Прикарпатський національний університет імені Василя Стефаника
Кафедра алгебри та геометрії

БАКАЛАВРСЬКА РОБОТА

на тему: ЗАСТОСУВАННЯ КРИВИХ БЕЗЬЄ У КОМП'ЮТЕРНІЙ ГРАФІЦІ

Виконала: студентка IV курсу,
групи М–41, спеціальності "Математика"

Буй Антоніна Ігорівна

Керівник: к.ф.-м.н. Копорх К.М.

Рецензент: к.ф.-м.н., доц. Гаврилків В.М.

Національна шкала: _____
Університетська шкала: _____
Оцінка ECTS: _____

м. Івано-Франківськ – 2025 рік

ЗМІСТ

ВСТУП

Криві Безьє займають важливе місце в сучасній математиці та комп'ютерній графіці завдяки своїй здатності ефективно описувати складні форми і поверхні. Цей математичний інструмент, вперше запропонований в середині XX століття, широко використовується в таких галузях, як анімація, CAD-системи, та векторна графіка. Криві Безьє дозволяють створювати гладкі криві, які легко контролювати за допомогою контрольних точок, що надає великі можливості для дизайну та моделювання.

Актуальність дослідження кривих Безьє обумовлена їх численними застосуваннями інженерії, комп'ютерних науках, геймдейві та інших сферах, де існує необхідність моделювання реальних об'єктів. Проте, не зважаючи на їх поширеність, недостатньо досліджені математичні властивості, функціональні характеристики кривих Безьє та їх практичне застосування, що викликає інтерес до більш глибокого аналізу даної теми.

Об'єктом дослідження є криві Безьє як математичний інструмент моделювання графічних об'єктів. **Предмет дослідження –** методи, алгоритми та особливості застосування кривих Безьє в комп'ютерній графіці.

Мета курсової роботи полягає в систематизації знань про криві Безьє, розкритті теоретичних основ поняття та математичних властивостей, вивченні їх функціонального призначення в різних сферах та практичній реалізації з використанням найактуальніших методик програмного моделювання.

Для досягнення поставленої мети сформовано такі **завдання**:

- Проаналізувати історію виникнення та розвитку дослідження кривих Безьє.
- Визначити теоретичний базис поняття кривих Безьє, їх сутність та термінологію.
- Дослідити математичне підґрунтя та геометричні властивості кривих Безьє.
- Визначити шляхи застосування кривих Безьє відповідно до їх функціонального призначення.
- Дослідити алгоритми штучного інтелекту щодо апроксимації кривих Безьє.

- Визначити особливості практичного моделювання кривих Безьє на базі Open GL.

Завдяки постійному розширенню використання кривих Безьє, даний напрям досліджень знаходиться в стані активної розробки в Україні. У цьому контексті важливим є дослідження проведені такими науковцями як Молоденков К. [9], Холодняк Ю., Гавриленко Є., Зінов'єва О. [12], Надопта Т. [10], Похваленна О. [11], Ванін В., Вірченко Г., Яблонський П. [4], Сидоренко Ю., Залевська О. [1], Бідніченко О. [3], Євсєєв О. [6], Демків І. [5], Сорокін М. та Логвиненко Н. В. [2], чиї праці стали основою для написання даної роботи.

Основними теоретичними **методами** дослідження є аналіз наукової літератури та вивчення практичних прикладів використання кривих Безьє в сучасних технологіях. Практична складова реалізована за допомогою емпіричного методу комп'ютерного моделювання.

Практичне значення одержаних результатів полягає у можливості використання матеріалів дослідження для вдосконалення методів математичного моделювання та комп'ютерного дизайну.

Робота складається зі вступу та двох основних розділів, висновків щодо дослідження, а також списку використаних джерел. Обсяг роботи - 58 сторінок, які включають 27 рисунків, чотири додатки та одну таблицю, список використаних джерел містить 35 найменувань.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ЗАСАДИ ІСТОРИЧНІ ПЕРЕДУМОВИ ВИКОРИСТАННЯ КРИВИХ БЕЗЬЄ

1.1 Історія виникнення та розвитку дослідження кривих Безьє

Важливим питанням у сучасній науковій спільноті залишається визначення історичних передумов розвитку поняття про криві Безьє, адже потреба у ньому з'явилася ще задовго до виникнення тогочасних комп'ютерних технологій. Актуальним науковці вважають період, коли моделювання об'єктів здійснювалось за допомогою сплайна – гнучкого інструмента з пластику чи дерева, який фіксували на креслярській дошці свинцевими тягарцями, що називалися «качками». Пересування цих вантажів змінювало форму сплайна, дозволяючи налаштовувати його контури [18].

Такий підхід був недосконалим, інженери потребували точного розміщення «качок» і дорогого обладнання, а його використання потребувало значного простору. Математично він також не був оптимльним, адже не існувало розробленого рішення для точного аналізу кривих, побудованих таким способом. Поява комп'ютерів лише загострила потребу у розробках цього напрямку, адже криві легко малювати від руки, але відтворення плавних кривих на комп'ютері вимагало нового підходу[32].

Для візуалізації кривих на комп'ютері потрібна математична функція, яка вказує як їх формувати. Перший крок у цьому напрямку зробив Сергій Бернштейн у 1912 році, розробивши концепцію плавних поліномів — виразів, що складаються з кількох алгебраїчних доданків. Однак через відсутність обчислювальної техніки його ідея залишалася нереалізованою аж до 1960-х років [22, 32].

В середині XX століття темпи технічного прогресування комп'ютерної графіки продовжували значно відставати відносно споживацьких потреб у цій галузі. Телевізійні компанії прагнули отримати прості та зручні інструменти для роботи зі сканованими зображеннями, які можна було б змінювати безпосередньо на екрані, зацікавлюючи глядачів різноманітними спецефектами. Однак,

обчислювальні ресурси наявної техніки досі не дозволяли виконувати ці завдання в оптимальні терміни [7].

Подібні проблеми виникали і в інших галузях, саме в цей період інженери стикнулися з потребою точного моделювання складних форм, що дозволило б оптимізувати дизайн кузовів автомобілів. З цією метою французький автоконцерн «Renault» звернувся до математика та інженера П’єра Безьє з проханням розробити універсальний та водночас простий метод опису складних плоских форм, необхідних для автоматизованої обробки листового металу. У 1962 році він створив систему кривих, яка виявилася настільки ефективною, що згодом стала основою не лише для графічних, а й для багатьох інших прикладних програм[7].

Не менш важливим фактом є те, що трохи раніше цієї події криві Безьє були незалежно розроблені Полем де Кастелжо для компанії «Citroën» та також використовувалися для проектування кузовів автомобілів. Суттєвою відмінністю між подальшими просуваннями цих розробок стало те, що дослідження де Кастелжо не публікувалися та ховалися компанією як виробнича таємниця аж до кінця 1960-х років, на відміну від кривих Безьє, які були опубліковані в 1962 році самим П’єром Безьє. Натомість, ім’ям де Кастелжо було названо розроблений ним рекурсивний метод обчислення та побудови кривих – алгоритм де Кастелжо [6].

За допомогою однієї простої математичної функції П’єр Безьє зробив революцію в цифровому дизайні. Його інструмент Computer-Aided Geometric Design (CAGD) називався UNISURF, що дозволяв дизайнерам малювати гладкі, точні криві на екрані комп’ютера [linearity]. Після публікації робіт Безьє його методики знайшли застосування у міжнародних компаніях, таких як Boeing і General Motors. Це стимулювало наукову співпрацю між інженерами та математиками, що дало поштовх до подальшого вдосконалення математичних моделей [7,18].

Успіх цієї розробки в автомобільній промисловості привернув увагу інженерів і розробників комп'ютерної графіки, які шукали ефективні інструменти для створення 3D-об'єктів. З розвитком технологій комп'ютерна графіка швидко

почала інтегрувати криві Безьє як основу для створення більш складних геометричних об'єктів, які стали основою для анімації та дизайну.

Наукова спільнота також активно зацікавилася властивостями кривих Безьє, що призвело до появи численних праць, присвячених їхньому математичному аналізу та передовим обчислювальним методам. Розробка алгоритмів для ефективного обчислення та використання цих кривих, зокрема алгоритму де Кастелжо, значно розширила їхній прикладний потенціалі зробила криві Безьє невід'ємною частиною сучасних систем комп'ютерного дизайну та графічного моделювання [7,18].

1.2 Поняття про криві Безьє: сутність та термінологія

Для вдалого моделювання об’єктів на основі побудови кривих Безьє, важливим є дослідження їх базових характеристик та опорних визначень. Саме розуміння цих елементів дозволяє коректно будувати криві різних ступенів, контролювати їхню форму та плавність, а також точно прогнозувати їх «поведінку» при зміні параметрів.

За своєю сутністю крива Безьє є параметричною кривою, яку можна задати виразом, в якому позиція точки на кривій визначається змінним параметром t. Параметрична крива генерується за допомогою лінійної інтерполяції. Основна ідея полягає в тому, що форма самої кривої контролюється кількома ключовими точками, які називаються *контрольними точками*. Крива починається з координат першої контрольної точки та закінчується на координатах останньої, а всі проміжні точки своїм розміщенням впливають на її вигини. Їх взаєморозміщення також впливає на загальний контур кривої, її напрямок та плавність [12].

Контрольні точки з’єднуються лініями, утворюючи багатокутник, який називається *контрольним полігоном*. Контрольний полігон не є частиною самої кривої, але він показує загальний напрямок і вигин кривої Безьє (рис.1).

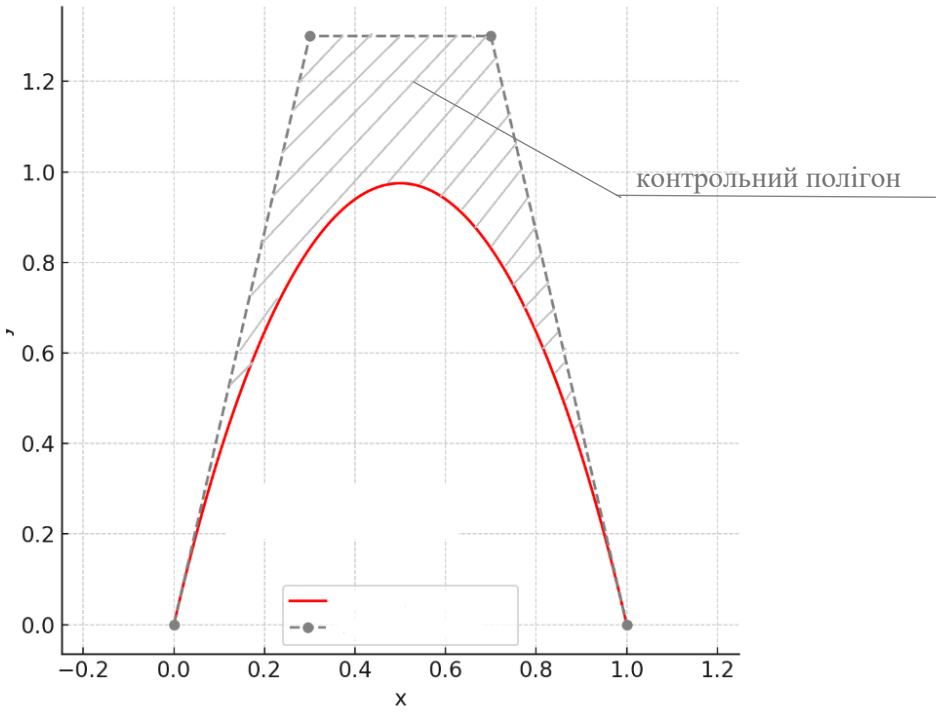

Рисунок 1.1 Межі контрольного полігону t в кубічній кривій Безьє.

Параметр t – це змінна, яка змінюється в зазначеній області і задає точку на кривій для кожного значення. Початок кривої відповідає значенню параметра 0, а кінець – значенню 1. У проміжних значеннях параметра t крива «плавно переміщується» від однієї контрольної точки до іншої (рис. 2).

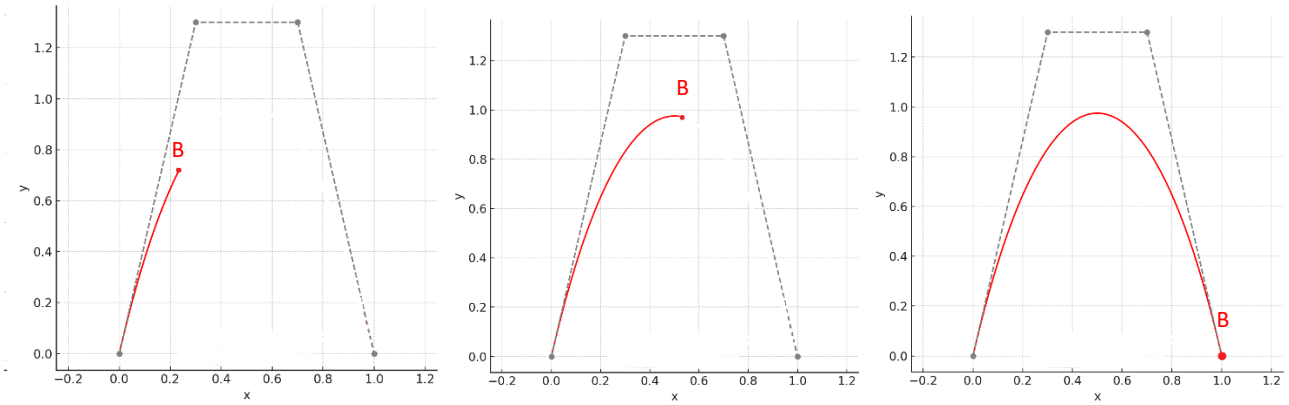

Рисунок 1.2 Траєкторія переміщення точки B через змінний параметр t в кубічній кривій Безьє.

Важливою характеристикою кривих Безьє є їх *ступеневість*, яка визначається кількістю її контрольних точок (крива n-го ступеня потребує n+1

таких точок). Саме ступінь кривої впливає на її вигин та можливості формування складних форм. За цією ознакою виділяють чотири групи кривих Безьє з унікальними індивідуальними особливостями:

1. *Лінійні (першого ступеня),* які утворюються двома контрольними точками та мають форму прямої лінії. Такі криві є найпростішими і не мають вигинів – вони просто з'єднують дві контрольні точки.
2. Основою побудови *квадратичних (другого ступеня)* кривих Безьє слугує сукупність трьох контрольних точок. Такі криві вже мають плавний вигин і можуть утворювати форму, подібну до параболи. Ця властивість часто використовується для створення простих дуг і вигинів у графічному дизайні.
3. *Кубічні криві (третього ступеня)* програмуються за допомогою чотирьох контрольних точок та дозволяють створювати більш складні вигини (рис.1.3(а)).

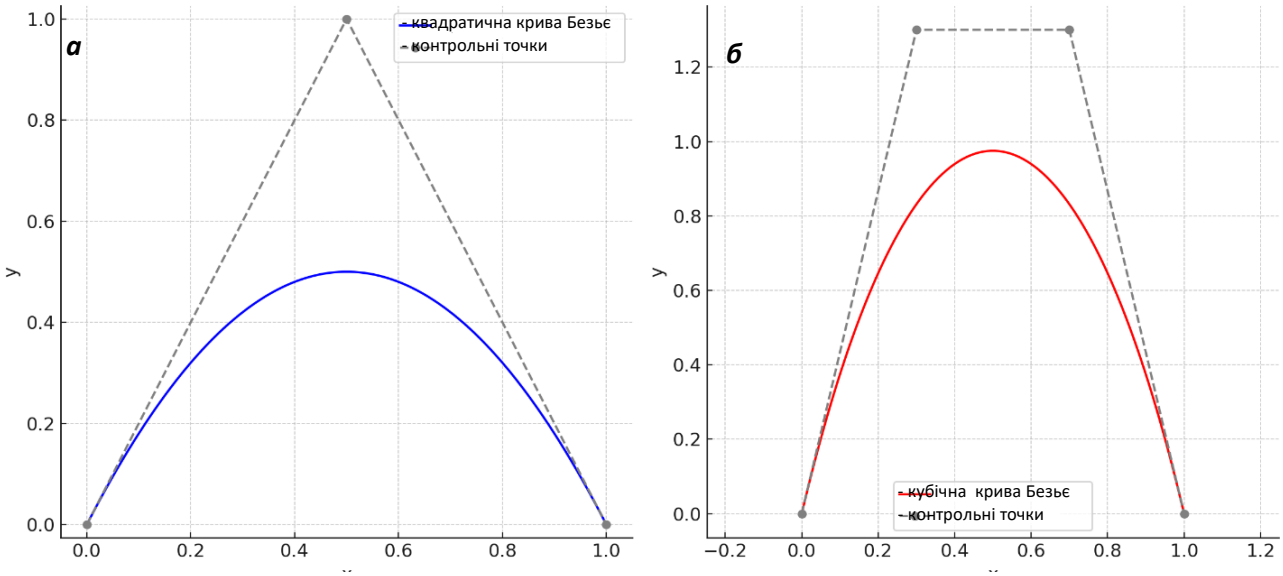

Рисунок 1.3 Різновиди кривих Безьє за кількістю контрольних точок: а) квадратична; б) кубічна.

Завдяки додатковій контрольній точці, ці криві забезпечують значно більшу гнучкість та контроль над формою. Цей тип вважається одним з найпоширеніших, адже використовується у векторній графіці та багатьох графічних програмах.

4. Криві Безьє *четвертого ступеня та вище* мають п'ять або більше контрольних точок і ще складніші вигини (рис.1.3(б)). Хоча вони дозволяють

отримати дуже гнучкі форми, такі криві рідко використовують на практиці через складність обчислень і більшу схильність до коливань нерівностей.

Чим вищий ступінь кривої, тим точніше можна контролювати її форму, однак складність обчислень і велика ймовірність виникнення небажаних коливань роблять високоступеневі криві менш поширеними [9].

Більш складними та функціональними є поверхні Безьє. Вони будуються на основі кривих Безьє, з'єднаних у двох напрямах, що дозволяє розтягувати й формувати їх у просторі з високою точністю. Поверхня Безьє формується із сітки контрольних точок, які розташовані у вигляді прямокутної матриці. В кожному рядку та стовпці цієї матриці можна уявити криві Безьє, що проходять через певні контрольні точки [10].

Щоб отримати поверхню, обирають два напрями – наприклад U і V. Спершу створюють криві Безьє у напрямку U, а потім для кожного значення параметра V будують криві, пов'язані з контрольними точками в напрямку V. У процесі такої побудови відбувається інтерполяція. У напрямку U точка поверхні інтерполюється по контрольних точках у цьому напрямку, а потім, змінюючи V, точка на поверхні інтерполюється вздовж кривих у напрямку V. Підсумкова поверхня – результат обчислення точок на кожній з цих кривих. Отже, поверхня Безьє фактично є двовимірним розширенням кривої Безьє, яке надає ще ширші можливості для практичного використання [11].

Висновки до розділу 1

Дослідивши теоретичні засади та історичні передумови використання кривих Безьє, можна зробити декілька проміжних висновків:

- Криві Безьє є параметричними, що дозволяє контролювати їх форму за допомогою контрольних точок.
- Вплив кожної контрольної точки локальний, що дає можливість змінювати форму кривої без впливу на всю її геометрію.
- Гладкість кривих визначається поліноміальним ступенем, що дозволяє створювати безперервні та плавні форми.
- Розробка концепції кривих розпочалася ще в 1912 році завдяки працям Сергія Бернштейна. Однак справжній прорив відбувся в 1960-х роках завдяки незалежним дослідженням П’єра Безьє та Поля де Кастелжо, які створили алгоритми побудови кривих, зокрема алгоритм де Кастелжо.
- Завдяки розробці алгоритмів, таких як алгоритм де Кастелжо, та створенню систем, як-от UNISURF, криві Безьє стали ключовими інструментами у комп’ютерному дизайні.
- Криві Безьє дали поштовх до подальших досліджень у галузі комп’ютерної графіки та чисельних методів. Вони стали основою для розробки складніших геометричних об’єктів, таких як поверхні Безьє, що мають ширший спектр застосувань у тривимірному моделюванні.

РОЗДІЛ 2.
АЛГЕБРАЇЧНІ ТА ФУНКЦІОНАЛЬНІ ВЛАСТИВОСТІ КРИВИХ БЕЗЬЄ

2.1 Математичне підґрунтя та геометричні властивості кривих Безьє

Крива Безьє – це параметрична крива, яка використовує поліномне рівняння Бернштейна як основу. У загальному формулюванні воно визначає криву на основі n +1 контрольних точок. Кожна така точка на кривій обчислюється як:

$$B(t) = \sum_{i=0}^{n} P_i \, b_i^n(t) \,, \tag{1}$$

де коефіцієнти P_i є власне *контрольними точками,* t – параметром на інтервалі [0,1], який визначає положення цих точок на кривій, а $b_{i,n}$ (t) – це поліноми Бернштейна, які визначають «вагу» для кожної точки на кривій залежно від параметру t. Сам поліном Бернштейна ступеня n для індексу i задається так [35]:

$$b_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}. \tag{2}$$

Даний вираз і є базисною функцією Безьє. В ньому $\binom{n}{i}$ є біномальним коефіцієнтом, який обчислюється за формулою [5]:

$$\binom{n}{i} = \frac{n!}{i!\,(n-i)!} \ . \tag{3}$$

Простіше кажучи, цей коефіцієнт обчислюється як кількість способів обрати i елементів із n без урахування порядку. Наприклад, для кубічної кривої (коли n = 3) маємо чотири поліноми Бернштейна для чотирьох контрольних точок (P_0, P_1, P_2, P_3):

$$\begin{aligned}
&1.\, b_0^3(t) = (1-t)^3, \\
&2.\, b_1^3(t) = 3(1-t)^2 \cdot t, \\
&3.\, b_2^3(t) = 3(1-t) \cdot t^2, \\
&4.\, b_3^3(t) = t^3.
\end{aligned} \tag{4}$$

Кожен з цих поліномів приймає значення в межах від 0 до 1, в залежності від параметра t. Це і визначає функціональну характеристику точки в досліджуваній кривій Безьє.

Провівши розрахунки за виразом (2) для кривої *n*-го ступеня (з довільною кількістю контрольних точок) та обравши для параметру t його крайні допустимі значення (0 та 1), можна зробити висновок, що при t = 0 тільки перший базисний поліном $b_0^n(t)$ дорівнює 1:

$$b_0^n(0) = \frac{n! \cdot t^0 (1-0)^{n-0}}{0! \cdot n!} = 1\,. \qquad (5)$$

Для всіх інших базисних функцій $b_i^n(t)$, де i > 0, вираз можна записати так:

$$b_i^n(0) = \frac{n! \cdot 0^i \cdot (1-0)^{n-i}}{i! \cdot (n-1)!}\,. \qquad (6)$$

Оскільки $0^i = 0$ для всіх i > 0, то:

$$b_i^n(0) = 0 \qquad (7)$$

Це доводить, що *крива починається саме в першій контрольній точці (вершині) – P_0.*

Аналогічно проведемо розрахунки для для t = 1. З них випливає, що тільки $b_{n,n}$ дорівнюватиме 1 [3]:

$$b_n^n(1) = \frac{n!\, t^n (1-1)^{n-n}}{n! \cdot (n-n)!} = 1\,. \qquad (8)$$

Для всіх інших базисних поліномів $b_{i,n}(1)$ (де i < n) вираз набуває вигляду:

$$b_i^n(1) = \frac{n! \cdot 1^i \cdot (1-1)^{n-i}}{i! \cdot (n-i)!}\,. \qquad (9)$$

Оскільки $(1 - 1)^{n-i} = 0$ для всіх i < n, отримаємо:

$$b_i^n\,(1) = 0 \qquad (10)$$

Розрахунок демонструє, що *крива завжди завершується в останній контрольній точці P_n* .Таким чином, крива плавно «перетікає» від початкової контрольної точки P_0 через проміжні точки до P_n (рис.1.2), при чому контрольні точки, які є ближчими до поточного значення t, мають більший вплив на форму кривої.

Поліноми Бернштейна мають властивість симетричності відносно зміни t на t – 1:

$$b_i^n(1-t) = B_{n-1}^n(t). \quad (11)$$

Це можна безпосередньо довести підставивши 1 – t замість t у базову формулу розрахунку полінома – вираз (2):

$$B_i^n(1-t) = \binom{n}{i}(1-t)^i t^{n-i}\,. \quad (12)$$

За означенням біномального коефіцієнта (3) маємо:

$$\binom{n}{i} = \binom{n}{n-1}, \quad (13)$$

тому:

$$b_i^n(1-t) = \binom{n}{n-i} t^{n-i}(1-t)^i = B_{n-i}^n(t). \quad (14)$$

Нехай ми міняємо місцями початкову та кінцеву точки , тобто замість $P_0,P_1,\ldots,P_n$ використовуємо $P_n,P_{n-1},\ldots,P_0$. Тоді нова крива B`(t) записується як:

$$B`(t) = \sum_{i=0}^{n} P_{n-i}\, B_i^n(t)\,. \quad (15)$$

Використаємо симетричність базисних поліномів (14):

$$b_i^n(t) = B_{n-i}^n(1-t). \quad (16)$$

Підставивши його у вираз (15), отримаємо:

$$B`(t) = \sum_{i=0}^{n} P_{n-i}\, B_{n-i}^n(1-t)\,. \quad (17)$$

Для спрощеного порівняння проведемо заміну індексу j = n – i:

$$B`(t) = \sum_{j=0}^{n} B_j^n(1-t)P_j\,. \quad (18)$$

Вихідна крива Безьє B(t) при зміні параметра t на t – 1 має вигляд:

$$B(1-t) = \sum_{i=0}^{n} B_i^n(1-t)P_i\,. \quad (19)$$

Згідно з останнім виразом для $B`(t)$, очевидно, що $B`(t) = B(1-t)$. Це доводить, що зміна напрямку параметра t (напрямку траєкторії) не змінює її геометричну

форму через симетричність базисних поліномів Бернштейна. Цю властивість називають *симетричністю* кривих Безьє [21].

Ще однією властивістю кривих Безьє є їх *дотичність до контрольного багатокутника (полігону) в кінцевих точках* (рис.1.1). Це можна легко відслідкувати, взявши першу похідну цієї кривої:

$$\boldsymbol{B}(t) = \frac{d\boldsymbol{r}(t)}{dt} = n\sum_{i=0}^{n-1}(\boldsymbol{b_{i+1}} - \boldsymbol{b_i})b_i^{n-1}(t)\,, \quad 0 \le \mathrm{t} \le 1. \quad (11)$$

Зокрема, ми отримуємо $B(0) = n(b_1 - b_0)$ та $r(1) = n(b_n - b_{n-1})$[web.mit]. Вираз (11) можна спростити встановивши $\Delta b_i = b_{i+1} - b_i$:

$$B(t) = n\sum_{i=0}^{n-1}\Delta b_i B_{i,n-1}(t)\,, \qquad 0 \le t \le 1. \qquad (12)$$

Перша похідна кривої Безьє називається годографом (кривою Безьє на степінь менше контрольні точки якої є різницями сусідніх точок оригінальної кривої, помноженими на n). Вони корисні при вивченні точок перегину, сингулярності та інших явищ у механіці, геометрії чи геофізиці [21].

Більш спрощеним варіантом дослідження властивостей кривих Безьє є опис лінійної кривої, де n = 1. У цьому випадку побудова здійснюється на основі двох контрольних точок P_0 і P_1:

$$B(t) = (1-t)P_0 + tP_1 \qquad (13)$$

Вираз (13) описує пряму лінію між двома точками. Коли t змінюється від 0 до 1, крива «рухається» безпосередньо від P_0 до P_1, що унеможливлює додавання будь якої кривизни. Поліноми Бернштейна у цьому випадку матимуть такий вигляд:

$1.\, b_0^1(t) = 1 - t\,,$

$2.\, b_1^1(t) = t.$ (14)

Такий варіант найпростішої кривої Безьє допомагає визначити їх основну ідею – *лінійну інтерполяцію між суміжними точками*. Це означає, що для трьох або більше точок застосовується повторна інтерполяція між результатами попередніх інтерполяцій.

Не менш важливою є *властивість зменшення варіації кривої Безьє*. Вона означає, що крива Безьє має менш складну форму порівняно з багатокутником, утвореним її контрольними точками. Вона ніколи не буде перетинати пряму (у двовимірному просторі) або площину (у тривимірному просторі) більше разів, ніж це робить контрольний багатокутник (полігон, утворений з'єднанням усіх контрольних точок). Через цю властивість крива Безьє має тенденцію згладжувати нерівності чи різкі зміни форми, які можуть бути присутніми в контрольному багатокутнику. Таким чином, форма кривої виглядає більш плавною. Якщо багатокутник, утворений контрольними точками, є опуклим (тобто всі його кути менші за 180 градусів), то і сама крива також буде опуклою. Це підкреслює той факт, що крива не виходить за межі "характеру" свого контрольного багатокутника [4, 21].

Зважаючи на всі ці властивості, криві Безьє демонструють свою універсальність та ефективність у математичному моделюванні плавних і точних форм. Їхня здатність точно описувати геометричні об'єкти через контрольні точки, забезпечуючи при цьому стабільність і симетрію, робить їх потужним інструментом для розв'язання різнотипних задач.

2.2 Шляхи використання кривих Безьє за галузями застосування

Криві Безьє є унікальним математичним інструментом, який активно використовується в сучасній графіці, дизайні, інженерії та багатьох інших галузях. Більш практичними та значущими вважаються криві Безьє другого та третього ступенів, тобто квадратичні та кубічні. Криві вищих ступенів при обробці вимагають більшого обсягу обчислювальних операцій, що робить цей процес надто трудомістким та коштовним. Об'єднавши криві нижчого порядку, можна створити складену криву Безьє, яка матиме вигляд єдиної кривої, але буде значно простішою у моделюванні та використанні [32].

Однією з найбільш поширених сфер застосування кривих Безьє є *комп'ютерна графіка* та *графічний дизайн*. Векторна графіка базується на математичному описі об'єктів, а криві Безьє слугують її фундаментом. Стандарти векторної графіки, такі як .svg, мови векторної графіки (наприклад PostScript) чи програми векторної графіки, такі як Adobe Illustrator, Linearity Curve, CorelDraw, Artline, Inskape, Timeworks Publisher і Allegro, використовують раніше згадані складені криві Безьє для опису та модифікацій геометричних об'єктів, створення логотипів, ілюстрацій, а також для редагування зображень. У деяких з них користувач може скористатися інструментом «перо», основою яких є саме криві Безьє (рис. 2.1).

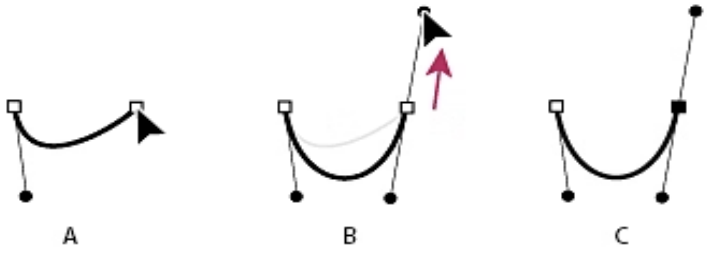

Рисунок 2.1 Принцип моделювання кривої інструментом «перо» в Adobe illustrator[8].

Важливим процесом у цьому випадку є растеризація кривої Безьє, тобто її перетворення з математично визначеної форми в набір дискретних точок, які можна зобразити на піксельній сітці. Найпростіший спосіб — обчислити багато близько розташованих точок, з'єднати їх відрізками й растеризувати отриману лінію. Однак, якщо точки занадто віддалені, результат може виглядати нерівним.
Більш точним підходом є адаптивний метод рекурсивного підрозділу, що перевіряє, наскільки крива наближена до прямої, і, за необхідності, ділить її на менші сегменти для подальшої обробки [27, 32]. Цей процес дозволяє отримати плавніші контури, які точно відповідають математичній моделі кривої. Різниця між растровим і векторним відображенням стає особливо помітною при масштабуванні: векторні шрифти зберігають чіткість і деталізацію, тоді як растрові можуть втрачати якість через обмежену роздільну здатність (рис. 2.2).

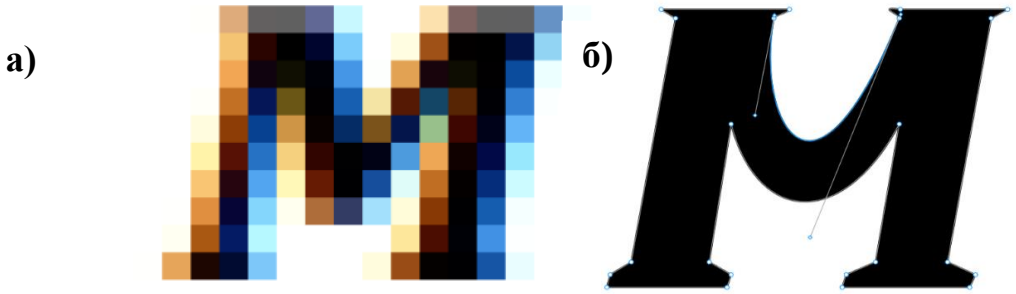

Рисунок 2.2 Відмінність растрової (а) та векторної (б) візуалізацій.

Криві Безьє, які мають властивість моделювання з легким масштабуванням та модифікацією контурів, широко застосовуються в *лазерному різанні*, *гравіруванні* та *3D-друку*.

Криві Безьє мають важливе значення для створення плавних траєкторій руху в *анімації*. У програмному забезпеченні, такому як Adobe Animate чи Blender, вони слугують засобами моделювання руху об’єктів, де кожен кадр генерується на основі заданих кривих, що дозволяє також контролювати їх швидкість. Такі анімації можна зустріти у різнотипній мультимедії чи кінематографі у вигляді повної (2D чи 3D) анімації, візуальних ефектів, динамічних переходів тощо [2]. Наприклад, у *тривимірній анімації* криві Безьє застосовуються для визначення траєкторій камер та інтерполяції ключових кадрів, що також забезпечує плавність і природність рухів (рис. 2.3).

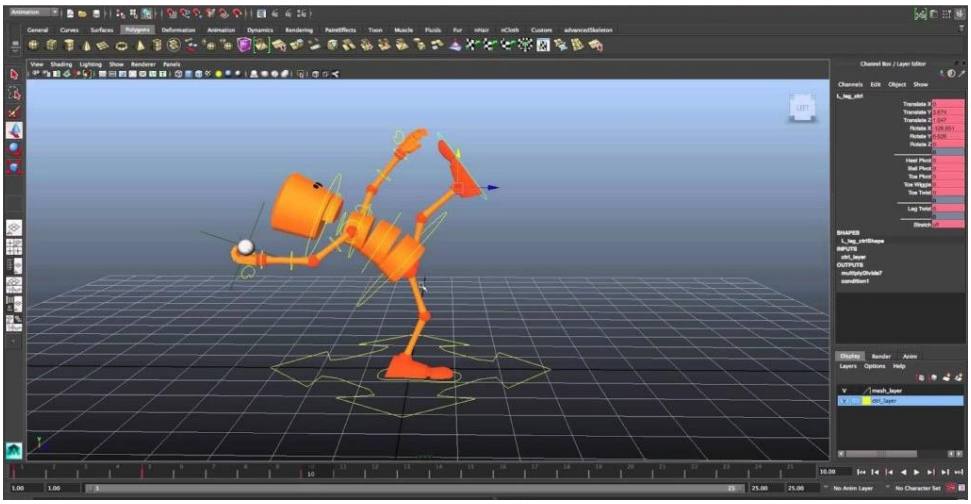

Рисунок 2.4 Налаштування траєкторій рухів об’єкта у 3D-анімаціях [15].

Цей функціонал застосовується і *в розробці відеоігор*, де криві також використовуються для моделювання траєкторій руху персонажів, камер та об'єктів.

Вони також є важливими для генерації геометричних форм і програмування рухів NPC, забезпечуючи природність та динамічність ігрових сценаріїв.

У сучасному *веб-дизайні* й розробці анімацій (CSS, JavaScript) вони задають динаміку руху елементів через функції easing, такі як cubic-bezier() (рис.2.4).

```css
.box {
  width: 50px;
  height: 50px;
  background-color: blue;
  position: relative;
  animation: move 2s cubic-bezier(0.25, 1, 0.5, 1) infinite;
```

Рисунок 2.4 Фрагмент коду з використанням CSS-функції cubic-bezier().

За допомогою таких CSS-функцій розробники можуть задавати унікальні ефекти переходу між елементами. Крім того, SVG-графіка, що базується на кривих Безьє, забезпечує створення інтерактивних векторних зображень, які адаптуються до різних розмірів екрана без втрати якості[30].

Не менш важливим є і застосування кривих Безьє в *медичній галузі*. Наприклад, у магнітно-резонансній томографії (МРТ) або комп'ютерній томографії (КТ) вони допомагають виділяти ключові області для подальшого аналізу або хірургічного планування. Для дослідження кровообігу та розробки пристроїв, таких як стенти, криві Безьє застосовуються для моделювання складних судинних структур. Це дозволяє вивчати взаємодію між судинами та медичними пристроями. При розробці протезів і біонічних кінцівок криві Безьє використовуються для створення ергономічних і анатомічно точних форм, що підвищує комфорт та функціональність виробів[3, 1].

Таким широким спектром застосування криві Безьє доводять свою ефективність та значний потенціал для подальшого використання. Їх інтеграція з новітніми технологіями, такими як штучний інтелект, 3D-друк і робототехніка, відкриває нові можливості для вдосконалення багатьох процесів і створення інноваційних продуктів.

Висновки до розділу 2

В розділі розглянуто структурні елементи математичного моделювання кривих Безьє. В їх основі лежить використання поліномів Бернштейна, які задають параметричну залежність точок кривої від контрольних координат. Доведено алгебраїчні та функціональні властивості кривих Безьє, а саме:

- визначення базисних кінцевих точок кривих Безьє,
- побудова структурних елементів на базі поліномів Бернштейна,
- симетричність кривої на основі симетричності поліномів Бернштейна відносно параметра t,
- дотичність до контрольного багатокутника (полігону) в кінцевих точках,
- лінійна інтерполяція між суміжними точками,
- властивість зменшення варіації кривої.

Вони демонструють простоту та універсальність досліджуваних кривих, їх практичну значущість в математичному моделюванні.

Продемонстровано широкий спектр застосування кривих Безьє у таких галузях як комп’ютерна графіка, графічний дизайн, анімація, лазерне різання, гравірування, 3D-друк, розробка відеоігор, веб-дизайн, медицина, тощо. Окреслено подальші перспективи та імовірні напрямки використання та розвитку досліджень кривих Безьє.

Таким чином, враховуючи отримані результати аналізу, криві Безьє є не лише важливим математичним концептом, але й універсальним інструментом для вирішення міждисциплінарних задач. Їхня простота, гнучкість і функціональність забезпечують незамінну роль у сучасній науці, інженерії та технологіях.

РОЗДІЛ 3.
ПРОГРАМНА РЕАЛІЗАЦІЯ КРИВИХ БЕЗЬЄ

3.1 Використання штучного інтелекту для апроксимації траєкторій кривими Безьє

У сучасних дослідженнях, пов’язаних із просторовим аналізом і моделюванням, зростає значення інструментів, здатних автоматично знаходити найкраще математичне представлення складних траєкторій. Одним із таких інструментів є криві Безьє, що використовуються для апроксимації ліній різної складності — від простих відрізків до складних контурів і рухів. Застосування штучного інтелекту в цій сфері дозволяє значно підвищити точність і ефективність побудови таких кривих, особливо за наявності великої кількості вхідних даних або складних форм [13].

Під штучним інтелектом (ШІ) у цьому контексті розуміється сукупність алгоритмічних методів, які дозволяють автоматизовано підбирати контрольні точки для кривих Безьє, орієнтуючись на задану траєкторію. Це може бути реалізовано, зокрема, за допомогою евристичних стратегій оптимізації або пошуку за зразком. Такий підхід дає змогу ефективно апроксимувати як ідеалізовані математичні контури, так і реальні просторові об'єкти — наприклад, фрагменти ландшафту, траєкторії руху чи контури елементів місцевості.

У цьому дослідженні для реалізації інтелектуальної апроксимації траєкторій кривими Безьє використано мовну модель ChatGPT, яка функціонує на основі сучасних архітектур глибокого навчання. Такий вибір зумовлений здатністю моделі ефективно інтерпретувати текстові інструкції, генерувати програмний код, адаптований до конкретних задач, та здійснювати логічну обробку даних. Крім того, ChatGPT може слугувати універсальним інструментом для автоматизації алгоритмічних побудов, що особливо важливо в умовах потреби швидкої генерації та візуалізації математичних кривих на основі заданих параметрів або вхідних координат [19].

Постановка задачі полягає в тому, щоб знайти наближену криву Безьє, яка найточніше повторює початкову траєкторію, задану як набір дискретних точок. У класичному підході така апроксимація виконується аналітичними методами, наприклад методом найменших квадратів. Натомість у контексті штучного інтелекту траєкторія розглядається як вхід до нейронної мережі, а контрольні точки кривої Безьє — як вихідні значення, які потрібно передбачити.

Архітектура моделі нейронної мережі, призначеної для апроксимації траєкторій кривими Безьє, залежить від складності вхідних даних, бажаної точності, а також обчислювальних ресурсів. У загальному випадку можна виділити два основні типи архітектур: щільно з'єднані мережі (MLP) та послідовні моделі (RNN, LSTM, трансформери). У контексті використання ШІ варто розглянути як базову, так і розширену архітектуру, яка може бути адаптована до задачі апроксимації кривих .

MLP (Multilayer Perceptron) є найпростішим і базовим варіантом архітектури, що ефективно працює у випадках, коли вхідні дані мають фіксовану довжину. Він ґрунтується на концепції регресійного прогнозування: нейромережа приймає на вхід координати точок, що описують траєкторію, і повертає координати контрольних точок кривої Безьє, які забезпечують найкращу наближену побудову цієї траєкторії [34].

Щоб реалізувати описану архітектуру, за допомогою нейронної мережі було сформовано запит побудови повнофункціональної моделі типу MLP (Multilayer Perceptron), яка навчається на прикладах випадкових кривих Безьє третього порядку. В результаті чіткого формулювання, мовна модель ChatGPT створює повноцінний код на базі Python 3 для апроксимації траєкторій кривими Безьє за обраним типом (дод. 1).

Центральним етапом побудови цього коду нейромережею є створення функції генерації таких кривих, де для кожної з них випадковим чином формуються чотири контрольні точки у двовимірному просторі. Ці точки передаються у функцію, яка обчислює координати самої кривої за допомогою класичної формули Безьє (рис. 3.1).

```python
def bezier_curve(control_points, num_points=100):
    t = np.linspace(0, 1, num_points).reshape(-1, 1)
    P0, P1, P2, P3 = control_points
    curve = (
        (1 - t)**3 * P0 +
        3 * (1 - t)**2 * t * P1 +
        3 * (1 - t) * t**2 * P2 +
        t**3 * P3
    )
    return curve
```

Рис. 3.1 Функція побудови кривої Безьє третього порядку на основі заданих контрольних точок.

Цей фрагмент коду виконує дискретизацію параметра t в межах [0, 1], а потім обчислює кожну точку кривої на основі параметричної формули кубічної кривої Безьє. Така реалізація дає можливість формувати множину траєкторій, які згодом використовуються як навчальні приклади для моделі.

Нейронна мережа, використана у цьому дослідженні, складається з трьох повнозв'язних шарів. Перший шар приймає на вхід лінеаризовану траєкторію з 40 значень (20 точок по 2 координати), обробляє її за допомогою нелінійної функції активації ReLU та передає результат на наступний шар (рис.).

```python
class BezierMLP(nn.Module):
    def __init__(self, input_dim=40, hidden_dim=128, output_dim=8):
        super(BezierMLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, output_dim)
        )
    def forward(self, x):
        return self.model(x)
```

Рис. 3.3 Архітектура програмування нейронної мережі на основі MLP.

У цій реалізації помітно, що мережа має два приховані шари по 128 нейронів кожен, що дає їй достатню ємність для апроксимації нелінійного відображення між траєкторією та відповідними контрольними точками. Вихідний шар складається з 8 нейронів, що відповідає координатам 4 контрольних точок кривої Безьє.

Щоб здійснити навчання моделі, необхідно сформувати датасет пар «траєкторія – контрольні точки». Кожна траєкторія будується за згенерованими випадковими контрольними точками, а самі контрольні точки зберігаються як мітки. Цей процес автоматизовано наступною функцією, яку відображено на рисунку 3.2.

```
def create_dataset(num_samples=1000):
    X, Y = [], []
    for _ in range(num_samples):
        curve, control_points = generate_sample_curve()
        X.append(curve.flatten())
        Y.append(control_points.flatten())
    return np.array(X, dtype=np.float32), np.array(Y, dtype=np.float32)
```

Рис. 3.2 Автоматизація формування датасету пар «траєкторія – контрольні точки».

Після формування навчального набору модель тренується шляхом мінімізації функції середньоквадратичної помилки (MSE), що вимірює відхилення передбачених координат контрольних точок від істинних. Під час навчання мережа поступово навчається відновлювати просторову структуру вхідної траєкторії, тобто шукати таку конфігурацію контрольних точок, яка дає найкраще наближення у формі кривої Безьє .

Після завершення тренування модель може передбачати контрольні точки нових траєкторій. Використавши онлайн-платформу Google Collab, роботу цього алгоритму можна візуалізувати у форматі рисунка (рис. 3.4 (а)).

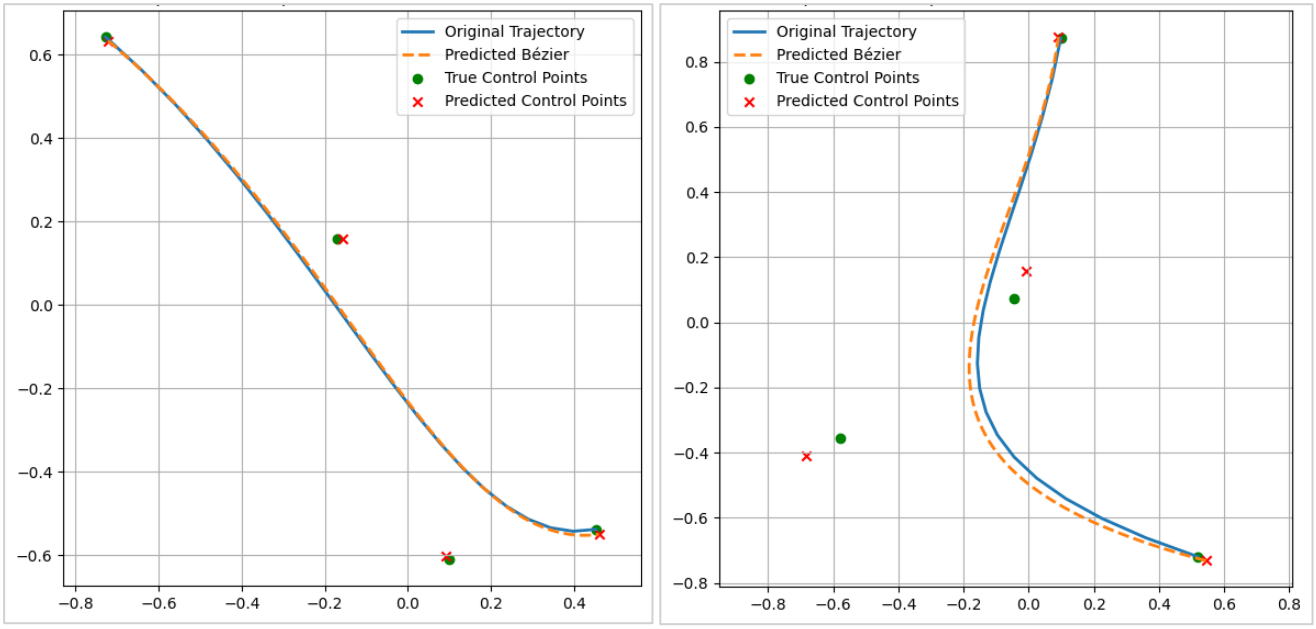

Рис. 3.4 Апроксимація заданої траєкторії кривою Безьє третього порядку, побудованою на основі архітектурних моделей а) MLP та б) LSTM.

Синя лінія позначає початкову траєкторію, що була згенерована за допомогою істинних контрольних точок. Помаранчева пунктирна крива — це результат, отриманий шляхом побудови кривої Безьє на основі передбачених мережею контрольних точок. Зелені точки — істинні контрольні точки, червоні хрестики — передбачені. Як видно з візуалізації, модель демонструє високу точність апроксимації, що свідчить про її здатність реконструювати просторову форму навіть на основі компактного векторного представлення.

Таким чином, нейронна мережа MLP виступає в ролі нелінійного регресора, який навчено здійснювати обернене відображення: з форми траєкторії до параметрів, які її визначають (контрольних точок). Такий підхід дозволяє автоматизувати побудову геометричних моделей, зменшуючи необхідність у ручному підборі параметрів, і може бути використаний у задачах комп'ютерної графіки, геоінформатики, робототехніки чи цифрового моделювання простору.

У контексті апроксимації траєкторій кривими Безьє важливо також розглянути послідовні моделі, такі як RNN (Recurrent Neural Networks), LSTM (Long Short-Term Memory) та трансформери, які відіграють важливу роль завдяки здатності працювати з послідовними даними змінної довжини. На відміну від MLP, що оперує фіксованими векторами, ці архітектури враховують часову або

просторову послідовність точок, зберігаючи інформацію про попередні елементи ряду.

RNN і LSTM дозволяють моделі формувати уявлення про локальну структуру траєкторії, ефективно кодувати залежності між точками та адаптуватися до варіативності форм. Трансформери, у свою чергу, завдяки механізму самоуваги (self-attention) здатні аналізувати глобальні взаємозв'язки між усіма точками траєкторії одночасно, що забезпечує високу гнучкість та точність при прогнозуванні контрольних точок. Таким чином, послідовні моделі забезпечують більш глибоке опрацювання складних структур вхідних даних і є потужним інструментом у задачах апроксимації, де важливим є збереження контексту та порядку точок.

У цьому дослідженні реалізовано архітектуру на основі LSTM — варіанту рекурентної мережі, який краще зберігає довготривалі залежності та менш схильний до затухання градієнта порівняно з класичними RNN. Для побудови кривої Безьє з передбачених контрольних точок нейромережа також створює повноцінний код на базі Python 3 для апроксимації траєкторій кривими Безьє за обраним типом (дод. 2).

Як і у випадку з MLP, нейромережа використовує класичну формулу кривої третього порядку (рис. 3.1). Цей метод дозволяє побудувати гладку та диференційовану криву, що проходить поблизу початкових і кінцевих точок і спрямована відповідно до внутрішніх контрольних векторів.

Вхідними даними для моделі є траєкторія, представлена як послідовність точок розмірності $N \times 2$, де N — кількість дискретизованих точок траєкторії, а кожна точка містить координати x_i, y_i. На основі цієї послідовності модель має передбачити вектор із восьми значень, які формують чотири контрольні точки кривої Безьє третього порядку, який у програмному варіанті має характерну архітектуру (рис.).

```python
class BezierLSTM(nn.Module):
    def __init__(self, input_size=2, hidden_size=64, num_layers=2, output_size=8):
        super(BezierLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = out[:, -1, :]  # використовуємо останній часовий крок
        return self.fc(out)
```

Рис. 3.5 Архітектура програмування нейронної мережі на основі LSTM.

LSTM-модуль приймає на вхід послідовність координат, послідовно обробляє кожен крок траєкторії та зберігає інформацію про контекст у прихованих станах. Після завершення проходження всієї послідовності модель формує вихідний вектор на основі останнього прихованого стану. Цей вектор передається до повнозв'язного шару, який здійснює проєкцію в простір розмірності 8 (чотири контрольні точки, по дві координати кожна).

Процес навчання нейронної мережі на основі архітектури LSTM має певні відмінності від класичної багатошарової перцептронної моделі (MLP), оскільки він враховує послідовну природу вхідних даних. У контексті апроксимації траєкторій кривими Безьє це означає, що вхідна послідовність точок повинна зберігати порядок, у якому точки з'являються вздовж кривої, оскільки саме цей порядок несе просторову інформацію про форму та напрям руху траєкторії. На відміну від MLP, де координати точок спочатку лінеаризуються в один вектор, у LSTM модель приймає тривимірний тензор розмірності (1, N, 2), де N — кількість дискретних

точок на траєкторії, а 2 — це кількість координат (x, y) (рис. 3.6) [14].

```
model = BezierLSTM()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

num_epochs = 500
for epoch in range(num_epochs):
    model.train()

    inputs = torch.tensor(trajectory, dtype=torch.float32).unsqueeze(0)  # форма: (1, N, 2)
    targets = torch.tensor(control_points.flatten(), dtype=torch.float32).unsqueeze(0)  # (1, 8)

    outputs = model(inputs)
    loss = criterion(outputs, targets)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 50 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.6f}")
```

Рис. 3.6 Цикл навчання LSTM-моделі для апроксимації контрольних точок кривої Безьє на основі вхідної траєкторії.

Такий формат дозволяє LSTM поетапно обробляти кожну точку, накопичуючи контекст та залежності між сусідніми значеннями, що особливо важливо для відображення кривизни та напрямку змін.

Для навчання мережі використовується функція втрат середньоквадратичної помилки (Mean Squared Error), яка порівнює передбачені координати контрольних точок з істинними значеннями (див. рис. 3.6). У процесі оптимізації застосовується метод градієнтного спуску з адаптивною швидкістю навчання (оптимізатор Adam), що дозволяє моделі швидко сходитися навіть за великої кількості параметрів. Протягом кожної ітерації модель переводиться в режим навчання, після чого отримує на вхід послідовність координат, а на виході генерує вектор координат контрольних точок кривої Безьє. Обчислюється похибка, модель оновлює свої «ваги», і цей цикл повторюється задану кількість епох. Таким чином, модель поступово навчається знаходити оптимальні контрольні точки, які забезпечують найближчу апроксимацію до заданої траєкторії.

На відміну від MLP, яка просто вивчає відповідність між повністю сплюснутим вектором координат та контрольними точками, LSTM аналізує просторову структуру всієї траєкторії, враховуючи її зміну в часі або за порядком точок. Це дає змогу ефективніше моделювати складніші криві, де локальні вигини та загальна форма мають значення.

Візуалізація результату, що представлена на рисунку 3.4 (б), демонструє якість апроксимації траєкторії моделлю. Синя лінія відображає оригінальну траєкторію, згенеровану на основі справжніх контрольних точок, тоді як помаранчева пунктирна крива представляє траєкторію, апроксимовану на основі передбачених LSTM контрольних точок. Зелені крапки позначають справжні контрольні точки, а червоні хрестики — передбачені. Як видно з рисунка, навіть при доволі обмеженій кількості епох тренування модель демонструє здатність коректно відтворювати загальну форму траєкторії.

Таким чином, використання LSTM для апроксимації траєкторій кривими Безьє дозволяє досягти високої якості наближення навіть у випадках, коли траєкторія має складну геометрію або просторові коливання. Порівняно з базовою архітектурою MLP, послідовна модель краще враховує порядок і залежності між точками, що покращує узгодженість передбачених контрольних точок із реальною формою траєкторії.

3.2 Програмна реалізація побудови кривих Безьє різної складності з використанням OpenGL

У контексті візуалізації та графічного рендерингу геометричних об’єктів, побудова кривих Безьє в реальному часі набуває особливого значення. Одним із найбільш придатних інструментів для такої задачі є бібліотека OpenGL — кросплатформене API для рендерингу дво- та тривимірної графіки, яке забезпечує низькорівневий контроль над графічним конвеєром.

Застосування OpenGL дає змогу не лише точно відтворити математично описану криву Безьє, але й інтегрувати її у складні сцени, забезпечити взаємодію з користувачем та досягти високої продуктивності. Побудова кривих за допомогою OpenGL передбачає поетапне виконання: визначення контрольних точок, обчислення положень точок на кривій за формулою Бернштейна та подальший рендеринг результату на екран. Цей підхід має переваги як у плані гнучкості представлення графічних об'єктів, так і в аспекті апаратного прискорення, що особливо важливо для інтерактивних додатків та візуалізацій у режимі реального часу [16, 17].

У межах реалізації побудови кривих Безьє за допомогою OpenGL було розроблено програмний модуль, що ілюструє динамічну візуалізацію кривої третього порядку з можливістю анімації (дод. 3). У результаті запуску реалізованої програми було отримано її динамічну візуалізацію, яка демонструє її поведінку у просторі при проєкції на площини X–Y, X–Z та Y–Z. На рисунках 3.7 наведено відповідно вигляд кривої відносно цих площин.

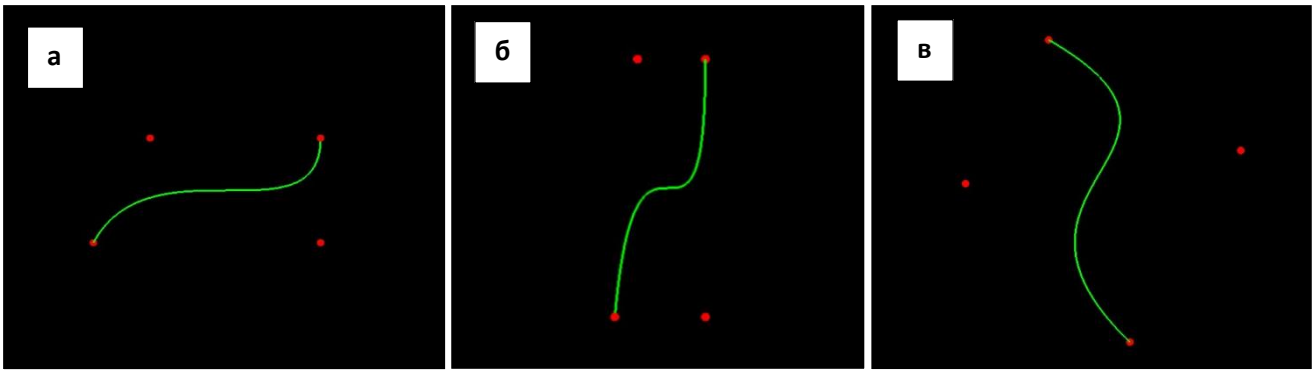

Рис. 3.7 Візуалізація кубічної кривої Безьє у а) фронтальній, б)горизонтальній та в)боковій проєкціях [33].

Такий підхід дозволяє проаналізувати зміну форми кривої під впливом анімації з різних точок огляду, що є важливим у випадках тривимірного моделювання об'єктів або траєкторій руху. Особливо помітною є деформація у площині X–Y, що відповідає за більшість візуальних коливань, оскільки компоненти синусоїдальних змін задано переважно для цих координат [15].

Основу програми становить функція *drawBezierCurve()*, яка відповідає за побудову кривої Безьє та виведення контрольних точок. Крива обчислюється за допомогою параметричної формули третього порядку, яка враховує чотири контрольні точки P_0, P_1, P_2, P_3. Кожна точка кривої для параметра t ∈ [0,1] визначається наступною формулою [28]:

$$B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \quad (15)$$

Відповідну реалізацію цього обчислення в коді можна побачити на рисунку 3.8.

```
float x = pow(1 - t, 3) * p0x + 3 * t * pow(1 - t, 2) * p1x +
          3 * pow(t, 2) * (1 - t) * p2x + pow(t, 3) * p3x;
float y = pow(1 - t, 3) * p0y + 3 * t * pow(1 - t, 2) * p1y +
          3 * pow(t, 2) * (1 - t) * p2y + pow(t, 3) * p3y;
```

Рис. 3.8 Реалізація математичної формули кубічної кривої Безьє у функції drawBezierCurve().

У продемонстрованій реалізації особливістю є залежність контрольних точок P_1 та P_2 від параметру *animationParameter*, який з часом змінюється за синусоїдальним законом (рис. 3.9).

```
float p1x = -1.0 + sin(animationParameter), p1y = 2.0;
float p2x = 1.0 + cos(animationParameter), p2y = -2.0;
```

Рис. 3.9 Залежність контрольних точок кубічної кривої від параметра анімації.

Завдяки цьому відбувається динамічна зміна кривої, що створює ефект анімації. Сам параметр анімації оновлюється у функції *update()* (рис. 3.10).

```
void update(int value) {
    animationParameter += 0.05;
    if (animationParameter > 2 * Pi) {
        animationParameter -= 2 * Pi;
    }
    glutPostRedisplay();
    glutTimerFunc(16, update, 0);
}
```

Рис. 3.10 Оновлення параметра анімації через функцію update().

Таймер викликає цю функцію кожні 16 мс., що забезпечує частоту приблизно 60 кадрів на секунду. Оскільки *animationParameter* впливає на синусоїдальні коливання, форма кривої змінюється плавно, створюючи візуальний ефект «живої» траєкторії.

Крім самої кривої, у сцені відображаються і контрольні точки. Вони візуалізуються як червоні маркери з використанням примітиву *GL_POINTS* (рис. 3.11).

```
glBegin(GL_POINTS);
glColor3f(1.0, 0.0, 0.0); // Червоний колір
glVertex3f(p0x, p0y, 0.0);
glVertex3f(p1x, p1y, 0.0);
glVertex3f(p2x, p2y, 0.0);
glVertex3f(p3x, p3y, 0.0);
glEnd();
```

Рис. 3.11 Візуалізація контрольних точок через OpenGL-примітив.

Також важливою є частина коду, що відповідає за налаштування тривимірної сцени та освітлення. Функція *scene()* конфігурує джерело світла, проєкцію та положення камери (рис. 3.12).

```
glOrtho(-5, 5, -5, 5, 2, 15);
gluLookAt(0.0, 0.0, 5.0,
          0.0, 0.0, 0.0,
          0.0, 1.0, 0.0);
```

Рис. 3.12 Налаштування сцени у функції scene() (камера, світло, проєкція).

Завдяки цьому забезпечується ортографічна проєкція сцени та зручний огляд графіки. Крім цього, функція *keyboard()* надає користувачеві можливість обертати

сцену навколо координатних осей у відповідь на натискання клавіш x, y або z, що реалізовано через виклики *glRotated()*.

Основний цикл програми організовується у функції *main()* (рис. 3.13).

```
glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGB);
glutDisplayFunc(display);
glutKeyboardFunc(keyboard);
glutTimerFunc(0, update, 0);
glutMainLoop();
```

Рис. 3.13 Ініціалізація OpenGL та запуск основного циклу у функції.

Цей фрагмент коду відповідає за ініціалізацію вікна, реєстрацію функцій обробки подій та запуск безкінечного циклу відображення сцени, яку ми можемо побачити на рисунках 3.7 у вигляді кубічної кривої Безьє [31].

У межах дослідження використання кривих Безьє в комп'ютерній графіці важливо розглянути приклад побудови анімованих кривих різної складності. Саме для цього додатковою необхідністю постає реалізація кривої п'ятого порядку за допомогою тієї ж бібліотеки – OpenGL (дод. 4). Не зважаючи на те, що програмна складова частково збігається з попереднім варіантом, де використано структуру побудови кубічної кривої, у цьому випадку задіяно шість контрольних точок, що відповідає кривій Безьє п'ятого порядку. Це уможливлює побудову складнішої траєкторії з багатшою геометричною структурою, що помітно на рис. 3.14.

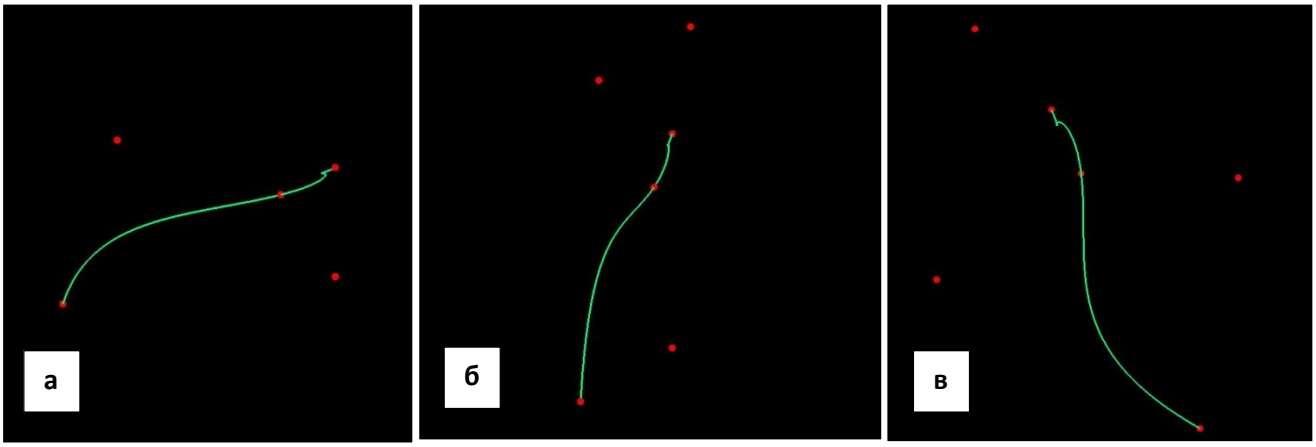

Рис. 3.14 Візуалізація кривої Безьє п'ятого порядку у а) фронтальній, б)горизонтальній та в)боковій проєкціях.

Програма реалізує анімовану візуалізацію кривої, форма якої змінюється в реальному часі під впливом періодичних змін координат контрольних точок.

Основна побудова здійснюється у функції *drawComplexBezierCurve()*, яка реалізує обчислення п’ятого порядку за узагальненою формулою Безьє [23]:

$$B(t) = \sum_{i=0}^{5} \binom{5}{i} (1-t)^{5-i} t^i P_i \ , \qquad (16)$$

На відміну від попередньої реалізації, де застосовувалося лише чотири точки, тут використано шість, включно зі статичними початковою та кінцевою точками P_0 та P_5. Це дозволяє відобразити криву, яка точніше моделює складні об'єкти або траєкторії у просторових системах. У коді продемонстрована формула 16 реалізована через вирази для кожної з шести контрольних точок (рис. 3.15).

```
float x = pow(1 - t, 5) * p0x +
          5 * t * pow(1 - t, 4) * p1x +
          10 * pow(t, 2) * pow(1 - t, 3) * p2x +
          10 * pow(t, 3) * pow(1 - t, 2) * p3x +
          5 * pow(t, 4) * (1 - t) * p4x +
          pow(t, 5) * p5x;
float y = pow(1 - t, 5) * p0y +
          5 * t * pow(1 - t, 4) * p1y +
          10 * pow(t, 2) * pow(1 - t, 3) * p2y +
          10 * pow(t, 3) * pow(1 - t, 2) * p3y +
          5 * pow(t, 4) * (1 - t) * p4y +
          pow(t, 5) * p5y;
```

Рис. 3.15 Реалізація математичної формули кривої Безьє п’ятого порядку у функції drawComplexBezierCurve().

Це дозволяє отримати точнішу апроксимацію складних траєкторій, особливо у випадках, коли потрібна більша варіативність форми або імітація природних рухів. Зокрема, збільшення кількості контрольних точок розширює простір керування формою кривої, дозволяючи локалізовано змінювати її конфігурацію без суттєвого впливу на всю лінію.

Оскільки завданням є створення анімації, координати частини контрольних точок змінюються динамічно відповідно до параметра *animationParameter*. Його також можна зустріти у попередньому програмному варіанті. Відмінним є те, що у випадку з кривою п’ятого порядку така залежність охоплює більше точок, ніж у кубічній кривій, що розширює можливості для формування складніших

деформацій. Значення координат задаються через тригонометричні функції з різною частотою, завдяки чому крива набуває складної, нестандартної форми під час анімації (рис.3.16).

```
float p1x = -2.0 + sin(animationParameter * 1.2),
      p1y = 2.0 + cos(animationParameter);
float p2x = 1.0 + cos(animationParameter * 0.7),
      p2y = -2.0 + sin(animationParameter * 1.5);
float p3x = 3.0 + sin(animationParameter * 0.5),
      p3y = 3.0 + cos(animationParameter * 0.8);
float p4x = 0.0 + cos(animationParameter * 1.3),
      p4y = 1.0 + sin(animationParameter * 1.1);
```

Рис. 3.16 Генерація координат для анімації кривої п'ятого порядку з різними частотами.

Застосування різних частот у функціях *sin* та *cos* до кожної координати контрольних точок створює ефект складної, негармонійної деформації, що нагадує природні рухи або хаотичні зміни форми. Анімація тут також реалізується за допомогою функції *update()*, яка збільшує значення параметра та повністю збігається зі структурою побудови для кубічної кривої (рис. 3.10). Таймер викликає оновлення з інтервалом у 16 мс, що забезпечує плавну анімацію. Зміна параметра відбувається у межах повного періоду 2π, після чого він обнуляється, що дає змогу створити циклічну деформацію кривої.

Однак така гнучкість супроводжується і певними недоліками. Збільшення порядку кривої зумовлює підвищену обчислювальну складність: зростає кількість операцій для обчислення координат кожної точки кривої, а також складність забезпечення стабільності анімації. Крім того, з додаванням нових контрольних точок втрачається інтуїтивна керованість формою: зміна однієї точки може непередбачувано впливати на загальний вигляд кривої, якщо не застосовано обмежень або алгоритмів згладжування.

Візуалізація кривої Безьє п'ятого порядку та пов'язаних з нею контрольних точок виконується з використанням стандартних інструментів OpenGL, зокрема графічних примітивів типу *GL_LINE_STRIP* для побудови контуру та *GL_POINTS* для маркування контрольних точок (рис. 3.17).

```
glBegin(GL_LINE_STRIP);
// обчислення кривої
glEnd();
```

Рис. 3.17 Фрагмент візуалізації кривої за допомогою примітиву GL_LINE_STRIP.

Зовнішній вигляд кривої контролюється параметром товщини лінії, що задається командою *glLineWidth()*, і кольором, встановленим функцією *glColor3f()*, завдяки чому вона чітко відображається на тлі сцени (рис. 3.18).

```
glBegin(GL_POINTS);
glColor3f(1.0, 0.0, 0.0);
glVertex3f(p0x, p0y, 0.0);
...
glVertex3f(p5x, p5y, 0.0);
glEnd();
```

Рис. 3.18 Візуалізація контрольних точок у вигляді червоних маркерів з GL_POINTS.

Контрольні точки, які визначають форму кривої, додатково візуалізуються у вигляді маркерів червоного кольору, що створюються за допомогою примітиву *GL_POINTS* у режимі згладжування. Їх розмір встановлюється через параметр *glPointSize()*, що дозволяє легко ідентифікувати положення кожної з них на площині чи в просторі. Така наочність дає можливість аналізувати вплив окремих точок на форму кривої під час анімації, особливо якщо координати точок змінюються в динаміці.

Камера, система координат та освітлення сцени налаштовуються у функції *scene()*, яка викликається під час ініціалізації програми. Вона встановлює ортогональну проєкцію *(glOrtho())*, позицію камери *(gluLookAt()),* джерела світла *(glLightfv())* і базові параметри візуалізації, включно з глибинним тестуванням і параметрами освітлення *(GL_LIGHTING, GL_LIGHT0, GL_COLOR_MATERIAL, GL_DEPTH_TEST)*. Це забезпечує тривимірне сприйняття сцени, а не лише плоску проєкцію, що особливо важливо для аналізу просторових кривих (рис. 3.12).

Крім того, реалізована можливість обертання сцени навколо осей координат шляхом натискання клавіш x, y та z. Цю функціональність забезпечує обробник

клавіатури *keyboard()*, у якому використано функцію *glRotated()* для зміни кута огляду (рис. 3.19).

```
if (key == 'x') { glRotated(10, 1, 0, 0); }
```

Рис. 3.19 Реалізація обертання сцени навколо координатних осей у функції keyboard().

Такий підхід дозволяє користувачеві інтерактивно переглядати криву з різних точок зору, що є корисним для просторового аналізу та пошуку аномалій у формі.

Як і в попередній реалізації кубічної кривої, основні виклики ініціалізації OpenGL, запуск циклу обробки подій та реєстрація функцій обробки візуалізації, анімації та вводу виконуються у функції *main()*. Вона ініціалізує вікно, задає його розміри, позицію та режими відображення (*GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGB*), після чого викликає *glutMainLoop()*, який підтримує безперервне оновлення графіки й обробку взаємодії з користувачем (рис. 3.13). Така структура дозволяє побудувати повноцінне інтерактивне графічне середовище для демонстрації та дослідження властивостей кривих Безьє.

Узагальнюючи порівняння фрагментів побудови двох продемонстрованих кривих Безьє дає змогу виявити ключові особливості кожного підходу та їх вплив на процес візуалізації в реальному часі (табл. 1).

Таблиця 1

Порівняння особливостей кривих Безьє третього та п’ятого порядків.

Характеристика	**Кубічна крива Безьє**	**Крива Безьє п’ятого порядку**
Кількість контрольних точок	4	6
Формула побудови	поліном третього порядку	поліном п’ятого порядку
Гнучкість форми	обмежена	висока
Локальна керованість	вища	зменшена
Обчислювальна складність	низька	вища
Реакція на зміну однієї точки	помірна	нелінійна, складна
Ступінь апроксимації складної траєкторії	помірний	високий

Чутливість до анімаційних змін	менша	більша
Візуальний ефект	плавна, передбачувана деформація	складна, варіативна деформація

З наявних даних можна виявити низку відмінностей, які виходять за межі простої зміни кількості контрольних точок. Насамперед, кубічна крива демонструє високу передбачуваність та легкість у керуванні формою. Завдяки невеликій кількості точок її геометрична поведінка є інтуїтивно зрозумілою: зміна розташування будь-якої з точок має локалізований та легко прогнозований вплив на загальний контур (рис. 3.20).

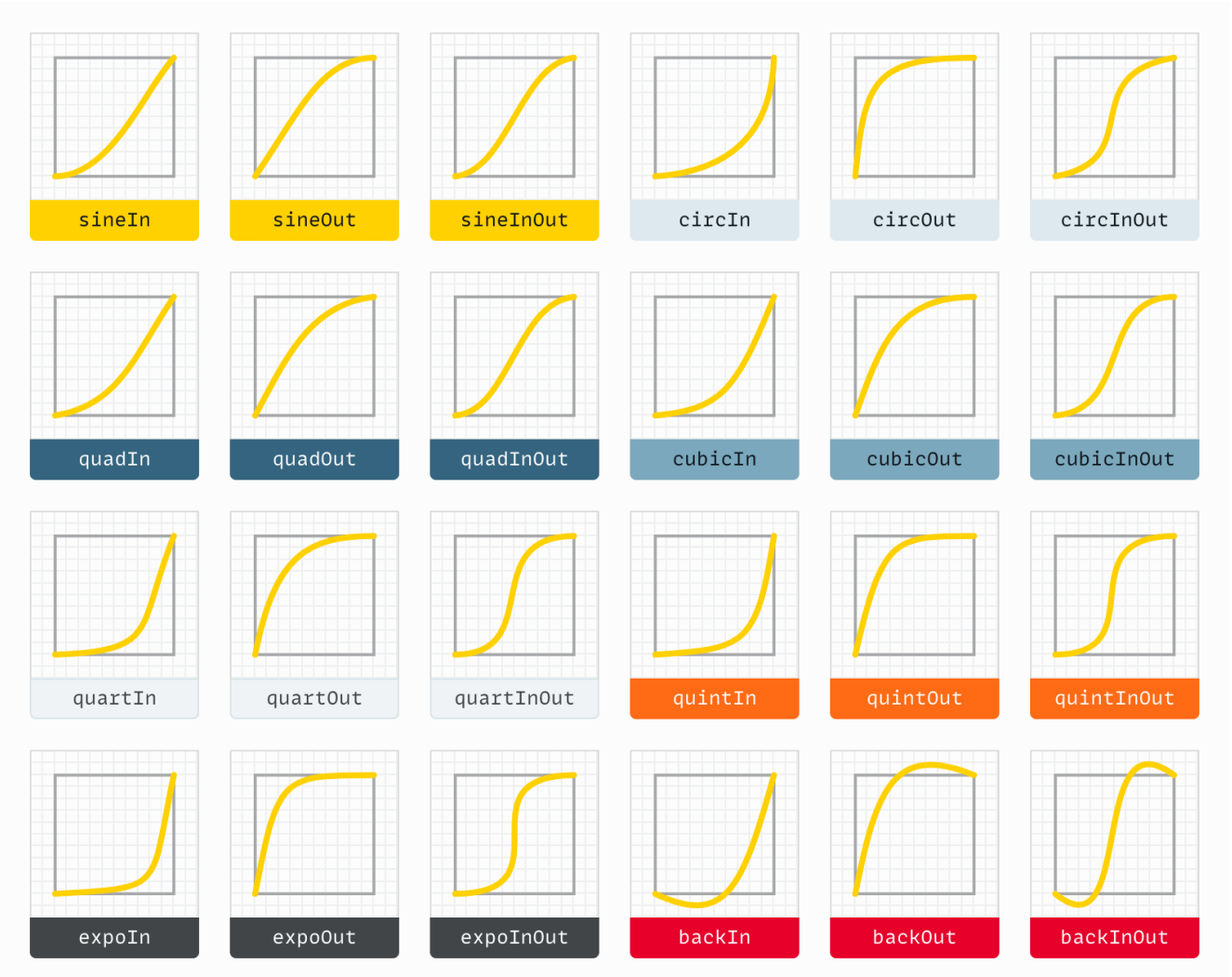

Рис. 3.20 Варіанти візуалізації кубічної кривої Безьє в залежності від зміни програмного параметра [34].

Це особливо зручно для задач, де важлива чітка відповідність між керувальними точками і формою кривої, зокрема у графічному дизайні, базовій анімації або при описі простих траєкторій руху.

Натомість крива п’ятого порядку, хоча й забезпечує значно більшу гнучкість у моделюванні складних контурів, характеризується більш вираженою нелінійністю взаємозв’язків між контрольними точками та формою. Зростає ймовірність появи глобальних змін форми при локальній модифікації окремої точки, що ускладнює ручне налаштування геометрії.

Крім того, різняться і апаратні вимоги. Обчислення п’ятого порядку потребує більшої кількості операцій над біноміальними коефіцієнтами та степеневими функціями, що впливає на продуктивність, особливо при високій частоті оновлення сцени. Це може бути критично важливим для систем з обмеженими ресурсами або при великій кількості одночасно візуалізованих об'єктів. Проте завдяки використанню OpenGL, що підтримує апаратне прискорення, навіть складні криві можуть бути ефективно оброблені при правильній оптимізації коду [20].

У підсумку, вибір між кривою третього та п’ятого порядку залежить від поставленої задачі: перша забезпечує простоту, ефективність та керованість, тоді як друга — високу гнучкість, складну динаміку та здатність моделювати реалістичні процеси. Таким чином, реалізація кривих різного порядку розкриває ширший спектр можливостей при проєктуванні графічних інтерфейсів, анімацій та візуалізацій складних систем.

Висновки до розділу 3

У розділі детально розглянуто програмну реалізацію побудови кривих Безьє з урахуванням як інтелектуальних підходів, так і класичних графічних методів. Основну увагу було приділено практичному аспекту побудови та візуалізації таких кривих, а також дослідженню можливостей їх апроксимації за допомогою моделей штучного інтелекту.

Здійснено апроксимацію заданих траєкторій за допомогою кривих Безьє шляхом прогнозування координат контрольних точок із використанням нейронних мереж. Зокрема, розглядалися моделі типу MLP та LSTM, які дозволили отримати точні результати апроксимації, зберігаючи плавність та форму кривих. Експериментальні результати засвідчили доцільність використання глибокого навчання для задач побудови кривих, що не тільки автоматизує процес, а й забезпечує достатньо високий рівень узагальнення на нових даних. Таким чином, використання нейронних мереж продемонструвало свою ефективність у контексті задач аналізу та синтезу геометричних форм.

Побудова кривих Безьє у середовищі OpenGL дозволила наочно продемонструвати процес генерації кривих третього та п’ятого порядків, а також реалізувати їхню анімацію. Візуалізація контрольних точок, самих кривих та поступового руху вздовж них надала можливість глибше зрозуміти динаміку зміни положення точок на кривій та вплив контрольних точок на форму траєкторії. Було продемонстровано, як змінюється крива в реальному часі в залежності від модифікацій координат контрольних точок. OpenGL реалізація забезпечила ефективну інтерактивну графіку з високою продуктивністю, що є особливо цінним для задач, де важлива швидкість рендерингу та якість візуалізації.

Узагальнюючи результати, можна стверджувати, що в рамках цього розділу було досягнуто гармонійного поєднання методів математичного моделювання кривих Безьє з інструментами штучного інтелекту та засобами комп’ютерної графіки. Це дозволило не лише виконати апроксимацію заданих траєкторій, а й візуалізувати отримані результати в інтерактивній формі.

ЗАГАЛЬНІ ВИСНОВКИ

Дослідження кривих Безьє дало можливість комплексно оцінити їх теоретичне, математичне, функціональне та практичне значення. На основі проведеного аналізу встановлено, що криві Безьє є ключовим інструментом у комп'ютерній графіці та суміжних галузях завдяки своїй простоті, гнучкості та ефективності. Актуальність теми зумовлена як їх широким використанням, так і потребою в глибшому розумінні їх математичних властивостей та прикладного потенціалу.

У роботі розв'язано наукову проблему систематизації знань про криві Безьє та їх адаптацію до сучасних потреб науки і техніки. Запропоновано цілісне дослідження їх математичних характеристик, зокрема алгебраїчних і геометричних властивостей, на основі базових розрахунків та диференціальної геометрії.

Здобуті результати підтверджують, що криві Безьє мають такі унікальні властивості: ступеневість, гнучкість у побудові складних форм завдяки параметричному опису та локальній дії контрольних точок, алгебраїчну основа на базі поліномів Бернштейна, яка забезпечує симетричність, гладкість та точне моделювання, низку математичних властивостей, які спрощують їх використання та модифікації (визначення базисних кінцевих точок, симетричність кривої на основі симетричності поліномів Бернштейна відносно параметра t, дотичність до контрольного багатокутника в кінцевих точках, лінійна інтерполяція між суміжними точками, зменшення варіації кривої), можливість повторної інтерполяція між результатами попередніх інтерполяцій, універсальність у застосуванні: від двовимірного графічного дизайну до тривимірного моделювання у галузях медицини, машинобудування, анімації, тощо.

На відміну від інших методів математичного моделювання, криві Безьє забезпечують більшу простоту управління формами та високий рівень точності. Це робить їх незамінними у практичних завданнях, таких як розробка інтерактивного дизайну, створення 3D-моделей або оптимізація виробничих процесів.

Практичне значення роботи полягає у можливості використовувати здобуті результати для вдосконалення алгоритмів комп'ютерного дизайну, розвитку CAD-

систем, а також створення нових підходів до моделювання складних геометричних об’єктів. Його значно посилено реалізацією побудови та апроксимації кривих Безьє з використанням сучасних інструментів штучного інтелекту (MLP, LSTM) і графічного середовища OpenGL. Запропоновані підходи дозволили автоматизувати процес побудови кривих на основі заданих траєкторій і забезпечити інтерактивну візуалізацію їх динаміки. Поєднання нейронних мереж з методами комп’ютерної графіки продемонструвало ефективність таких гібридних рішень у задачах аналізу, синтезу та оптимізації геометричних форм.

Рекомендації для подальших досліджень включають:

- Удосконалення алгоритмів побудови кривих та поверхонь Безьє, зокрема розширення можливостей адаптації до складних просторових форм у 3D-моделюванні та комп’ютерному дизайні;
- Оптимізація обчислювальних процедур для побудови кривих високих порядків з урахуванням обмежених ресурсів у реальному часі, що актуально для графічних рушіїв, візуалізації в ігрових застосунках та VR/AR;
- Інтеграція методів машинного навчання безпосередньо у процес побудови кривих — наприклад, для передбачення оптимальних контрольних точок на основі вхідних траєкторій або для автоматичного коригування кривих у CAD-системах;
- Дослідження потенціалу рекурентних і генеративних нейронних мереж (зокрема, LSTM, GRU, Transformer) у моделюванні динамічних траєкторій на базі кривих Безьє, включаючи адаптацію до змінних умов (наприклад, в задачах навігації або анімації);
- Розробка інтерактивних програмних інтерфейсів для візуального проєктування кривих із використанням елементів штучного інтелекту, що могли б забезпечити підтримку користувача у режимі реального часу;
- Модифікація та поєднання кривих Безьє з іншими типами кривих (наприклад, B-сплайнами, кривими Ерміта або Нюрнберга) для створення гібридних моделей з покращеними властивостями згладжування, локального контролю та стикування сегментів;

- Дослідження використання кривих Безьє в суміжних галузях, зокрема:
  1. у біомедичних візуалізаціях — для моделювання контурів органів, судин або траєкторій хірургічних інструментів;
  2. у робототехніці — для побудови плавних траєкторій руху маніпуляторів;
  3. у геоінформаційних системах — для генерації криволінійних маршрутів на основі геоданих;
  4. у розумних транспортних системах — для планування траєкторій автономного руху;
  5. у цифровій анімації — для автоматичного генерування міжкадрових рухів на основі ключових кадрів;
- Побудова навчальних та дослідницьких платформ для вивчення властивостей кривих Безьє студентами та дослідниками, з можливістю поєднання теоретичних пояснень із інтерактивними візуалізаціями.

Таким чином, криві Безьє в сучасному просторі є не лише важливим математичним концептом, але й фундаментом для подальшого розвитку численних сфер науки і техніки.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сидоренко Ю., Залевська О. Система деформаційного моделювання на основі кривих Безьє. Сучасні проблеми моделювання. 2021. № 20,. С. 176-183.
2. Сорокін М. С., Логвиненко Н. В. Алгоритм визначення поверхні 3D моделі на основі Кривої Безьє. Матеріали XX міжнародного форуму молоді "Молодь і індустрія 4.0 в XXI столітті". Харків: ДБТУ, 2024. С. 204 – 205.
3. Бідніченко О. Г. Аналіз властивостей кривих Безьє для геометричного моделювання обводів технічних об’єктів. Матеріали XIII міжнар. наук.-практ. конф. "Прикладна геометрія, інженерна графіка та об’єкти інтелектуальної власності". Київ : КПІ ім. І. Сікорського, 2024. Вип. 13. С. 41–43.
4. Ванін В., Вірченко Г., Яблонський П. До питання геометричного моделювання з використанням кривих Безьє. Прикладна геометрія та інженерна графіка. 2020. № 98. С. 29-34.
5. Демків І. І. Про властивості операторних поліномів типу Бернштейна, що наближають оператор Урисона. Український математичний журнал. 2004. № 9. С. 1172 – 1181.
6. Євсєєв О. С. Комп'ютерна анімація : навчальний посібник для студентів напряму підготовки 6.051501 "Видавничо-поліграфічна справа". Харків: ХНЕУ ім. С. Кузнеця, 2014. 152 с.
7. Історія виникнення та розвитку комп’ютерної графіки: веб-сайт. URL: https://phm.cuspu.edu.ua/nauka/naukovo-populiarni-publikatsii/891-istoriya-vynyknennya-ta-rozvytku-komp-yuternoyi-hrafiky.html (дата звернення 19.11.2024 р.).
8. Малювання за допомогою «Пера», «Олівця» або інструмента малювання кривих : веб-сайт. URL: https://helpx.adobe.com/ua/illustrator/using/drawing-pen-curvature-or-pencil.html (дата звернення 18.12.2024 р.).

9. Молоденков К. (2020). Побудова кубічної кривої Безьє за 4 точками. Курсова робота за спеціальністю 122 «Комп'ютерні науки та інформаційні технології». Київ: національний університет «Києво-Могилянська академія», 2020. 20 с.
10. Надопта Т. А. Моделювання профільних абрисів прототипу взуття з використанням кривих Безьє. ВІСНИК, 2007. С. 9-13.
11. Похваленна О. О. Проектування обводів машин і агрегатів, які працюють в рухомому середовищі: дипл. роб. на здобуття осв. ступ. бакалавр. Київ: 2020. 60 с.
12. Холодняк Ю., Гавриленко Є., Зінов'єва О. Розробка алгоритму моделювання кривих з заданими властивостями. Науковий вісник Таврійського державного агротехнологічного університету, Вип.13(1). 2023. 16 с.
13. A Primer on Bézier Curves : веб-сайт. URL: https://pomax.github.io/bezierinfo/ (дата звернення: 28.04.2025)
14. Bézier Curves - and the logic behind them. Richard Ekwonye : веб-сайт. URL: https://blog.richardekwonye.com/bezier-curves (дата звернення: 28.04.2025)
15. Bezier Curves Explained: Essential Concepts. Lenovo US : веб-сайт. URL: https://www.lenovo.com/us/en/glossary/bezier-curve/?orgRef=https%253A%252F%252Fwww.google.com%252F (дата звернення 09.03.2025 р.).
16. Bezier Curves in OpenGL. GeeksforGeeks : веб-сайт. URL: https://www.geeksforgeeks.org/bezier-curves-in-opengl/ (дата звернення: 28.04.2025)
17. Bezier Curves in OpenGL – With GLUT / Zerihun M. Medium : веб-сайт. URL: https://medium.com/@zmarkm2021/b%C3%A9zier-curves-in-opengl-with-glut-8b2d71163691 (дата звернення: 28.04.2025)
18. 13 Bézier Curve: History of Product Surfaces : веб-сайт. URL: https://medium.com/@wicar/bézier-curve-history-of-product-surfaces-1c4c8d08b98c (дата звернення 19.11.2024 р.).
19. ChatGPT : веб-сайт. URL: https://chatgpt.com (дата звернення 09.03.2025 р.).

20. Curve Line Fitting with Bezier Curves. OpenReview : веб-сайт. URL: https://openreview.net/forum?id=dxMffCAd4w (дата звернення: 20.04.2025)
21. 14 Differential Geometry of Curves : веб-сайт. URL: https://web.mit.edu/hyperbook/Patrikalakis-Maekawa-Cho/node21.html (дата звернення 01.12.2024 р.).
22. Defining a curas a Bezier curve. Journal of Taibah University for Science : веб-сайт. URL: https://www.researchgate.net/publication/332445061_Defining_a_curve_as_a_Bezier_curve (дата звернення: 20.04.2025).
23. Dynamic and Adaptive Path Planning Method Based on Bézier Curve for Intelligent Vehicles. MDPI Electronics : веб-сайт. URL: https://www.mdpi.com/2079-9292/14/3/494 (дата звернення: 08.05.2025)
24. 15 ECG production: 3D Animation : веб-сайт. URL: https://www.ecgprod.com/services/animation/3d-animation/ (дата звернення 19.11.2024 р.).
25. Generative art using Bézier Curves : веб-сайт. URL: https://robotenique.github.io/posts/bezier-curve-gen/ (дата звернення 09.03.2025 р.).
26. Google Colab : веб-сайт. URL: https://colab.research.google.com (дата звернення 09.03.2025 р.).
27. How to Generate Curves – A Fun Introduction to Bézier Curves / Devansh. Machine Learning Made Simple : веб-сайт. URL: https://machine-learning-made-simple.medium.com/how-to-generate-curves-a-fun-introduction-to-b%C3%A9zier-curves-e0651b0cce88 (дата звернення: 02.05.2025)
28. LearnOpenGL — сучасний посібник з OpenGL 3.3+ : веб-сайт. URL: https://learnopengl.com/ (дата звернення: 08.05.2025)
29. Nerding Out With Bezier Curves: веб-сайт. URL: https://medium.com/free-code-camp/nerding-out-with-bezier-curves-6e3c0bc48e2f (дата звернення: 08.05.2025).

30. Probabilistic Bézier Curves for Efficient Modeling of Continuous Stochastic Processes. AAAI Conference on Artificial Intelligence : веб-сайт. URL: https://cdn.aaai.org/ojs/6576/6576-13-9801-1-10-20200519.pdf (дата звернення: 28.05.2025)
31. Smooth Trajectory Generation for Multirotor UAVs using Improved Path Search Algorithm and Bezier Curves. MDPI Sensors : веб-сайт. URL: https://www.mdpi.com/1424-8220/21/7/2460 (дата звернення: 28.05.2025)
32. 16 The birth of Bézier curves and how it shaped graphic design : веб-сайт. URL: https://www.linearity.io/blog/bezier-curves/ (дата звернення 19.11.2024 р.).
33. The Industry's Foundation for High Performance Graphics (Open GL) : веб-сайт. URL: https://www.opengl.org (дата звернення 01.04.2025 р.).
34. Understanding easing and cubic-bezier curves in CSS : веб-сайт. URL: https://joshcollinsworth.com/blog/easing-curves (дата звернення 01.04.2025 р.).
35. 17 What is a Bezier Curve? : веб-сайт. URL: https://medium.com/data-science-shorts/what-is-a-bezier-curve-ba54935ef5b1 (дата звернення 01.11.2024 р.).

ДОДАТКИ

Додаток 1.

Код на базі Python 3 для апроксимації траєкторій кривими Безьє методом MLP

```
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt

# Функція побудови кривої Безьє
def bezier_curve(control_points, num_points=100):
    t = np.linspace(0, 1, num_points).reshape(-1, 1)
    P0, P1, P2, P3 = control_points
    curve = (
        (1 - t)**3 * P0 +
        3 * (1 - t)**2 * t * P1 +
        3 * (1 - t) * t**2 * P2 +
        t**3 * P3
    )
    return curve

# Нейронна мережа (MLP)
class BezierMLP(nn.Module):
    def __init__(self, input_dim=40, hidden_dim=128, output_dim=8):
        super(BezierMLP, self).__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, output_dim)
        )

    def forward(self, x):
        return self.model(x)

# Генерація випадкової кривої
def generate_sample_curve():
    control_points = np.random.rand(4, 2) * 2 - 1
    curve = bezier_curve(control_points, num_points=20)
    return curve, control_points

# Створення навчального датасету
def create_dataset(num_samples=1000):
    X, Y = [], []
    for _ in range(num_samples):
        curve, control_points = generate_sample_curve()
        X.append(curve.flatten())
        Y.append(control_points.flatten())
```

```
    return np.array(X, dtype=np.float32), np.array(Y, dtype=np.float32)

# Дані
X_data, Y_data = create_dataset(1000)
X_train = torch.tensor(X_data)
Y_train = torch.tensor(Y_data)

# Модель
model = BezierMLP()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Навчання
for epoch in range(200):
    idx = torch.randperm(X_train.size(0))
    for i in range(0, X_train.size(0), 32):
        batch_idx = idx[i:i+32]
        x_batch = X_train[batch_idx]
        y_batch = Y_train[batch_idx]

        optimizer.zero_grad()
        preds = model(x_batch)
        loss = criterion(preds, y_batch)
        loss.backward()
        optimizer.step()

# Тестування
test_curve, true_cp = generate_sample_curve()
test_input = torch.tensor(test_curve.flatten(), dtype=torch.float32).unsqueeze(0)
predicted_cp = model(test_input).detach().numpy().reshape(4, 2)
predicted_curve = bezier_curve(predicted_cp)

# Візуалізація
plt.figure(figsize=(6, 6))
plt.plot(test_curve[:, 0], test_curve[:, 1], label="Original Trajectory", linewidth=2)
plt.plot(predicted_curve[:, 0], predicted_curve[:, 1], '--', label="Predicted Bézier", linewidth=2)
plt.scatter(*zip(*true_cp), c='green', label='True Control Points')
plt.scatter(*zip(*predicted_cp), c='red', marker='x', label='Predicted Control Points')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.title("Апроксимація кривої Безьє за допомогою MLP")
plt.tight_layout()
plt.show()
```

Додаток 2.

Код на базі Python 3 для апроксимації траєкторій кривими Безьє методом LSTM

```
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt

# Функція для побудови кривої Безьє 3-го порядку
def bezier_curve(control_points, num_points=100):
    t = np.linspace(0, 1, num_points).reshape(-1, 1)
    P0, P1, P2, P3 = control_points
    curve = (
        (1 - t)**3 * P0 +
        3 * (1 - t)**2 * t * P1 +
        3 * (1 - t) * t**2 * P2 +
        t**3 * P3
    )
    return curve

# Генерація випадкової кривої
def generate_sample_curve():
    control_points = np.random.rand(4, 2) * 2 - 1
    curve = bezier_curve(control_points, num_points=20)
    return curve, control_points

# Створення датасету
def create_dataset(num_samples=1000):
    X, Y = [], []
    for _ in range(num_samples):
        curve, control_points = generate_sample_curve()
        X.append(curve)
        Y.append(control_points.flatten())
    return np.array(X, dtype=np.float32), np.array(Y, dtype=np.float32)

# Архітектура моделі на основі LSTM
class BezierLSTM(nn.Module):
    def __init__(self, input_size=2, hidden_size=64, num_layers=2, output_size=8):
        super(BezierLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = out[:, -1, :]  # беремо останній стан
        return self.fc(out)
```

```
# Підготовка даних
X_data, Y_data = create_dataset(1000)
X_train = torch.tensor(X_data)
Y_train = torch.tensor(Y_data)

# Ініціалізація моделі, функції втрат і оптимізатора
model = BezierLSTM()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Навчання
for epoch in range(20):  # можна більше для кращого результату
    idx = torch.randperm(X_train.size(0))
    for i in range(0, X_train.size(0), 32):
        batch_idx = idx[i:i+32]
        x_batch = X_train[batch_idx]
        y_batch = Y_train[batch_idx]

        preds = model(x_batch)
        loss = criterion(preds, y_batch)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    if epoch % 5 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")

# Тестування моделі
test_curve, true_cp = generate_sample_curve()
test_input = torch.tensor(test_curve, dtype=torch.float32).unsqueeze(0)
predicted_cp = model(test_input).detach().numpy().reshape(4, 2)
predicted_curve = bezier_curve(predicted_cp)

# Візуалізація
plt.figure(figsize=(6, 6))
plt.plot(test_curve[:, 0], test_curve[:, 1], label="Original Trajectory", linewidth=2)
plt.plot(predicted_curve[:, 0], predicted_curve[:, 1], '--', label="Predicted Bézier",
linewidth=2)
plt.scatter(*zip(*true_cp), c='green', label='True Control Points')
plt.scatter(*zip(*predicted_cp), c='red', marker='x', label='Predicted Control Points')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.title("Апроксимація кривої Безьє за допомогою LSTM")
plt.tight_layout()
plt.show()
```

Додаток 3.

Код побудови анімованої кубічної кривої Безьє в OpenGL

```
tonya bui, [05.03.2025 20:16]
#include <iostream>
#include <GL/glut.h>
#include <math.h>
//------------------КОНСТАНТИ ТА ГЛОБАЛЬНІ ЗМІННІ-------------------------
int Width = 800, Height = 800;
#define Pi 3.14159265358979323384
float animationParameter = 0.0; // Параметр для анімації
//-------------------Координатні осі-----------------------------------
void coordinate_axis(void)
{
   glLineWidth(2);
   glColor3d(1, 1, 1);
   glBegin(GL_LINES);
   glVertex3d(0, 0, 0); glVertex3d(5, 0, 0); //Ox
   glVertex3d(4.8, 0.1, 0); glVertex3d(5, 0, 0);
   glVertex3d(4.8, -0.1, 0); glVertex3d(5, 0, 0);
   glVertex3d(0, 0, 0); glVertex3d(0, 5, 0); //oy
   glVertex3d(0.1, 4.8, 0); glVertex3d(0, 5, 0);
   glVertex3d(-0.1, 4.8, 0); glVertex3d(0, 5, 0);
   glVertex3d(0, 0, 0); glVertex3d(0, 0, 5);//oz
   glVertex3d(-0.1, 0.1, 4.8); glVertex3d(0, 0, 5);
   glVertex3d(0.1, -0.1, 4.8); glVertex3d(0, 0, 5);
   glEnd();
}
// Функція для малювання кривої Безьє
void drawBezierCurve() {
   glDisable(GL_LIGHTING); // Вимкнення освітлення для кривої
   glColor3f(0.0, 1.0, 0.0); // Зелений колір
   glLineWidth(3.0);
   glBegin(GL_LINE_STRIP);
   for (float t = 0.0; t <= 1.0; t += 0.01) {
      // Контрольні точки кривої Безьє
      float p0x = -2.0, p0y = -2.0;
      float p1x = -1.0 + sin(animationParameter), p1y = 2.0;
      float p2x = 1.0 + cos(animationParameter), p2y = -2.0;
      float p3x = 2.0, p3y = 2.0;

      // Формула кривої Безьє третього порядку
      float x = pow(1 - t, 3) * p0x + 3 * t * pow(1 - t, 2) * p1x + 3 * pow(t, 2) * (1 - t) * p2x
+ pow(t, 3) * p3x;
      float y = pow(1 - t, 3) * p0y + 3 * t * pow(1 - t, 2) * p1y + 3 * pow(t, 2) * (1 - t) * p2y
+ pow(t, 3) * p3y;
      glVertex3f(x, y, 0.0);
```

```
    }
    glEnd();

    // Додаємо малювання контрольних точок червоним кольором
    glEnable(GL_POINT_SMOOTH); // Зробити точки гладкими
    glPointSize(10.0);
    glBegin(GL_POINTS);
    glColor3f(1.0, 0.0, 0.0); // Червоний колір
    float p0x = -2.0, p0y = -2.0;
    float p1x = -1.0 + sin(animationParameter), p1y = 2.0;
    float p2x = 1.0 + cos(animationParameter), p2y = -2.0;
    float p3x = 2.0, p3y = 2.0;

    glVertex3f(p0x, p0y, 0.0);
    glVertex3f(p1x, p1y, 0.0);
    glVertex3f(p2x, p2y, 0.0);
    glVertex3f(p3x, p3y, 0.0);
    glEnd();
    glDisable(GL_POINT_SMOOTH); // Вимкнути гладкі точки після використання

    glEnable(GL_LIGHTING); // Увімкнення освітлення назад
}
// Функція анімації
void update(int value) {
    animationParameter += 0.05;
    if (animationParameter > 2 * Pi) {
        animationParameter -= 2 * Pi;
    }
    glutPostRedisplay();
    glutTimerFunc(16, update, 0);
}
void keyboard(unsigned char key, int x0, int y0)
{
    if (key == 'x')
    {
        glRotated(10, 1, 0, 0);
        glutPostRedisplay();
    }
    if (key == 'y')
    {
        glRotated(10, 0, 1, 0);
        glutPostRedisplay();
    }
    if (key == 'z')
    {
        glRotated(10, 0, 0, 1);
        glutPostRedisplay();
    }
```

```
}
void display(void) {
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   //coordinate_axis();
   drawBezierCurve();
   glutSwapBuffers();
}
void scene(void)
{
   glClearColor(0, 0, 0, 0);
   GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat light_position[] = { 3.0, 3.0, -3.0, 1.0 };
   GLfloat light_dir[] = { 1.0,1.0,1.0,1.0 };
   glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
   glLightfv(GL_LIGHT0, GL_POSITION, light_position);
   glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, light_dir);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_DEPTH_TEST);
   glMatrixMode(GL_PROJECTION);
   glMatrixMode(GL_MODELVIEW);
   glOrtho(-5, 5, -5, 5, 2, 15);
   gluLookAt(0.0, 0.0, 5.0,
      0.0, 0.0, 0.0,
      0.0, 1.0, 0.);
}
void main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitWindowPosition(0, 0);
   glutInitWindowSize(Width, Height);
   glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGB);
   glutCreateWindow("Animated Bezier Curve");
   scene();
   glutDisplayFunc(display);

tonya bui, [05.03.2025 20:16]
glutKeyboardFunc(keyboard);
   glutTimerFunc(0, update, 0); // Додаємо таймер для анімації
   glutMainLoop();
}
```

Додаток 4.

Код побудови анімованої кривої Безьє п’ятого порядку в OpenGL

```
tonya bui, [05.03.2025 20:31]
#include <iostream>
#include <GL/glut.h>
#include <math.h>
//------------------КОНСТАНТІ ТА ГЛОБАЛЬНІ ЗМІННІ-----------------------
int Width = 800, Height = 800;
#define Pi 3.14159265358979323384
float animationParameter = 0.0; // Параметр для анімації
//-------------------Координатні осі-----------------------------------
void coordinate_axis(void)
{
   glLineWidth(2);
   glColor3d(1, 1, 1);
   glBegin(GL_LINES);
   glVertex3d(0, 0, 0); glVertex3d(5, 0, 0); //Ox
   glVertex3d(4.8, 0.1, 0); glVertex3d(5, 0, 0);
   glVertex3d(4.8, -0.1, 0); glVertex3d(5, 0, 0);
   glVertex3d(0, 0, 0); glVertex3d(0, 5, 0); //oy
   glVertex3d(0.1, 4.8, 0); glVertex3d(0, 5, 0);
   glVertex3d(-0.1, 4.8, 0); glVertex3d(0, 5, 0);
   glVertex3d(0, 0, 0); glVertex3d(0, 0, 5);//oz
   glVertex3d(-0.1, 0.1, 4.8); glVertex3d(0, 0, 5);
   glVertex3d(0.1, -0.1, 4.8); glVertex3d(0, 0, 5);
   glEnd();
}

void drawComplexBezierCurve() {
   glDisable(GL_LIGHTING);
   glColor3f(0.0, 1.0, 0.5);
   glLineWidth(3.0);
   glBegin(GL_LINE_STRIP);
   for (float t = 0.0; t <= 1.0; t += 0.01) {
      // Контрольні точки з більш складною траєкторією
      float p0x = -3.0, p0y = -3.0;
      float p1x = -2.0 + sin(animationParameter * 1.2),
          p1y = 2.0 + cos(animationParameter);
      float p2x = 1.0 + cos(animationParameter * 0.7),
          p2y = -2.0 + sin(animationParameter * 1.5);
      float p3x = 3.0 + sin(animationParameter * 0.5),
          p3y = 3.0 + cos(animationParameter * 0.8);
      float p4x = 0.0 + cos(animationParameter * 1.3),
          p4y = 1.0 + sin(animationParameter * 1.1);
      float p5x = 2.0, p5y = 2.0;
```

```
        // Розширена формула кривої Безьє п'ятого порядку
        float x = pow(1 - t, 5) * p0x +
                5 * t * pow(1 - t, 4) * p1x +
                10 * pow(t, 2) * pow(1 - t, 3) * p2x +
                10 * pow(t, 3) * pow(1 - t, 2) * p3x +
                5 * pow(t, 4) * (1 - t) * p4x +
                pow(t, 5) * p5x;
        float y = pow(1 - t, 5) * p0y +
                5 * t * pow(1 - t, 4) * p1y +
                10 * pow(t, 2) * pow(1 - t, 3) * p2y +
                10 * pow(t, 3) * pow(1 - t, 2) * p3y +
                5 * pow(t, 4) * (1 - t) * p4y +
                pow(t, 5) * p5y;
        glVertex3f(x, y, 0.0);
    }
    glEnd();

    // Малювання контрольних точок
    glEnable(GL_POINT_SMOOTH);
    glPointSize(10.0);
    glBegin(GL_POINTS);
    glColor3f(1.0, 0.0, 0.0);
    float p0x = -3.0, p0y = -3.0;
    float p1x = -2.0 + sin(animationParameter * 1.2),
        p1y = 2.0 + cos(animationParameter);
    float p2x = 1.0 + cos(animationParameter * 0.7),
        p2y = -2.0 + sin(animationParameter * 1.5);
    float p3x = 3.0 + sin(animationParameter * 0.5),
        p3y = 3.0 + cos(animationParameter * 0.8);
    float p4x = 0.0 + cos(animationParameter * 1.3),
        p4y = 1.0 + sin(animationParameter * 1.1);
    float p5x = 2.0, p5y = 2.0;

    glVertex3f(p0x, p0y, 0.0);
    glVertex3f(p1x, p1y, 0.0);
    glVertex3f(p2x, p2y, 0.0);
    glVertex3f(p3x, p3y, 0.0);
    glVertex3f(p4x, p4y, 0.0);
    glVertex3f(p5x, p5y, 0.0);
    glEnd();
    glDisable(GL_POINT_SMOOTH);
    glEnable(GL_LIGHTING);
}

// Функція анімації
void update(int value) {
    animationParameter += 0.05;
    if (animationParameter > 2 * Pi) {
```

```
        animationParameter -= 2 * Pi;
    }
    glutPostRedisplay();
    glutTimerFunc(16, update, 0);
}

void keyboard(unsigned char key, int x0, int y0)
{
    if (key == 'x')
    {
        glRotated(10, 1, 0, 0);
        glutPostRedisplay();
    }
    if (key == 'y')
    {
        glRotated(10, 0, 1, 0);
        glutPostRedisplay();
    }
    if (key == 'z')
    {
        glRotated(10, 0, 0, 1);
        glutPostRedisplay();
    }
}

void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //coordinate_axis();
    drawComplexBezierCurve();
    glutSwapBuffers();
}

void scene(void)
{
    glClearColor(0, 0, 0, 0);
    GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat

tonya bui, [05.03.2025 20:31]
light_position[] = { 3.0, 3.0, -3.0, 1.0 };
    GLfloat light_dir[] = { 1.0,1.0,1.0,1.0 };
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, light_dir);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_COLOR_MATERIAL);
    glEnable(GL_DEPTH_TEST);
```

```
    glMatrixMode(GL_PROJECTION);
    glMatrixMode(GL_MODELVIEW);
    glOrtho(-5, 5, -5, 5, 2, 15);
    gluLookAt(0.0, 0.0, 5.0,
        0.0, 0.0, 0.0,
        0.0, 1.0, 0.);
}

void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(0, 0);
    glutInitWindowSize(Width, Height);
    glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGB);
    glutCreateWindow("Animated Complex Bezier Curve");
    scene();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutTimerFunc(0, update, 0); // Додаємо таймер для анімації
    glutMainLoop();
}
```