної роботи було здійснено комплексне дослідження ключових підходів до інтерполяції, розглянуто їх математичну основу, історичний розвиток, а також проаналізовано можливості практичного впровадження в обчислювальних процесах. Інтерполяція відіграє визначальну роль у чисельному аналізі, дозволяючи відновлювати значення функцій на основі обмеженого набору дискретних точок, що є надзвичайно актуальним у наукових обчисленнях, комп'ютерній графіці, моделюванні фізичних процесів тощо.

У теоретичній частині особливу увагу приділено поліноміальним методам інтерполяції, зокрема — формулі Лагранжа, методу розділених різниць Ньютона та кубічним сплайнам. У результаті порівняльного аналізу виявлено, що методи Лагранжа і Ньютона є доцільними для задач з обмеженою кількістю вузлів, тоді як кубічні сплайни демонструють суттєві переваги при великій кількості даних, забезпечуючи плавність та стабільність апроксимації без різких осциляцій.

Особливої цінності дослідження набуває завдяки реалізації програмного модуля, що візуалізує застосування інтерполяційних методів. У рамках практичної частини було розроблено інтерактивну графічну програму з використанням бібліотеки OpenGL, де персонаж рухається до заданої цілі (школи), обходячи перешкоду. Анімація цього руху реалізована із застосуванням принципів лінійної інтерполяції координат, що забезпечує плавну зміну положення об'єкта у просторі. Такий підхід дозволив продемонструвати ефективність інтерполяції не лише як абстрактного чисельного методу, а як практичного інструмента для створення динамічних і візуально привабливих програм.

Отже, проведене дослідження підтвердило, що для завдань, які потребують високої точності та плавності відновлення функцій між дискретними значеннями, кубічні сплайни залишаються найкращим вибором. Водночас, лінійна інтерполяція має своє місце в реальному програмуванні, зокрема в задачах комп’ютерної анімації, і може слугувати ефективним і простим інструментом для побудови графічних сценаріїв. Поєднання теоретичного аналізу та програмної реалізації дозволило всебічно розкрити потенціал інтерполяційних методів у сучасному застосуванні.

ДОДАТОК

```
#include <GLUT/glut.h>
#include <iostream>

float manX = -4.0f;// Початкова позиція людини по осі X
float manY = -0.5f;// Початкова позиція людини по осі Y
float schoolX = 4.0f;// Позиція школи по осі X
bool isMoving = false;// Статус, чи рухається людина
bool isJumping = false;// Статус, чи людина стрибає
bool hasReachedSchool = false;// Статус, чи людина досягла школи
float jumpHeight = 0.0f;// Висота стрибка
float jumpSpeed = 0.1f;// Швидкість стрибка
float obstacleX = 0.0f;// Позиція перешкоди по осі X
float obstacleWidth = 0.5f;// Ширина перешкоди
bool hasAvoidedObstacle = false;// Статус, чи людина уникнула перешкоди
int avoidStage = 0;// Стадія уникання перешкоди
float avoidStartX = 0.0f;// Початкова X позиція для уникання
float avoidStartY = -0.5f;// Початкова Y позиція для уникання


// Функція для малювання сонця
void drawSun() {
    glColor3f(1.0f, 1.0f, 0.0f);// Жовтий колір для сонця
    float radius = 0.3f;
    float x = 4.0f, y = 2.5f;
    glBegin(GL_TRIANGLE_FAN);// Малюємо сонце як багато граней
    glVertex2f(x, y);
    for (int i = 0; i <= 100; i++) {
        float angle = 2.0f * 3.14159f * i / 100;
        glVertex2f(x + cos(angle) * radius, y + sin(angle) * radius);
    }
    glEnd();
}


// Функція для малювання квітки
void drawFlower(float x, float y) {
    glColor3f(1.0f, 0.0f, 1.0f); // пелюстки
    for (int i = 0; i < 8; ++i) {
        float angle = i * 3.14159f / 4.0f;
        float dx = cos(angle) * 0.03f;
        float dy = sin(angle) * 0.03f;
        glBegin(GL_POLYGON);// Малюємо пелюстки
        for (int j = 0; j < 20; ++j) {
            float a = j * 2 * 3.14159f / 20;
            glVertex2f(x + dx + cos(a) * 0.01f, y + dy + sin(a) *
0.01f);
        }
        glEnd();
    }

    glColor3f(1.0f, 1.0f, 0.0f); // серцевина
    glBegin(GL_POLYGON);
    for (int j = 0; j < 20; ++j) {
        float a = j * 2 * 3.14159f / 20;
        glVertex2f(x + cos(a) * 0.01f, y + sin(a) * 0.01f);
    }
```

```
    glEnd();
}


// Функція для малювання озера
void drawLake() {
    glColor3f(0.2f, 0.6f, 0.9f);// Блакитний колір для озера
    glBegin(GL_POLYGON);
    for (int i = 0; i <= 100; i++) {
        float angle = 2.0f * 3.14159f * i / 100;
        float radiusX = 0.7f;
        float radiusY = 0.3f;
        glVertex2f(-3.0f + cos(angle) * radiusX, -1.5f + sin(angle) *
radiusY);
    }
    glEnd();
}


// Функція для малювання людини
void drawMan() {
    glColor3f(1.0f, 0.0f, 0.0f);// Червоний колір для тіла людини
    glBegin(GL_QUADS);// Малюємо прямокутник для тіла
    glVertex2f(manX - 0.05f, manY - 0.1f + jumpHeight);
    glVertex2f(manX + 0.05f, manY - 0.1f + jumpHeight);
    glVertex2f(manX + 0.05f, manY + 0.1f + jumpHeight);
    glVertex2f(manX - 0.05f, manY + 0.1f + jumpHeight);
    glEnd();
    // Малюємо голову
    glColor3f(1.0f, 1.0f, 0.0f);// Жовтий колір для голови
    glBegin(GL_QUADS);
    glVertex2f(manX - 0.06f, manY + 0.1f + jumpHeight);
    glVertex2f(manX + 0.06f, manY + 0.1f + jumpHeight);
    glVertex2f(manX + 0.06f, manY + 0.16f + jumpHeight);
    glVertex2f(manX - 0.06f, manY + 0.16f + jumpHeight);
    glEnd();
    // Малюємо ноги
    glColor3f(0.0f, 0.0f, 1.0f);
    glBegin(GL_QUADS);
    glVertex2f(manX - 0.1f, manY + 0.05f + jumpHeight);
    glVertex2f(manX - 0.05f, manY + 0.05f + jumpHeight);
    glVertex2f(manX - 0.05f, manY + 0.02f + jumpHeight);
    glVertex2f(manX - 0.1f, manY + 0.02f + jumpHeight);

    glVertex2f(manX + 0.05f, manY + 0.05f + jumpHeight);
    glVertex2f(manX + 0.1f, manY + 0.05f + jumpHeight);
    glVertex2f(manX + 0.1f, manY + 0.02f + jumpHeight);
    glVertex2f(manX + 0.05f, manY + 0.02f + jumpHeight);
    glEnd();
    // Малюємо взуття
    glColor3f(0.0f, 1.0f, 0.0f);// Зелений колір для взуття
    glBegin(GL_QUADS);
    glVertex2f(manX - 0.05f, manY - 0.1f + jumpHeight);
    glVertex2f(manX - 0.05f, manY - 0.2f + jumpHeight);
    glVertex2f(manX - 0.02f, manY - 0.2f + jumpHeight);
    glVertex2f(manX - 0.02f, manY - 0.1f + jumpHeight);

    glVertex2f(manX + 0.02f, manY - 0.1f + jumpHeight);
```

```
    glVertex2f(manX + 0.02f, manY - 0.2f + jumpHeight);
    glVertex2f(manX + 0.05f, manY - 0.2f + jumpHeight);
    glVertex2f(manX + 0.05f, manY - 0.1f + jumpHeight);
    glEnd();
}


// Функція для малювання школи
void drawSchool() {
    // Малюємо будинок школи
    glColor3f(0.8f, 0.4f, 0.2f);
    glBegin(GL_QUADS);
    glVertex2f(schoolX - 0.5f, -0.5f);
    glVertex2f(schoolX + 0.5f, -0.5f);
    glVertex2f(schoolX + 0.5f, 0.5f);
    glVertex2f(schoolX - 0.5f, 0.5f);
    glEnd();
    // Малюємо дах
    glColor3f(0.8f, 0.0f, 0.0f);
    glBegin(GL_TRIANGLES);
    glVertex2f(schoolX - 0.55f, 0.5f);
    glVertex2f(schoolX + 0.55f, 0.5f);
    glVertex2f(schoolX, 0.8f);
    glEnd();
    // Малюємо двері
    glColor3f(0.4f, 0.2f, 0.0f);
    glBegin(GL_QUADS);
    glVertex2f(schoolX - 0.1f, -0.5f);
    glVertex2f(schoolX + 0.1f, -0.5f);
    glVertex2f(schoolX + 0.1f, -0.2f);
    glVertex2f(schoolX - 0.1f, -0.2f);
    glEnd();
    // Малюємо вікна
    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);
    glVertex2f(schoolX - 0.4f, 0.0f);
    glVertex2f(schoolX - 0.2f, 0.0f);
    glVertex2f(schoolX - 0.2f, 0.2f);
    glVertex2f(schoolX - 0.4f, 0.2f);

    glVertex2f(schoolX + 0.2f, 0.0f);
    glVertex2f(schoolX + 0.4f, 0.0f);
    glVertex2f(schoolX + 0.4f, 0.2f);
    glVertex2f(schoolX + 0.2f, 0.2f);
    glEnd();
    // Дзвін
    glColor3f(1.0f, 0.84f, 0.0f);
    glBegin(GL_POLYGON);
    for (int i = 0; i < 20; ++i) {
        float angle = i * 2.0f * 3.14159f / 20;
        glVertex2f(schoolX + 0.0f + cos(angle) * 0.05f, 0.68f +
sin(angle) * 0.05f);
    }
    glEnd();

    // Сходи
    for (int i = 0; i < 3; ++i) {
        glColor3f(0.4f, 0.4f, 0.4f);
```

```
        glBegin(GL_QUADS);
        glVertex2f(schoolX - 0.15f - i * 0.05f, -0.5f - i * 0.05f);
        glVertex2f(schoolX + 0.15f + i * 0.05f, -0.5f - i * 0.05f);
        glVertex2f(schoolX + 0.15f + i * 0.05f, -0.5f - (i + 1) *
0.05f);
        glVertex2f(schoolX - 0.15f - i * 0.05f, -0.5f - (i + 1) *
0.05f);
        glEnd();
    }

}


// Функція для малювання перешкоди
void drawObstacle(float x) {
    // Світло-дерев'яний колір
    glColor3f(0.95f, 0.9f, 0.85f);

    // Вертикальні стовпчики
    for (float i = -0.15f; i <= 0.15f; i += 0.06f) {
        float baseX = x + i;

        // Ніжка
        glBegin(GL_QUADS);
        glVertex2f(baseX - 0.01f, -0.5f);
        glVertex2f(baseX + 0.01f, -0.5f);
        glVertex2f(baseX + 0.01f, -0.38f);
        glVertex2f(baseX - 0.01f, -0.38f);
        glEnd();

        // Округла верхівка
        glBegin(GL_POLYGON);
        for (int j = 0; j <= 180; j += 10) {
            float angle = j * 3.14159f / 180.0f;
            float radius = 0.01f;
            glVertex2f(baseX + cos(angle) * radius, -0.38f + sin(angle)
* radius);
        }
        glEnd();
    }

    // Горизонтальні перекладини
    glBegin(GL_QUADS);
    glVertex2f(x - 0.17f, -0.43f);
    glVertex2f(x + 0.17f, -0.43f);
    glVertex2f(x + 0.17f, -0.41f);
    glVertex2f(x - 0.17f, -0.41f);
    glEnd();

    glBegin(GL_QUADS);
    glVertex2f(x - 0.17f, -0.48f);
    glVertex2f(x + 0.17f, -0.48f);
    glVertex2f(x + 0.17f, -0.46f);
    glVertex2f(x - 0.17f, -0.46f);
    glEnd();
}
```

```
void renderBitmapString(float x, float y, void* font, const char* string)
{
    glRasterPos2f(x, y);
    while (*string) {
        glutBitmapCharacter(font, *string);
        ++string;
    }
}

bool isCollidingWithObstacle() {
    // Перевірка на зіткнення:
        // 1. Людина (з урахуванням її розміру) знаходиться в межах
перешкоди по осі X
        // 2. Людина на землі (перевірка висоти стрибка)
    return (manX + 0.05f > obstacleX - obstacleWidth &&
            manX - 0.05f < obstacleX + obstacleWidth &&
            jumpHeight < 0.3f);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);


    // Небо
    glBegin(GL_QUADS);
    glColor3f(0.5f, 0.8f, 0.95f);
    glVertex2f(-5.0f, 0.0f);
    glVertex2f(5.0f, 0.0f);
    glVertex2f(5.0f, 3.0f);
    glVertex2f(-5.0f, 3.0f);
    glEnd();

    // Трава
    glBegin(GL_QUADS);
    glColor3f(0.3f, 0.9f, 0.3f);
    glVertex2f(-5.0f, -3.0f);
    glVertex2f(5.0f, -3.0f);
    glVertex2f(5.0f, 0.0f);
    glVertex2f(-5.0f, 0.0f);
    glEnd();
    // Малюємо сонце, квітки, озеро, людину, школу та перешкоду
    drawSun();
    drawLake();
    drawSchool();
    drawObstacle(obstacleX);
    drawMan();
    // Квіти ліворуч і праворуч
    for (float x = -4.5f; x < 4.5f; x += 0.5f) {
        if (fabs(x - obstacleX) > 0.6f && fabs(x - schoolX) > 0.8f) {
            drawFlower(x, -0.45f);
        }
    }


    if (hasReachedSchool) {
        glColor3f(0.0f, 0.0f, 0.0f);
```

```
        renderBitmapString(-0.3f, 2.5f, GLUT_BITMAP_HELVETICA_18,
"School");
    }

    glutSwapBuffers();
}

void update(int value) {
    // Якщо людина рухається до школи
    if (isMoving) {
        // Якщо людина ще не досягла школи, рухаємось
        if (manX < schoolX) {
            // Якщо на шляху перешкода, і людина не уникала її раніше
            if (avoidStage == 0 && isCollidingWithObstacle() &&
!hasAvoidedObstacle) {
                // Починаємо процес уникання перешкоди
                avoidStage = 1;
                avoidStartX = manX;
                avoidStartY = manY;
            }
            // Якщо етап уникання 1, людина починає підніматися
            if (avoidStage == 1) {
                manY += 0.02f;
                // Піднімаємо людину
                                // Якщо досягли певної висоти,
переходимо до етапу 2
                if (manY >= 0.3f) {
                    avoidStage = 2;
                }
            }// Етап уникання 2, рухаємо людину вбік, щоб обійти
перешкоду
            else if (avoidStage == 2) {
                manX += 0.02f;// Переміщаємо людину вбік
                // Якщо обійшли перешкоду, переходимо до етапу 3
                if (manX > obstacleX + obstacleWidth) {
                    avoidStage = 3;
                    hasAvoidedObstacle = true; //  більше не обробляти
паркан
                }
            }
            // Етап уникання 3, опускаємо людину назад
            else if (avoidStage == 3) {
                manY -= 0.02f;// Опускаємо людину
                // Якщо досягли початкової висоти, повертаємось до етапу
0
                if (manY <= avoidStartY) {
                    manY = avoidStartY;// Встановлюємо людину назад на
початкову висоту
                    avoidStage = 0;
                }
            }
            // Якщо уникнення не потрібно, просто рухаємось вперед
            else {
                manX += 0.02f;
            }
            // Якщо людина досягла школи, зупиняємо рух
            if (manX >= schoolX - 0.5f) {
                isMoving = false;// Зупиняємо рух
```

```
                hasReachedSchool = true;// Відмічаємо, що людина досягла школи
            }
        }
    }

    // Якщо людина стрибає, підвищуємо висоту стрибка
    if (isJumping) {
        jumpHeight += jumpSpeed;// Підвищуємо висоту стрибка
        // Якщо досягли максимальної висоти, припиняємо стрибок
        if (jumpHeight >= 0.5f) {
            isJumping = false;// Закінчуємо стрибок
        }
    }
    // Якщо стрибок завершено, знижуємо висоту
    else if (jumpHeight > 0.0f) {
        jumpHeight -= jumpSpeed;// Зменшуємо висоту стрибка
    }

    glutPostRedisplay();
    // Викликаємо цю функцію кожні 16 мілісекунд (близько 60 FPS)
    glutTimerFunc(16, update, 0);
}

void keyboard(unsigned char key, int x, int y) {
    if (key == ' ') {
        isMoving = true;
    }
    if (key == 'j' && !isJumping) {
        isJumping = true;
    }
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Man Going to School");

    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-5.0, 5.0, -3.0, 3.0);

    glutDisplayFunc(display);
    glutTimerFunc(0, update, 0);
    glutKeyboardFunc(keyboard);

    glutMainLoop();
    return 0;
}
```

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

J Szabados, Peter Vertesi INTERPOLATION OF FUNCTIONS

Vladimir Rovenski Interpolation of Functions

Interpolation and Curve Fitting

OpenGL Documentation: https://www.opengl.org/documentation/

Learn OpenGL .Beginner's Guide to 3D Rendering and Game Development with OpenGL and C++