ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра механіки, автоматизації та інформаційних технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти « бакалавр »

« Комп'ютерне моделювання складного руху матеріальної точки »
(тема кваліфікаційної роботи українською мовою)
« Computer modeling of the complex motion of a material point »
(тема кваліфікаційної роботи англійською мовою)

Виконала: студентка денної форми навчання спеціальності 126 – Інформаційні системи та технології
(код, назва спеціальності)
Освітня програма _____
(назва)
Сімченко Каріна В'ячеславівна
(прізвище, ім'я, по-батькові здобувача)

Керівник Палій К.С. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)
Рецензент Косирева Л.А.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 2023 р.

Завідувач(ка) кафедри
_____ Алла Рачинська
(підпис) (прізвище,ім'я)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.2023 р.

Оцінка_____/____/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ _____
(підпис) (прізвище, ім'я)

Одеса 2023

АНОТАЦІЯ

У дипломній роботі розробляється тема «Комп’ютерне моделювання складного руху матеріальної точки».

Мета роботи – створення математичної моделі руху матеріальної точки з використанням відповідних фізичних законів і рівнянь та реалізація комп'ютерної моделі руху за допомогою програмного забезпечення або використання спеціалізованих симуляційних середовищ, які дозволяють візуалізувати та аналізувати рух матеріальних точок.

В результаті проведених в роботі досліджень та аналізу отриманих результатів моделювання, порівняння їх з теоретичними очікуваннями і спостереженнями з’являється можливість оцінити точність та адекватність моделі складного руху матеріальної точки. Рух матеріальної точки є простою моделлю, що дозволяє досліджувати різні аспекти системи без урахування її внутрішньої структури. Отримані результати можуть бути застосовані для вирішення конкретних задач або удосконалення систем керування рухом роботів або інших технічних систем.

ABSTRACT

The topic "Computer modeling of complex motion of a material point" is being developed in the diploma thesis.

The objective of the work is to create a mathematical model of the motion of a material point using relevant physical laws and equations, and to implement a computer model of the motion using software or specialized simulation environments that allow for visualization and analysis of the motion of material points.

By conducting research and analyzing the obtained results of the modeling, comparing them with theoretical expectations and observations, it becomes possible to assess the accuracy and adequacy of the model of complex motion of a material point. The motion of a material point serves as a simplified model that allows for the investigation of various aspects of a system without considering its internal structure. The obtained results can be applied to solve specific tasks or improve motion control systems for robots or other technical systems.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API (ApplicationProgrammingInterface) - це набір правил та протоколів, які визначають, як різні компоненти програмного забезпечення можуть взаємодіяти між собою.

GDI+ (Graphics Device Interface Plus) - це програмний інтерфейс (API) графічного пристрою, розроблений компанією Microsoft.

HTML (HyperText Markup Language) - це мова розмітки, що використовується для створення та структурування веб-сторінок.

MSDN (Microsoft Developer Network) є набором ресурсів, які надаються компанією Microsoft для розробників програмного забезпечення.

PC (Personal Computer) - це персональний комп'ютер, загальноприйнятий термін.

RTF (Rich Text Format) - це формат файлу, який використовується для зберігання та обміну документами, що містять форматований текст.

VS (Visual Studio) - це інтегроване середовище розробки компанії Microsoft.

ВСТУП

В 20-му столітті були зроблені перші кроки в робототехніці, коли виникли перші механічні пристрої, що нагадували людські рухи. Однак, справжній прорив відбувся у другій половині 20-го століття з появою перших електронних комп'ютерів і мікропроцесорів. Це відкрило нові можливості для створення автономних роботів з програмованими функціями.

В кібернетиці та конструюванні роботів зазвичай використовуються принципи механіки, електроніки та програмування. Коливання матеріальної точки в рухомій системі відліку є одним із аспектів дослідження руху, який може бути використаний в робототехніці для реалізації рухів та функцій у роботах. Наприклад, коливання можуть використовуватись для стабілізації руху робота або для створення плавних та прецизійних рухів. Відомі приклади використання коливань в робототехніці [1] включають використання підсилюючих контролерів для стабілізації руху, використання регуляторів зворотного зв'язку для контролю коливань робота та використання коливань для імітації природних рухів, наприклад, у роботах-ходулях або роботах-плавальниках.

У теоретичній механіці, як відомо, існують дві основні моделі для вивчення руху - модель матеріальної точки і модель системи матеріальних точок [2, 3]. З розвитком сучасних автоматизованих систем велике значення набуває візуалізація складного руху матеріальної точки за допомогою сучасних програмних засобів. Для цього можна використовувати об'єктно-орієнтовану мову програмування C# [4-7]. Ця мова розробки в середовищі Microsoft Visual Studio дозволяє побудувати 3D-модель за допомогою бібліотеки DirectX 9.0 [8].

У даній роботі розглядається випадок коливання матеріальної точки в рухомій системі відліку. Рухома система відліку моделюється як рамна конструкція, яка може здійснювати обертальний рух з постійною швидкістю

навколо нерухомої вісі. Матеріальна точка рухається вздовж бічного ребра рамної конструкції згідно заданого закону відносного руху.

Для ілюстрації складного руху матеріальної точки в тривимірному просторі необхідно побудувати вектори швидкостей та прискорення у відносному та абсолютному рухах. Усі вектори повинні бути представлені для даного моменту часу, щоб відображати поточний стан руху.

Комп'ютерне моделювання абсолютного руху матеріальної точки в тривимірному просторі включає в себе не тільки моделювання самого руху, а і побудову траєкторії абсолютного руху та годографів векторів відносного та переносного рухів.

Дослідження та моделювання коливань матеріальних точок у рухомих системах відліку допомагають розробляти нові алгоритми та стратегії управління рухом роботів. Це покращує стабільність, точність та безпеку роботів під час виконання різних завдань.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Аналізуючи предметну область комп'ютерного моделювання руху матеріальної точки, можна зазначити наступні ключові аспекти:

1. Моделювання руху: Комп'ютерне моделювання дозволяє створювати віртуальні моделі руху матеріальних точок, відтворюючи їх поведінку на основі фізичних законів та рівнянь. Це дозволяє досліджувати та прогнозувати рух матеріальних точок у різних умовах та отримувати числові результати.
2. Віртуальні середовища та програмне забезпечення: Для комп'ютерного моделювання руху матеріальної точки використовуються спеціалізовані програмні засоби та віртуальні середовища. Це можуть бути програми для чисельного розв'язання рівнянь руху, середовища для візуалізації та аналізу результатів моделювання, а також бібліотеки або фреймворки для реалізації комп'ютерних моделей.
3. Валідація та порівняння з експериментом: Важливим етапом комп'ютерного моделювання руху матеріальної точки є валідація результатів шляхом порівняння з експериментальними даними. Це дозволяє перевірити адекватність та точність моделі, а також підтвердити правильність використовуваних фізичних законів.
4. Застосування в робототехніці: Комп'ютерне моделювання руху матеріальної точки має практичне застосування в робототехніці. Воно дозволяє вивчати та оптимізувати рухові характеристики роботів, визначати траєкторії руху, аналізувати вплив зовнішніх факторів на рух роботів та розробляти керуючі алгоритми. Основні пункти аналізу застосування додатку в робототехніці можуть включати наступне:
    1) Математичні моделі руху: Вивчення математичних моделей, що описують рух матеріальної точки, таких як рух по прямій лінії, криволінійний рух, кінематика та динаміка руху. Дослідження різних типів руху, їх властивостей та обмежень.

2) Контроль та планування руху: Аналіз методів та алгоритмів контролю та планування руху матеріальних точок в робототехніці. Вивчення проблеми траєкторійного планування, визначення оптимальних шляхів руху, уникнення перешкод та досягнення заданих цілей.
3) Стабілізація руху: Дослідження методів стабілізації руху матеріальних точок для забезпечення точності та стійкості роботів. Вивчення різних контролерів та систем стабілізації, таких як PID-регулятори, адаптивні системи та нейромережеві контролери.
4) Навігація та відслідковування: Аналіз методів навігації та відслідковування руху матеріальних точок в робототехніці. Вивчення систем відслідковування об'єктів, включаючи використання датчиків, комп'ютерного зору та інших сенсорів для визначення положення та орієнтації.
5) Використання коливань: Розгляд використання коливань у робототехніці, які можуть бути застосовані для стабілізації руху, адаптації до змінних умов, зниження впливу зовнішніх збурень та інших цілей. Аналіз методів моделювання та контролю коливань для досягнення потрібних результатів.

В цілому, комп'ютерне моделювання руху матеріальної точки є важливим інструментом для дослідження та оптимізації рухових процесів в робототехніці. Воно дозволяє прогнозувати та вивчати різні аспекти руху матеріальних точок, що має практичне значення для розробки та удосконалення робототехнічних систем, а також розуміти основні концепції, проблеми та рішення, пов'язані з рухом роботів. Він сприяє розробці нових алгоритмів та технологій, які дозволяють роботам виконувати складні завдання з надійністю та ефективністю.

1.1 Визначення користувачів програмного продукту та їх задач

Користувачі програмного додатку «Комп'ютерне моделювання складного руху матеріальної точки» можуть бути різного профілю і зацікавленості. Основні категорії користувачів включають:

1. Студенти та дослідники: Даний додаток може бути корисним для студентів технічних спеціальностей, які вивчають механіку, фізику та робототехніку. Він допоможе їм краще зрозуміти принципи складного руху матеріальної точки, виконувати комп'ютерне моделювання та аналізувати результати, а також може бути корисним для виконання практичних завдань та лабораторних робіт. Дослідники також можуть використовувати додаток для проведення власних досліджень у галузі руху матеріальних точок.
    1. Викладачі та навчальні заклади: Для викладачів додаток може стати цінним інструментом для навчання та викладання. Вони можуть використовувати його для демонстрації складних рухів матеріальних точок на заняттях, проведення практичних лабораторних робіт та оцінювання робіт студентів. Навчальні заклади можуть включати у програму навчання додаток «Комп'ютерне моделювання складного руху матеріальної точки» для підготовки студентів до реальних задач робототехніки.
    2. Інженери та дослідницькі організації: Інженери, які працюють у галузі робототехніки, можуть використовувати додаток для моделювання та валідації руху матеріальних точок у своїх проектах. Він може допомогти їм покращити системи керування, розробити нові алгоритми та перевірити їх ефективність перед фізичною реалізацією. Вони можуть використовувати його для проектування та віртуального тестування руху роботів, вирішення задач керування рухом та оптимізації робототехнічних систем.

3. Хобісти та ентузіасти: Додаток також може зацікавити людей, які просто цікавляться робототехнікою та механікою. Вони можуть використовувати його для вивчення основних принципів руху матеріальних точок, експериментування з різними параметрами руху та візуалізації результатів.

Звичайно, користувачів може бути багато і залежатиме від конкретного контексту використання програмного додатку. Проте, ці групи користувачів можуть бути основними споживачами та зацікавленими сторонами даного додатку.

1.2 Огляд конкурентних програмних продуктів та визначення переваг

Враховуючи специфічність розробленого додатку, можна надати загальний огляд кількох типових програмних засобів, які широко використовуються для комп'ютерного моделювання руху матеріальних точок в робототехніці.

1. MATLAB/Simulink: MATLAB є популярним програмним засобом для наукових обчислень і моделювання, а Simulink є його модулем для моделювання динамічних систем. Вони надають багато інструментів для моделювання руху матеріальних точок, включаючи чисельне розв'язання диференціальних рівнянь, візуалізацію результатів та можливості аналізу системи.
2. Gazebo: Gazebo є відкритим симулятором для роботів та рухомих систем. Він надає можливості моделювання руху матеріальних точок у 3D-середовищі, візуалізацію та інтеграцію з іншими програмними інструментами для робототехніки.
3. CoppeliaSim: CoppeliaSim (раніше відомий як V-REP) є інтегрованим середовищем для симуляції роботів та автономних систем. Воно надає засоби для моделювання руху матеріальних точок, створення складних сценаріїв та візуалізацію результатів.

4. ANSYS: ANSYS є потужним програмним засобом для чисельного моделювання та симуляції фізичних процесів. Він має модулі для моделювання руху матеріальних точок, враховуючи фізичні закони та взаємодію з оточенням.

Ці програмні продукти можуть мати різні функціональні можливості, і вибір залежатиме від конкретних потреб, бюджету та вимог дослідження чи проекту. Рекомендується провести детальне дослідження кожного з них, ознайомитися з їх можливостями та порівняти їх за потрібними критеріями перед прийняттям рішення. Однак, додаток «Комп'ютерне моделювання руху матеріальної точки» в порівнянні з конкурентними програмними продуктами має наступні переваги:

1. Спеціалізація: Додаток «Комп'ютерне моделювання руху матеріальної точки» спеціально розроблений для моделювання руху матеріальних точок, що дозволяє зосередитися на цій конкретній предметній області. Це може призвести до більш високої якості та ефективності моделювання.
2. Простота використання: Додаток має інтуїтивний і зручний інтерфейс, що полегшує роботу з ним. Це особливо важливо для користувачів без попереднього досвіду у моделюванні руху матеріальних точок.
3. Розширені можливості: Додаток може надавати додаткові функції розрахунків швидкості та прискорення матеріальної точки, будувати їх відповідні годографи з траєкторіями та траєкторією самої точки, які покращують процес моделювання, та реалізовувати візуалізацію результатів у зручному форматі.
4. Функціональність та налаштування: Додаток може надавати широкі можливості для налаштування параметрів моделі. Це дозволяє більш гнучко адаптувати модель до конкретних потреб дослідження або проекту.

Враховуючи ці переваги, додаток «Комп'ютерне моделювання руху матеріальної точки» може бути вигідним вибором для тих, хто шукає програмний продукт, спеціалізований на моделюванні руху матеріальних точок і з високою якістю, простотою використання та розширеними можливостями.

2 АНАЛІТИЧНІ РОЗРАХУНКИ ТА ПРОЕКТУВАННЯ

Аналітичні розрахунки та проектування додатку "Комп'ютерне моделювання руху матеріальної точки" включають такі етапи, як:

- аналіз вимог користувачів до додатку та визначення функціональних і не функціональних вимог, обмежень, необхідних функцій та можливостей додатку;
- проектування архітектури, де визначається загальна структура додатку та його компоненти, розробляються схеми класів та інші моделі, які описують взаємодію між різними частинами додатку;
- розробка інтерфейсу користувача – планується та проектується інтерфейс користувача, що забезпечує зручну та інтуїтивну взаємодію з додатком;
- розробка алгоритмів та математичних моделей, що описують рух матеріальної точки. Застосовуються відповідні фізичні закони, рівняння та методи моделювання.

2.1. Механіка складного руху.

Керуючись основними вимогами користувача, в якості принципу роботи додатку було враховано нижченаведені теоретичні відомості з курсу Теоретичної механіки.

Нехай точка рухається відносно системи відліку *Oxyz*, яка в свою чергу пересувається відносно до (нерухомої) системи $\Omega\xi\eta\varsigma$. Тоді рух, швидкість та прискорення точки, які розглядаються відносно системи *Oxyz*, називаються відносними, а відносно системи $\Omega\xi\eta\varsigma$ — абсолютними. Рух системи *Oxyz* відносно нерухомої $\Omega\xi\eta\varsigma$ є для рухаючоїсь точки переносним рухом, а швидкість та прискорення тієї незмінно зв'язаної з рухомою системою відліку точки простору, в якій у даний момент знаходиться точка, що рухається, називаються переносними.

Для розв'язку ряду задач механіки необхідно встановити залежність між швидкостями та прискореннями точки відносно до рухомої та нерухомої систем відліку. Для швидкостей ця залежність вже була знайдена геометрично (§6, п.15) [3] та має вид:

$$\overrightarrow{v_a} = \overrightarrow{v_r} + \overrightarrow{v_e} . \qquad (2.1.1)$$

Відповідну залежність для прискорень отримаємо, використовуючи поняття похідної вектора.

Нехай рухома *Oxyz* та нерухома $\Omega\xi\eta\varsigma$ системи відліку мають спільний початок О, та нехай ω – миттєва кутова швидкість рухомої системи *Oxyz* відносно нерухомої (Рис. 1). Розглянемо точку М, здійснюючу рух, що не залежить від руху тріедра *Oxyz*. Її радіус-вектор $r = OM$ буде зі зміною часу змінюватися у кожній із систем відліку по різним законам. Тоді за деякий відрізок часу Δt вектор *r* отримає відносно до вісей $\Omega\xi\eta\varsigma$ та *Oxyz* різні прирости, які ми позначимо як Δr та Δř відповідно.

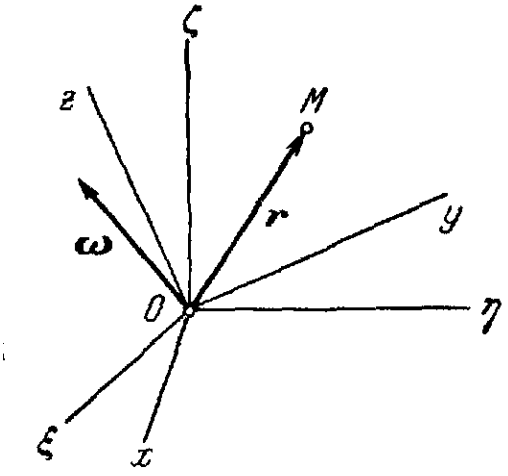

Рисунок 1

Границя відношень Δr та Δř до Δt→0 дає відповідні похідні:

$$\frac{dr}{dt} = lim_{\Delta t \to 0} \frac{\Delta r}{\Delta t} \qquad \text{та} \qquad \frac{\check{d}r}{dt} = \lim_{n \to \infty} \frac{\Delta \check{r}}{\Delta t} . \qquad (2.1.2)$$

Похідну $\frac{dr}{dt}$ назвемо «абсолютною» або «повною», а похідну $\frac{\tilde{d}r}{dt}$ – «відносною» або «локальною». Знайдемо залежність між цими похідними. З цією метою звернемося до їхньої кінематичної суті. Користуючись визначеннями відносної v_r та абсолютної v_a швидкостей [2], можна записати, що:

$$\frac{dr}{dt} = v_a\,, \qquad \frac{\tilde{d}r}{dt} = v_r\,. \tag{2.1.3}$$

Але, згідно (2.1.1), де переносна швидкість v_e є швидкість тієї незмінно зв'язаної з тріедром *Oxyz* точки простору, в якій у наданий момент знаходиться точка М. Тоді за формулою Ейлера $v_e = \omega \times r$ [3] та враховуючи вираз (2.1.1), маємо: $\frac{dr}{dt} = \frac{\tilde{d}r}{dt} + \omega \times \mathrm{r}.$

Так як будь-який вектор $a = a(t)$, заданий як нерозривна та диференційована функція часу, можна розглядати як радіус-вектор деякої точки (кінця цього вектора), то повна та локальна похідні будь-якого вектора $a(t)$ будуть зв'язані тим же співвідношенням:

$$\frac{da}{dt} = \frac{\tilde{d}a}{dt} + \omega \times a\,. \tag{2.1.4}$$

Якщо скористатися проекціями, то локальна похідна будь-якого вектора a відносно системи *Oxyz* може бути визначена як вектор, проекції якого на вісі координат цієї системи дорівнюють похідним від проекцій вектора a на ті самі вісі.

Треба відмітити. Що формула (2.1.4) зберігає свій вид і в тому випадку, коли *Oxyz*, окрім обертання навколо точки *O*, здійснює ще й поступовий рух, тобто пересувається як вільне тіло.

Розглянемо часні випадки.

Якщо система *Oxyz* нерухома, тоді: $\omega = 0$, $\frac{da}{dt} = \frac{\mathring{d}a}{dt}$.

Якщо вектор a нерухомий відносно до основної системи $\Omega\xi\eta\varsigma$, тоді: $\frac{da}{dt} = 0$, $\frac{\mathring{d}a}{dt} = -\omega \times a$.

Якщо вектор a незмінно зв'язаний з тріедром *Oxyz*, тоді: $\frac{\mathring{d}a}{dt} = 0$, $\frac{da}{dt} = \omega \times a$, тобто, як і повинно бути, швидкість кінця вектора a визначається в цьому випадку як і швидкість точки твердого тіла, скріпленого з тріедром *Oxyz*.

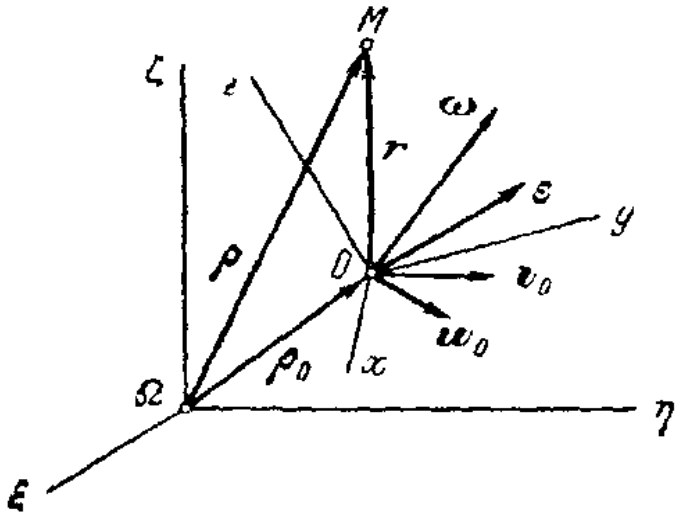

Рисунок 2

Тепер нехай рухома система *Oxyz* рухається відносно нерухомої $\Omega\xi\eta\varsigma$ як вільне тверде тіло. Позначимо швидкість та прискорення початку (полюса) *O* відносно до вісей $\Omega\xi\eta\varsigma$ через v_0 та w_0, а миттєву кутову швидкість та кутове прискорення самого тригранника *Oxyz* відносно до тих же вісей $\Omega\xi\eta\varsigma$ через ω та ε (Рис. 2). Розглянемо точку M, здійснюючу рух, який зовсім не залежить від руху системи Oxyz. Позначимо як ρ та r її абсолютний та відносний радіус-вектори, а через ρ_0 радіус-вектор точки *O*. Тоді в будь-який момент часу: $\rho = \rho_0 + r$. Візьмемо від обох частин цієї рівності повну похідну за часом. Враховуючи формули (2.1.3) і (2.1.4), будемо мати:

$$v_a = \frac{d\rho}{dt} = \frac{d\rho_o}{dt} + \frac{dr}{dt} = v_0 + \frac{\mathring{d}r}{dt} + \omega \times r + v_r \qquad (2.1.5)$$

Але за формулою (1) §12 [3] $v_0 + \omega \times r$ є швидкість тієї незмінно зв'язаної з системою *Oxyz* точки, в якій у даний момент часу знаходиться точка M, отже, за означенням це – переносна швидкість, тобто:

$$v_0 + \omega \times r = v_e \tag{2.1.6}$$

Із (2.1.5) знаходимо:

$$v_a = v_r + v_e \tag{2.1.7}$$

Беручи тепер похідні від обох частин рівності (2.1.7), отримаємо: $w_a = \frac{dv_a}{dt} = \frac{dv_r}{dt} + \frac{dv_e}{dt}$, або, після заміни v_e його значенням із (2.1.6):

$$w_a = \frac{d}{dt}(v_0 + \omega \times r) + \frac{dv_r}{dt} = \frac{dv_0}{dt} + \frac{d\omega}{dt} \times r + \omega \times \frac{dr}{dt} + \frac{dv_r}{dt}.$$

Застосовуючи формулу (2.1.4) до r та v_r, отримаємо:

$$w = \frac{dv_0}{dt} + \frac{d\omega}{dt} \times r + \omega \times \left(\frac{\tilde{d}r}{dt} + \omega \times r\right) + \frac{\tilde{d}v_r}{dt} + \omega \times v_r.$$

В цьому виразі

$$\frac{dv_0}{dt} = w_0, \qquad \frac{d\omega}{dt} = \varepsilon, \qquad \frac{\tilde{d}r}{dt} = v_r,$$

і його можна представити у вигляді:

$$w = w_0 + \varepsilon \times r + \omega \times (\omega \times r) + 2(\omega \times v_r)$$

або, після приведення:

$$w = \frac{\tilde{d}v_r}{dt} + w_0 + \varepsilon \times r + \omega \times (\omega \times r) + 2(\omega \times v_r). \tag{2.1.8}$$

Розглянемо, що являють собою складові правої частини рівності (2.1.8). Величина

$$w_r = \frac{\tilde{d}v_r}{dt} = \frac{\tilde{d}^2 r}{dt^2} \tag{2.1.9}$$

є за визначенням відносне прискорення (як локальна похідна від відносної швидкості за часом). Іншим шляхом у цьому можна впевнитися, поклавши у рівності (2.1.8) $\omega = 0,\ \varepsilon = 0,\ w_0 = 0$, тобто вважаючи вісі *Oxyz* нерухомими. Тоді повне прискорення точки М повинно співпасти з відносним та ми прийдемо до рівності (2.1.9).

Величина

$$w_e = w_0 + \varepsilon \times r + \omega \times (\omega \times r) \tag{2.1.10}$$

є переносне прискорення, так як, згідно з формулою (6) §12[3], вона дорівнює прискоренню тієї незмінно зв’язаної з системою *Oxyz* точки, в якій у даний

момент часу знаходиться точка М. Іншим шляхом це можна отримати, поклавши у рівності (2.1.8) $v_r = 0$, $w_r = 0$, тобто вважаючи, що точка М незмінно зв'язана з системою *Oxyz*. Тоді її повне прискорення співпадає з переносним і ми отримаємо рівність (2.1.10).

Величина

$$w_c = 2(\omega \times v_r), \tag{2.1.11}$$

яка не входить ні у переносне, ні у відносне прискорення, називається обертовим або кориолісовим прискоренням.

В результаті отримаємо наступну теорему про складання прискорень або теорему Кориоліса: абсолютне прискорення точки у складному русі дорівнює геометричній сумі відносного,переносного та кориолісового прискорень $\overrightarrow{w_a} = \overrightarrow{w_r} + \overrightarrow{w_e} + \overrightarrow{w_c}$. (2.1.12)

Якщо переносний рух (рух рухомої системи *Oxyz*) є поступовим, то $w_c = 0$, так як $\omega = 0$, і ми маємо:

$$\overrightarrow{w_a} = \overrightarrow{w_r} + \overrightarrow{w_e}. \tag{2.1.13}$$

Отже, при поступовому переносному русі абсолютне прискорення точки дорівнює геометричній сумі відносного та переносного прискорень.

Кориолісове прискорення з'являється тільки тоді, коли рухомі вісі при своєму русі обертаються (звідси термін «обертове» прискорення). Вектор w_c є сумою двох векторів $\omega \times v_r$. Один з них враховує зміну вектора відносної швидкості v_r у не поступовому переносному русі, а інший – зміну переносної швидкості v_e у відносному пересуванні точки (при зміні вектора r у відносному русі).

Якщо рухома система відліку рухається поступово, рівномірно та прямолінійно, то $\omega = 0$, $\varepsilon = 0$, $w_0 = 0$ та, як можна побачити із (2.1.10) та (2.1.11), $w_e = 0$ та $w_c = 0$, тобто в цьому випадку відносне та абсолютне прискорення співпадають. Треба ще відмітити, що кориолісове прискорення

може дорівнювати нулеві в даний момент часу, якщо в цей момент часу $\omega = 0$, або $v_r = 0$, чи $v_r \parallel \omega$.

У тих випадках, коли $w_c \neq 0$, його модуль, згідно (2.1.11), обчислюється за формулою: $w_c = 2\omega v_r \sin(\widehat{\omega, v_r})$, а напрямок визначається як напрямок векторного множення $\omega \times v_r$. Напрямок w_c можна ще знайти, спроектувавши вектор v_r на площину П, перпендикулярну до ω, та повернувши цю проекцію на 90^0 у бік переносного обертання, що можна побачити з рисунку 3. Такий спосіб зручно використовувати у разі плаского руху, коли v_r вже лежить у площині, перпендикулярній до ω.

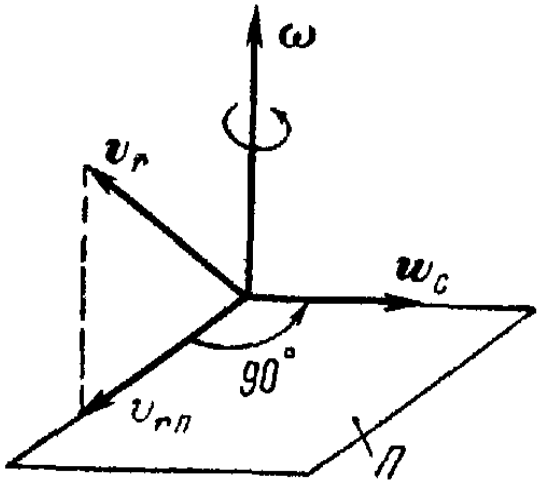

Рисунок 3

Задача формулюється наступним чином: Прямокутник *ABCD* обертається навколо сторони CD з кутовою швидкістю $\omega = const$ (Рис. 4). Вздовж *AB* рухається точка *M* за законом

$$\xi = a \sin(b\, t). \qquad (2.1.14)$$

Задано розмір: *DA=CB=a*. Визначити абсолютну швидкість та абсолютне прискорення точки у заданий момент часу [9].

Під час аналітичного розв'язку задачі було встановлено, що у відносному русі точка *M* рухається по прямій *AB*. У переносному – разом з віссю *AB* та при цьому описує окружність. Положення точки *M* на вісі *AB* визначається за законом (2.1.14) у заданий момент часу.

Абсолютну швидкість можна порахувати за допомогою теореми про складання швидкостей (2.1.1), де v_r – відносна швидкість, v_e – переносна, які, в свою чергу, можуть бути знайдені за відповідними формулами:

$$v_r = \frac{d\xi}{dt} = a\ b\ \cos(b\ t).$$

У переносному русі точка M рухається за окружністю навколо вертикальної вісі CD, тому: $v_e = R\omega$, де R – радіус окружності, що описує точка M, обертаючись навколо вісі CD. $R = CB = a$, ω – кутова швидкість обертання прямокутника ABCD.

Абсолютне прискорення точки можна знайти, скориставшись формулою (2.1.12).

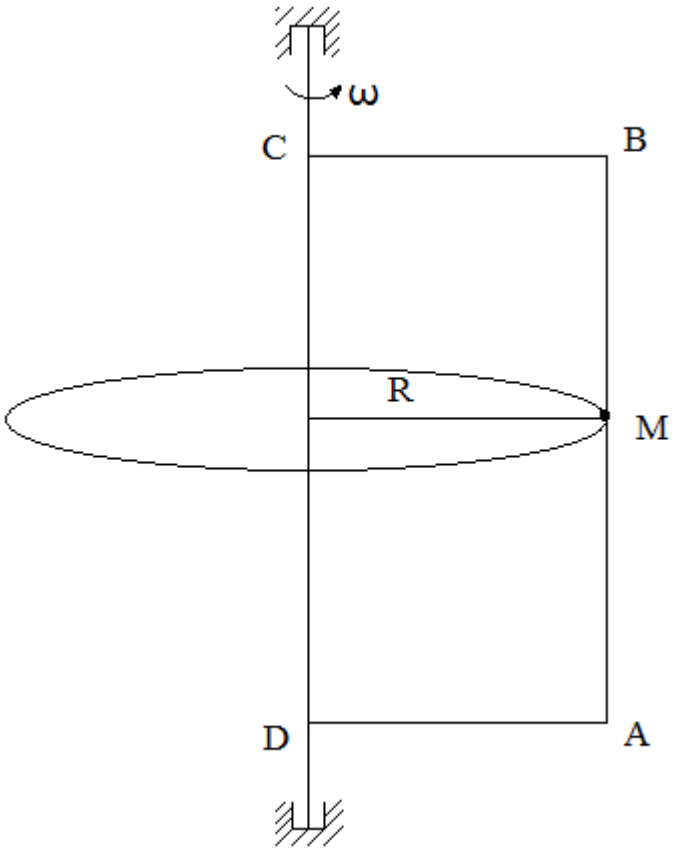

Рисунок 4

2.2 Графічні можливості мови C#, що було використано для розробки Windows-додатка

Мова **C#** на платформі **Windows** використовує графічний інтерфейс **GDI+** За допомогою **GDI+** ми можемо візуалізовувати графічні образи у формах **Windows** та елементах керування.

Комп'ютерне проектування й робота із тривимірними об'єктами практикується вже протягом декількох десятиліть, однак лише останнім часом, коли навіть базові моделі домашніх комп'ютерів стали досить потужними, був досягнутий справжній прогрес в **3D**-Графіці.

3D-Об'єкт — це об'ємне тіло, що володіє довжиною, шириною й глибиною, та насправді **3D**-Об'єкти існують тільки в пам'яті комп'ютера. Монітор лише відображає їх на пласкій поверхні екрана. Тривимірний об'єкт характеризується властивою йому формою й текстурою поверхні. Форма — це геометрія об'єкта, яка звичайно описується серією взаємозалежних точок (вершин) у тривимірному просторі й багатокутників (граней — замкнених двомірних фігур із трьома або більш сторонами). Причому для простоти в більшості випадків як базових багатокутників вибираються трикутники й з'єднуються один з одним по кілька сотень або навіть тисяч, утворюючи складні тривимірні сітки. Текстура поверхні — це простий опис властивостей поверхні подібних багатокутників: їх кольору, прозорості і так далі.

Щоб відобразити **3D**-Об'єкт на екрані, комп'ютеру доводиться виконувати деякі серйозні математичні розрахунки [8]. Спочатку з'ясовується, які частини об'єкта будуть бачені із заданої точки огляду. Крім того, об'єкт не буде виглядати об'ємним, якщо в зображенні не буде позначена перспектива. Тому комп'ютер, використовуючи інформацію про глибину, коректує зображення таким чином, щоб створити на двомірному екрані ілюзію глибини. Потім, враховуючи інформацію про текстуру, з'ясовується, як об'єкт взаємодіє зі світлом: якого кольору він повинен бути, яка кількість світла повинна відбиватися від кожної його грані, чи повинні виникати які-небудь тіні і так далі.

Відображення тривимірних об'єктів на плоскому екрані — настільки складна задача, що основні комп'ютерні компанії стали вбудовувати **3D**-функції прямо в операційну систему, щоб програмістам не доводилося писати їх самостійно. Це робиться за допомогою **API**, що спрощують роботу

з **3D**-Об'єктами і їх перемальовування на екрані [4, 8]. На **PC** «рідним» **API** є **Direct 3D**, який (як частина **Directx**) поставляється в комплекті з операційною системою **Windows 95/98.**

Однак крім цього існують і сотні інших **3D API**, орієнтованих на виконання якого-небудь певного кола задач. Для тривимірних ігор найважливішим фактором є швидкість: щоб гра була цікавої, необхідно, щоб об'єкти на екрані перемальовувалися якнайшвидше. У задачах же тривимірного моделювання, навпаки, важливо, щоб об'єкти перемальовувалися як можна точніше. Тому в таких програмах не варто через кожні кілька хвилин запускати рендеринг, оскільки це дуже тривала процедура. Для прискорення відображення **3D**-Графіки на екрані зображення перемальовуються на згадку на «віртуальні екрани» — фрейм-буфери — аналогічно знімкам екранів у пам'яті комп'ютера. В одному із фреймів може бути записана базова форма об'єкта, в іншому — накладена на нього текстура, в інших — висвітлення, прозорість, альфа-канал (простіше говорячи — маска) і інші ефекти [4, 5]. Таким чином, перед видачею об'єкта на екран комп'ютер може за один повний прохід скомпілювати всі необхідні графічні ефекти й згенерувати більш плавні анімації. Зворотний бік медалі — колосальні об'єми, «що з'їдаються» оперативною пам'яттю, причому чим вище дозвіл вашого екрана, тим більше пам'яті потрібно на такі **Ram**-Буфери.

Microsoft Directx — це набір програмних інтерфейсів, використовуваних для керування низкою апаратних компонентів комп'ютера. До складу цих інтерфейсів входять: **Directdraw** — для швидкого доступу до відеопам'яті, **Directsound** — для висновку аудіоінформації на звукову карту, **Directplay** — для організації багатокористувацької роботи через модем, локальну мережу або **Internet**, **Directinput** — для обробки введення інформації із клавіатури, від миші або джойстика й **Direct3D** — ядро підтримки тривимірної графіки, використовуване разом з **Directdraw.**

Для розроблювачів **Directx** являє собою набір програмних інтерфейсів, використання яких дозволяє розв'язати дві основні задачі [8]. По-перше, **Directx** перетворює розроблені з його допомогою додатка в сумісні з будь-якою версією **Windows** і працюючі на будь-якому комп'ютері, де встановлена ця операційна система, незалежно від типу використовуваного апаратного забезпечення. При цьому подібні додатки максимально використовують можливості комп'ютера, забезпечуючи найкращу продуктивність. Це досягається за рахунок сервісів, надаваних низькорівневими інтерфейсами **DirectxFoundation**. По-друге, **Directx** надає розроблювачам можливість абстрагуватися від того або іншого типу дисплейного адаптера, звукової карти або **3**D-Прискорювача й зосередитися на логіці роботи самої програми. Крім цього **Directx** розроблений таким чином, щоб автоматично підтримувати майбутні новації в програмних і апаратних забезпеченнях, і, отже, розроблювачі й користувачі можуть бути впевнені в тому, що й надалі будуть використовувати максимальні можливості своїх комп'ютерів.

Direct3D.Direct3D — це потужна, гнучка бібліотека, яка дозволяє розроблювачеві створювати додатки, що використовують тривимірну графіку, і в той же час не потребуючи глибоких знань складних принципів комп'ютерної графіки [4, 8]. Він підтримує два режими роботи — **ImmediateMode і RetainedMode**.

У режимі **ImmediateMode Direct3D** забезпечує розроблювачам апаратну підтримку ігрових і мультимедійних додатків у середовищі **Microsoft Windows**. Режим **RetainedMode Direct3D** полегшує створення й анімацію тривимірних світів, підтримуючи дві нові функції: інтерполятори анімації зі змішанням кольорів, плавними переміщеннями об'єктів і множиною різних видів трансформації, а також послідовне заповнення сіткової структури **3D**-Об'єктів (**meshes**), що дозволяє здійснювати їхнє поступове завантаження з вилучених серверів. Відзначимо, що **Direct3D**-Додатки спілкуються із графічними обладнаннями однаково, незалежно від

режиму — **Retained** або **Immediate**. Реально **Direct3D** є інтерфейсом до об'єкта **Directdraw**.

Directdraw — це менеджер керування пам'яттю для графічних і відеоповерхонь (surfaces) базовий набір, що забезпечує, функцій для мультимедійних додатків, що працюють на платформі **Microsoft Windows**. На відміну від традиційної **Windows**-Графіки **Directdraw** використовує прямий доступ до дисплейної пам'яті й графічним обладнанням, забезпечуючи при цьому повну сумісність із **Windows**-Додатками. **Directdraw** існує незалежно від **GDI** і обидва інтерфейси мають можливість прямого доступу до графічних обладнань через апаратно-незалежні шари [4, 5, 6]. На відміну від **GDI**, **Directdraw** по можливості використовує апаратні функції. Відзначимо, що **Directdraw** може надавати доступ до поверхонь як до контекстів обладнань, що дозволяє використовувати функції **GDI** для роботи з поверхнями. **Directdraw** підтримує роботу з більшим числом дисплейних адаптерів — від простих моніторів до складних професійних обладнань. Працюючи на рівні графічних поверхонь, **Directdraw** служить базою для високорівневих графічних функцій і інтерфейсів та дозволяє або використовувати апаратні можливості, надавані обладнаннями, або эмулірувати їх при необхідності.

DirectxTransform служить для реалізації різних типів трансформацій і перехідних ефектів, які можуть бути застосовані до графічних зображень і тривимірних об'єктів. Завдяки своїй гнучкій архітектурі, **DirectxTransform** може настроюватися на різні типи даних і, таким чином, є програмно розширюваним.

2.3 Mesh-об’єкти

Усі фігури, що малюються на сцені за допомогою **Direct3D**, являють собою сукупність точок – вершин. Інформація про кожну вершину містить такі параметри, як положення в просторі, колір вершини, інформацію про

використовувану текстуру і т.д. Таким чином, вершина в **Direct3D** являє собою єдину структуру даних, називану **FVF** (**FlexibleVertexFormat** – гнучкий формат вершини). На щастя, у керованому **Directx** є об'єкт, який може інкапсулювати вершини, що й завантажуються, а також індексувати данні. Такий об'єкт називається — об'єкт **Mesh** [8]. Об'єкти **Mesh** можуть використовуватися для збереження будь-якого типу графічних даних, але головним чином призначені для формування складних моделей. Технологія **Mesh** включає кілька методів або алгоритмів, що дозволяють збільшити ефективність рендеринга відображуваних об'єктів. Усі **Mesh**-Об'єкти містять верховий буфер і індексний буфер, плюс буфер атрибутів.

Фактичний об'єкт **Mesh** постійно перебуває в бібліотеці розширень (**Direct3D Extensionslibrary (D3DX)**). Перш ніж використовувати **Mesh**-об'єкти, необхідно додати посилання на бібліотеку **Microsoft.Directx.Direct3DX.dll**. Після того як ми це зробили, необхідно оголосити змінну для об'єкта **Mesh**. Її можна розмістити там же, де розміщалися б значення верхового або індексного буфери.

Є кілька готових об'єктів, які можна використовувати при використанні **Mesh**-Файлів.

Куб: *mesh = Mesh.Box(device, 2.0f, 2.0f, 2.0f);*

Даний метод створить новий **Mesh**-Об'єкт, який містить вершину й індекси, необхідні для рендеринга куба із заданою висотою, шириною й глибиною (значення 2.0f).

Width - ширина Визначає розмір куба уздовж осі X;

Height - висота Визначає розмір куба уздовж осі Y;

Depth - глибина Визначає розмір куба уздовж осі Z;

Циліндр: *mesh = Mesh.Cylinder(device, 2.0f, 2.0f, 2.0f, 36, 36);*

Radiusl- радіус циліндра на від'ємному кінці осі Z. Це значення повинне бути більше або дорівнює 0.0f.

Radius2- радіус циліндра на додатному кінці осі Z. Це значення повинне бути більше або дорівнювати 0.0f.

Length - довжина циліндра на вісі Z.

Slices - число секторів (slices) уздовж головної вісі (більше значення додасть більше вершин).

Stacks - число стеків вздовж головної вісі (більше значення додасть більше вершин).

Полігон: *mesh = Mesh.Polygon(device, 2.0f, 8);*

Length - довжина кожної сторони полігона.

Sides- число сторін полігона.

Сфера: *mesh = Mesh.Sphere(device, 2.0f, 36, 36);*

Radius - радіус сфери. Це значення повинне бути більше або дорівнювати 0.

Slices - число секторів (slices) уздовж головної вісі (більше значення додасть більше вершин).

Stacks - число стеків вздовж головної осі (більше значення додасть більше вершин).

Тор: *mesh = Mesh.Torus(device, 0.5f, 2.0f, 36, 18);*

Innerradius - внутрішній радіус тора. Це значення повинне бути більше або дорівнювати нулеві.

Outterradius - зовнішній радіус тора. Це значення повинне бути більше або дорівнювати трьом.

Rings - число кілець у поперечному перерізі тора. Це значення повинне бути більше або дорівнювати трьом.

Оперуючи цими примітивними фігурами та поєднуючи їх між собою, можна створювати досить складні конструкції. А якщо ще й змінювати їх текстуру, то можна отримати вражаючі візуальні ефекти та надати їм реалістичний зовнішній вигляд. Текстура - це зображення або шар, який накладається на поверхню Mesh-об'єкта. Це може бути колірна текстура, текстура зображення або текстура з ефектами, такими як блиск або відблиск.

Змінюючи текстуру, можна створювати різні ефекти, наприклад, симулювати різні матеріали, такі як дерево, скло або метал, або надати об'єкту деталізований вигляд, включаючи тиснення, візерунки або текст. Загалом, використання меш-об'єктів та змінення їх текстур дозволяє створювати різноманітні та реалістичні графічні об'єкти в комп'ютерній графіці та інших областях.

3 ПРОЦЕС КОМП'ЮТЕРНОГО МОДЕЛЮВАННЯ АНІМАЦІЇ СКЛАДНОГО РУХУ МАТЕРІАЛЬНОЇ ТОЧКИ

3.1 Етап візуального програмування

Етап візуального програмування є творчим, під час його реалізації розроблювач оснащений потужним інструментом середовища розробки **Microsoft VS** – набором готових компонентів, які необхідним чином розташовуються в додатку. На цьому ж етапі змінюються властивості компонентів, які відповідають за його оформлення, зовнішній вигляд і подальшу роботу додатка.

Для візуалізації анімації складного руху матеріальної точки був створений додаток, для якого нам знадобилося три форми: **Form1**, **Form2, Form3**. Одна форма головна. Саме вона з'являється першою при завантаженні додатка – **Form1**. Необхідно змінити цій формі властивість **FormBorderStyle** на **FixedDialog**. За допомогою другої форми – **Form2**, користувачеві надана можливість змінювати кінематичні параметри руху системи: кутової швидкості та параметрів *a* і b, що входять до закону руху матеріальної точки. Третя форма – **Form3**, призначена для зміни геометричних параметрів системи, а саме ширини рамки *CB* (Рис. 4). Найбільший інтерес представляє перша форма – **Form1**. Для спрощення подальшої роботи на цій же формі розташовується компонент **menustrip**, за допомогою якого генеруємо меню виду:

- **Рисунок =>**

  Намалювати

  Стерти
- **Рух =>**

  Почати

  Зупинити
- **Параметри =>**

Геометричні

Кінематичні

- **Траєкторія =>**

  Траекторія МТ

  Годограф We

  Годограф Wr

  Годограф Ve

  Годограф Vr

- **Вектори =>**

  We

  Ve

  Wr

  Vr

- **Вид зображення =>**

  Суцільний

  Каркасний

- **Вихід**
- **Довідка**

Розрахункові данні для користувача розташовані на головній формі на панелі **panel1**, щоб відділити робочу область від інформаційної. На цьому об’єкті треба розмістити сім компонентів **label**, що відображатимуть у собі назву тих даних, які показуватимуть сім компонентів **textBox**, яким для зручності на етапі написання програмного коду задати відповідні ім’я та, оскільки ці компоненти служитимуть для виводу інформації змінити їх властивість **ReadOnly** на **False**, щоб не можна було вводити інформацію.

Враховуючи специфічність дизайну, необхідно змінити колір деяким компонентам за допомогою властивості **BackColor**, а саме: головній формі задати колір **MintCream**, панелі, щоб відрізнялась, – **SeaGreen,**

контекстному меню – **PaleGreen**, у стані доступності – **Lime**, у стані недоступності, тобто тим кнопкам, що з самого початку будуть недоступними, а це: **годографWeToolStrip**, **годографWrToolStripMenuItem**, **годографVeToolStripMenuItem** та **годографVrToolStripMenuItem** – колір **OrangeRed**. Ці кольри впродовж створювання додатка будуть змінюватися на етапі написання програмного коду.

Для реалізації, безпосередньо, руху системи необхідно додати на головну форму компонент **timer1** та змінити інтервал за допомогою властивості **Interval** на **10**.

На другій формі – **Form2**, розміщено по три компоненти **label** та **numericUpDown** для зміни кінетичних параметрів конструкції. У перших трьох призначення нести в собі назву змінюваного параметру, тому змінено властивість **Text** відповідно назві змінюваного параметру, а у других – можливість змінювати самі значення відповідних параметрів. Розташовані вони навпроти один одного. Знов таки, для зручності під час написання програмного коду треба змінити ім’я компонентів **numericUpDown**. Цим же компонентам у властивості **DecimalPlaces** задати значення **1**, що означатиме кількість знаків після коми у десятинному дробу. Також для кожного з них треба виділити інтервал зміни значення за допомогою властивостей **Maximum**, **Minimum** згідно мети досягнення коректної роботи програми та розмір кроку, що визначає властивість **Increment**. Для першого це [7, -7] з кроком 0.5 , для другого – визначається програмно в ході роботи залежно від довжини рамки, для третього за бажанням програміста – [10, -10] з кроком 0.1. Також на цій формі містяться дві кнопки **button1** та **button2**, що подають сигнал програмі про згоду або небажання змінювати параметри. Їм також треба змінити колір на **Lime**.

На третій формі – **Form3**, розташовано по одному компоненту **label** та **numericUpDown** для зміни ширини рамної конструкції, яким задані відповідні значення, як і у компонентів другої форми. Це інтервал [0.5, 2.0] з кроком 0.1, для зміни якого по-перше треба змінити властивість **DecimalPlaces**, задавши значення **1**, що, так само, означатиме кількість знаків після коми у десятинному дробу, та властивість компоненту **label** – **Text**, вписавши назву змінюваного параметру. І знову, як і на другій формі – дві кнопки **button1** та **button2**, що також подаватимуть сигнал програмі про згоду або небажання змінювати параметри, яким також треба змінити колір на **Lime**. Це означатиме, що кнопки доступні.

На доданок треба змінити розміри другої та третьої форм до бажаних розмірів. Це можна здійснити вручну за допомогою звичайної стрілочки, навівши курсор миші на край форми.

Наведені вище дії є основними для розробки даного **Windows** – додатку. Усі інші зміни, що стануть наявними впродовж роботи з додатком були впроваджені на етапі створення програмного коду.

3.2 Розробка програмного коду

Другим етапом побудови програмного продукту в об'єктно-орієнтованім середовищі розробки **Microsoft VS** є написання функцій подій для наявних компонентів. При цьому потрібно враховувати, що на першому етапі середовище розробки самостійно генерує код для вже доданих компонентів.

Для роботи з **DirectX** у середовищі розробки **Microsoft VS 2012** необхідно переконатися, що в проект додані необхідні посилання на керований **DirectX**, що дозволять надалі використовувати його компоненти. Для цього, використовуючи меню проекту, необхідно кликнути на команду **Add References** і обрати додатки **Microsoft.DirectX і Microsoft.DirectX. Direct3D**. Для нормального подальшого функціонування програми та щоб звертатися до класів **DirectX**, не вказуючи кожного разу довгих префіксів просторів імен необхідно підключити деякі бібліотеки самостійно за допомогою директиви **using**, а саме **Microsoft.DirectX, Microsoft.DirectX.Direct3D,** що дозволяють автоматично ідентифікувати використовувані компоненти.

Програма складається з декількох файлів. Кожний файл містить функції загального призначення та функції деяких подій. Розглянемо роботу нашого додатка й найцікавіші функції подій.

Спочатку необхідно визначити об'єкт використовуваний додатком, і додати до нього нову змінну (**Device**) «**d3d**». Для цього включимо наступний рядок у розділ визначення класу:

```
Device d3d;
```

Далі треба заповнити вікно чистим кольором (перемалювати).У даному додатку було використано відтінок блакитного кольору – `Color.Azure`. У зв’язку з цим скористаємося процедурою **Clear**.

```
d3d.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Azure, 1.0f, 0); d3d.Present();
```

Перший параметр методу **Clear** визначає, що саме ми прагнемо очистити (поточне вікно). Другим параметром є колір, використовуваний при очищенні. Після того, як пристрій використаний, можемо оновити стан дисплея. Дана процедура виконується методом вистави **Present.**

Головним етапом візуалізації є створення конструкції з **Mesh**-об'єктів. Фактичний об'єкт **Mesh** постійно перебуває в бібліотеці розширень, тому перш ніж його використовувати, необхідно додати посилання на бібліотеку **Microsoft.DirectX.Direct3DX.dll**.

Після завантаження необхідних посилань у проект оголошується змінна для об'єкта **Mesh**. Враховуючи, що ми маємо справу зі складною рамною конструкцією, таких змінних треба оголосити декілька, кожна з яких буде відповідати за перемальовування відповідної частини конструкції.

```
Mesh M_tochka;
 Mesh os;
 Mesh cilindr_gorisontal;
 Mesh cilindr_vertical;
 Mesh plastina;
 Mesh verhn_krep;
 Mesh nign_krep;
 Mesh podshipnik;
 Mesh vector;
 Mesh vectorST;
```

- система складається з вісі, закріпленої шарнірно, тобто зверху міститься циліндр, у якому площа однієї основи дорівнює нулю (з геометричної точки зору це конус), а знизу сфера, що дозволяє їй з точки зору механіки обертатися, втоплена в брусі.
- на вісі додатково кріпиться рамка, яка складається з двох горизонтальних та одного вертикального циліндрів, нерозривно закріплених між собою.
- вільний простір між рамними циліндрами заповнює пластина, що являє собою прямокутник із дуже вузькою однією зі сторін.
- Вектори складаються з подовжених тонких циліндрів та невеликих конусів на кінці, імітуючи стрілочку.
- матеріальною точкою є маленька сфера.

У методі **Form1_Load** міститься код, що створює моделі **Mesh**-об'єктів.

Найпростішою є функція **private void выходToolStripMenuItem1_Click(object sender, EventArgs e)**, що викликається кнопкою меню **«Вихід»**. Вона просто закриває головну форму, у зв’язку з чим закривається вся програма.

Більш важливою для користувача є кнопка меню **«Довідка»**, що виклакає функцію **private void справкаToolStripMenuItem_Click(object sender, EventArgs e)**. Ця функція просто відкриває вказаний файл довідки, розроблений окремо.

Функція **private void Picture()** практично є основною. Вона викликається багатьма кнопками контекстного меню: «**Рисунок => Намалювати»**, «**Траєкторія»**, «**Вектори»**, «**Вид зображення»**. Саме вона відповідає за появу зображення на екрані монітору. Залежно від того, на який пункт меню натискаю користувач, виконується відповідна частина цієї функції. Та вона не в змозі самостійно відповідати за рух системи, тому її викликає функція **private void timer1_Tick(object sender, EventArgs e)**, яка сама викликається програмою із частотою 10 разів на секунду, що зумовлено властивістю компонента **timer1 – Interval – 10.**

Складовою попередньої функції є функція, що відповідає за положення матеріальної точки вздовж координатної вісі ***OY*** – **private float Координата_Y_MT(float t1)**. Тут запрограмовано закон руху матеріальної точки, та,оскільки усі вектори кінематичних характеристик виходять саме з неї, отже і усіх векторів.

За допомогою кнопки меню **«Рисунок => Стерти»**, користувач викликає функцію **private void Ochis()**, в якій просто весь малюнок замальовується кольором головної форми та зтираються дані з текстових віконечок, що розташовані на панелі.

При натисканні користувачем пункту меню **«Рух => Почати»**, викликається функція події **private void начатьToolStripMenuItem_ Click(object sender, EventArgs e)**. У ній викликається попередня функція та вмикається **timer1**. Зображення починає перемальовуватися згідно нових координат на новому фоні, що забезпечує саме попередня функція. Ця функція також відповідає за коректність в роботі додатку, тобто деякі кнопки тут стають недступними та змінюють свій колір згідно дизайнерській задумці. Такі дії виконує майже кожна кнопка меню, але в залежності від того, що робить у наданий час програма.

Протидією цьому є подія, що настає після натискання пункту меню «**Рух => Зупинити**». Через це викликається наступна функція – **private void остановитьToolStripMenuItem_Click(object sender, EventArgs e)**, де просто вимикається **timer1**. Тут також змінюється дизайн меню згідно виконаних дій.

Кнопка меню «**Параметри => Геометричні**» викликає функцію події **private void геометрическиеToolStripMenuItem_Click(object sender, EventArgs e)**, яка перевизначає усі **Mesh** – об’єкти згідно зі зміненими розмірами рамки, тобто налаштовує усі інші частини конструкції, розміри та положення яких залежать від ширини рамки, та робить баченою третю форму **Form3**.

На відміну від попередньої, функція події **private void кинематическиеToolStripMenuItem_Click(object sender, EventArgs e)**, виклик якої зумовлено вибором кнопки меню «**Параметри => Кінематичні**», просто відкриває другу форму **Form2**.

Пунктом меню **«Вектори»** та в залежності від вибору користувача з наведених у цьому пункті кнопочок викликаються наступні функції подій:

private void wnToolStripMenuItem_Click(object sender, EventArgs e), **private void wrToolStripMenuItem_Click(object sender, EventArgs e)**, **private void veToolStripMenuItem_Click(object sender, EventArgs e)** та **private void vrToolStripMenuItem_Click(object sender, EventArgs e)**. Вони є подібними, тому що виконують подібні діїї, але відносно різних об’єктів, а саме вмикають або вимикають зображення відповідних векторів, а також відповідно умовам коректності змінюють дизайн контекстного меню.

Подібними є також функції, що викликаються пунктом меню **«Траєкторія»**:

private void траєкторияMTToolStripMenuItem_Click(object sender, EventArgs e),

private void годографWeToolStripMenuItem_Click(object sender, EventArgs e),

private void годографWrToolStripMenuItem_Click(object sender, EventArgs e),

private void годографVeToolStripMenuItem_Click(object sender, EventArgs e) та

private void годографVrToolStripMenuItem_Click(object sender, EventArgs e). Ці функції просто вмикають/вимикають позначку навпроти того підпункту меню з наведених у пункті **«Таєкторія»**.

Кнопка меню **«Вид зображення»** дозволяє вибирити між каркасним та суцільним видом зображення. За цим стоять дві подібні між собою фукції події: **private void сплошнойToolStripMenuItem_Click(object sender, EventArgs e) та private void каркасныйToolStripMenuItem_Click(object sender, EventArgs e)**. Але в них самих просто змінюється положення позначки про те, який саме з них було вибрано, а сама зміна зображення зінюється фактично саме в функції **private void Picture()**.

У функції **private float Skorost(float t1)** та **private float Uskorenie(float t1)** розраховуються абсолютні швидкість та прискорення, які виводяться у відповідні віконця за допомогою функції **private void Form1_Shown(object sender, EventArgs e)**.

Таким чином побудований **Windows** – додаток, який моделює анімацію складного руху матеріальної точки за наведеним законом руху (21).

2.3. Зразок роботи Windows- додатка.

З метою візуалізації складного руху матеріальної точки було створено даний **Windows**-додаток, що надає користувачеві можливість побачити анімований складний рух матеріальної точки у просторі за наданим законом (21) [п.2.1.], а саме оберт із заданою кутовою швидкістю рамної конструкції у просторі навколо своєї вісі, навпроти якої (на іншій вісі рамної конструкції) матеріальна точка закріплена таким чином, що має можливість рухатися вздовж вісі, що її утримує.

Починаючи роботу з додатком, користувач ніби спілкується з ним, тому що спочатку баченими є контекстне меню, завдяки якому можна задавати програмі команди, що виконують бажану дію з наданих, та панель внизу форми, на якій міститься інформація про початкові дані та ті дані, що отримуються в ході розрахунків або змінюються самим користувачем [додаток 1 (рис. 5)]. На початку та впродовж користування додатком деякі кнопки контекстного меню будуть змінювати свій стан на досяжний або недосяжні. Це не є недоліком програми, а, насамперед, зумовлено її логічною та коректною роботою. Треба погодитися, що не є логічним затерти малюнок навмисно, обдумано або ненароком, коли той здійснює рух, при якому, на доданок, підраховуються деякі потрібні дані.

Отже розпочнімо роботу з додатком!

Якщо натиснути на кнопку меню «**Рисунок => Намалювати»**, то на екрані монітору можна побачити зображення рамної конструкції [додаток 1 (рис. 6)]. У цей момент або перед виконанням вищезгаданих дій, а також впродовж усього використання додатка, можна змінити вид зображення, звернувшись за допомогою до пункту меню «**Вид зображення»,** який запропонує вибрати: **«Суцільний** або **Каркасний»**. У першому випадку результатом буде рисунок 6 з додатку 1, у другому – рисунок 7 з того ж додатку. Перевага виду «**Каркасний»** полягає у тому, що саме таким чином можна побачити вектори відносних швидкості та прискорення матеріальної точки, що зображені згідно розрахунків вздовж вісі, на якій закріплено саму матеріальну точку. На даному етапі використання додатка можна здійснити зображення векторів, перелік яких запропоновано у пункті меню «**Вектори»** [додаток 1 (рис. 9)]. Зображені вектори відрізнятимуться за кольором, а саме: вектори прискорень будуть чорного кольору, а вектори швидкостей синього. Такий підхід було реалізовано з метою надання можливості візуально без перевірки за допомогою меню, де навпроти ввімкненого вектору будуть встановлені мітки, відрізняти, наприклад, вектор відносної швидкості від вектору відносного прискорення, що, як вже було згадано, направлені вздовж однієї вісі, та мають властивість змінювати свій напрямок в залежності від часу згідно закону руху (21) [п.2.1.],.

Можна розпочинати рух системи. Цю дію можна реалізувати скориставшись кнопкою контекстного меню «**Рух => Почати»** [додаток 1 (рис. 8)]. Протидією цьому може бути наступний вибір користувача: **«Рух => Зупинити»**. Доки рух не буде зупинено, є можливість подивитися, які траєкторії описують матеріальна точка та годографи векторів швидкостей та прискорень за допомогою, знов таки, кнопки меню, а саме «**Траєкторія»**. Далі буде запропоновано обрати бажаний об'єкт з наведених, навпроти якого також буде автоматично встановлено мітку про увімкнення цього пункту меню [додаток 1 (рис. 10 – рис. 13)]. Та є одна незначна умова: годографи

векторів можна вмикати за умовою увімкнення самих векторів, тобто, якщо кнопка увімкнення вектора не буде вибрана, завдяки чому сам вектор не буде баченим, то і кнопка увімкнення його годографу доступною не буде. Це зумовлено, знов таки, коректною та логічною роботою програми. Як тільки прибрати мітку навпроти вектора, зникає і відповідний годограф з описаною ним траєкторією. З зупинкою руху, зображення не зникає разом із отриманими траєкторіями. Програма ніби переходить в режим очікування певної дії від користувача. Це може бути будь що з наведених та доступних, окрім «**Рисунок => Намалювати»**, тому що рисунок вже є намальованим, та **«Рух => Зупинити»**, тому що зупинку вже зроблено.

Усе зображення зникає при виборі кнопки «**Рисунок => Стерти»**. У цьому разі вікно знов приймає вид, як показано на рисунку 5 додатка 1. Тепер можна починати роботу з нуля.

Під час роботи з додатком можна змінювати деякі параметри руху системи, а саме: геометричні, що відповідають за розміри рамної конструкції (її ширину), та кінематичні, що, безпосередньо, входять до складу закону руху матеріальної точки та кутову швидкість, що відповідає за рух системи в цілому. Цю дію виконує пункт меню «**Параметри => Геометричні** або **Кінематичні»**.

Натиснувши «**Параметри => Геометричні»**, користувач побачить нове вікно під назвою *«Зміна геометричних параметрів»*, у якому можна змінити ширину рамки з проміжного значення 1.5 [додаток 2 (мрис. 14)], що задане за умовчанням на мінімальне – 0.5 [додаток 2 (рис. 15)] або максимальне – 2.0 [додаток 2 (рис. 16)]. Такий інтервал зумовлений коректною робою додатку та з урахуванням розмірів моніторів.

Якщо вибрати «**Параметри => Кінематичні»**, з’явиться наступне вікно під назвою *«Зміна кінематичних параметрів»*. Тут можна змінювати три параметри, а саме: параметр *a*, параметр *b* та кутову швидкість, з проміжних, поданих за умовчанням [додаток 3 (рис. 17)], на максимальні

[додаток 3 (рис. 18)] або мінімальні [додаток 3 (рис. 20)]. Інтервал зміни кутової швидкості зумовлений коректним графічним зображенням, тобто немає сенсу задавати дуже велику кутову швидкість, тому що у такому разі конструкція обертається дуже швидко і зникає можливість розгледіти хоча б щось із зображеного силуету. Значення із знаком «-» забезпечують обертовий рух в іншу сторону. При спробі задати нульову кутову швидкість кнопка згоди на формі *«Зміна кінематичних параметрів»* стає червоного кольору [додаток 3 (рис. 19)] та при її натисканні видається повідомлення про помилку [додаток 3 (рис. 21)]. Нові, задані користувачем, параметри будуть відображені у головній формі на панелі внизу самої форми.

У відповідь на подію, яка генерується натисканням пункту меню **«Довідка»** з'являється вікно під назвою *«Довідка»*, яке можна побачити на рисунку 22 додатка 4. Саме тут можна знайти відповідь на цікавлячи питання та загалом розібратися у принципі роботи програми.

4 ІНСТРУКЦІЯ ДЛЯ КОРИСТУВАЧА

Help-файл або довідкова система – найнеобхідніший елемент будь-якої програми, навіть самої невеликої. Добре, коли довідка дає можливість швидко освоїти принцип роботи з програмою, фактично не звертаючись за допомогою додаткових підручників. Її зміст може бути зовсім маленьким – таким, як довідка до блокноту **Windows**, або величезним – таким, як **MSDN (** бібліотека, що являє собою за змістом, гігантську довідкову систему).

Різновидів довідкових систем дві – традиційна (.**hlp**) та сучасна (**.chm**). У даному додатку було використано традиційний різновид довідки у форматі (**.hlp**). Створення довідкової системи у форматі **hlp** - не зовсім тривіальна задача. Для цього необхідна програма для редагування **HTML** (**Notepad, Microsoft Frontpage, Homesite, etc**.) і компілятор **HTML Help** (наприклад **HTML HelpWorkshop від Microsoft**). Щоб створити довідковий файл, необхідно спочатку підготувати файли змісту, розділів та проекту. Файл змісту визначає структуру та зовнішній вигляд вкладки **Contents.** Файл розділів являє собою документ **RTF** та вміщає в собі текст та графіку, з яких складається кінцевий документ та виводиться у довідковому файлі. З метою створювання такого файлу використовують **Word**. Файл проекту об’єднує файли розділів та змісту.

Help пишеться для користувачів, і необхідно писати його мовою самих користувачів. **Help** - це не технічний опис програми: у більшості випадків користувача мало хвилює, що "даний елемент є контролом типу **Richtextbox,** і тому йому властиві всі обмеження даного типу контрольних елементів" (реальний приклад з однієї програми, що не має ніякого відношення до програмування). Крім того, необхідно пам'ятати, що в довідку звичайно заглядають коли знадобиться знайти відповідь на якесь конкретне питання. Взагалі, для тексту й змісту довідкових систем характерні два підходи: або розповісти для чого щось призначене, або про те, як щось зробити (**wantingtoknowvs. wantingtodo**). Обоє цих підходів будуть раціональними.

Вирішувати який підхід вибрати необхідно, природно, виходячи з тих питань, які задають користувачі на стадії, коли **Help** ще немає (а звичайно, поки програма існує у вигляді перших бета-версій, його й не буває). Зрештою, грамотно написаний **Help** позбавить від купи додаткових питань користувачів.

Однієї із самих широко розповсюджених помилок при створенні довідкової системи є зловживання довгими й найдокладнішими поясненнями, що перетворюють **Help** у якусь подобу наукового трактату. Докладні пояснення потрібні для друкованої документації, а в **Help** необхідно швидко допомогти користувачеві знайти відповідь на хвилююче його питання. Треба сказати, що довідкова система - це не тільки якийсь абстрактний файл довідки, але ще й контекстна система допомоги. Звичайно користувачі досить неохоче беруться запускати перший пункт із меню **Help** (тому, що в цьому випадку їм доведеться шукати й розбиратися в довідці), але цілком готові нажати **F1** на якомусь пункті меню або скористатися питанням "**What'sThis**?" у діалогах - так що саме розробці системи контекстної допомоги слід приділити найбільше уваги. А це не зовсім тривіальна задача, над якою треба подумати не менше, чим над проектуванням будь-якої іншої частини інтерфейсу програми.

Є дві речі, які, безумовно, необхідні будь-якій професійній програмі. По-перше, така довідкова система, яка дозволяє легко й швидко розібратися у всіх можливостях вашої програми. І, по-друге, такий інтерфейс, який дозволить користувачеві ніколи не замислюватися про те, є чи у вашої програми **Help.**

У нашому додатку розроблений файл довідки традиційним способом. Був використаний компілятор довідки **Microsoft HelpWorkshop.** Довідковий файл складається з файлу змісту, розділів і проекту, які готувалися на різних етапах написання. Спочатку був підготовлений файл змісту. Він визначає структуру й зовнішній вигляд довідки. Потім файл розділів, який містить

текст і малюнки й з яких складається підсумковий документ. Для його створення було використано звичайний **Microsoft Word.** Завершальним етапом було створення файлу проекту, який об'єднав файли розділів і змісту.

Для підключення довідки в додаток був використаний клас **Help.** Він дозволяє додатку відображати користувачам файли довідкової системи. Новий екземпляр класу створити неможливо. Тому, щоб додати довідку до нашого додатка, ми скористалися статичним методом **Showhelp()** [додаток 4 (рис. 22)]. Метод класу **Help** викликається в нашому додатку у відповідь на подію, яка генерується натисканням користувача пункту меню **«Справка».** Це логічно й зручно для користувача.

ВИСНОВКИ

У даній роботі розроблено **Windows-**додаток, який моделює складний рух матеріальної точки. Для рамної конструкції, на бічному ребрі якої рухається вагома точка, змодельована **3-D** анімація з використанням бібліотеки **DirectX9.0**.

Користувач розробленого додатку може задавати різні значення для параметрів задачі: геометричні розміри рамної конструкції, кутову швидкість переносного руху, амплітуду та частоту відносного руху матеріальної точки. Під час вводу значень робиться перевірка на коректність даних.

Додаток має функціональні можливості побудови векторів переносної та відносної швидкостей, доцентрового прискорення переносного руху та лінійного прискорення відносного руху. Якщо якісь з векторів обрано, то вони зображуються також під час руху матеріальної точки, згідно свого напряму у даний момент часу. У відповідних вікнах додатку виводяться значення кінематичних величин.

Для аналізу складного руху користувач може вибрати відповідні опції побудови траєкторії руху матеріальної точки у абсолютному русі, годографів векторів швидкостей та прискорення.

Для побудови моделі конструкції та траєкторій і годографів використовувалися різні підходи **3-D** графіки. Сама модель конструювалася з готових **Mesh**-об’єктів, а просторові криві з використанням масиву вершин. Для побудови просторових кривих були використані списки точок з координатами матеріальної точки у складному русі.

Крім того, у додатку є опції: відображення конструкції у каркасному вигляді або у вигляді суцільних об’єктів, очищення отриманих візуальних даних та довідка про користування додатком.

Перспективи подальшого розвитку теми "Комп'ютерне моделювання руху матеріальної точки" можуть включати такі аспекти:

1. Розширення функціональності: Додаток може бути розширений для моделювання більш складних систем, а також розвивати нові алгоритми та методи, які дозволять моделювати різноманітні фізичні явища та взаємодії.
2. Вдосконалення візуалізації: Можна покращити візуальне представлення руху матеріальної точки шляхом використання більш реалістичних графічних ефектів, текстур, освітлення та анімації. Такі покращення дозволять користувачам краще розуміти та сприймати результати моделювання.
3. Інтеграція з іншими технологіями: Додаток може бути інтегрований з іншими технологіями, такими як віртуальна реальність (VR) або розширена реальність (AR), що дозволить користувачам взаємодіяти з моделлю в більш іммерсивний спосіб (повністю занурює користувача в віртуальну або розширену реальність, створюючи відчуття поглинання і присутності у цьому віртуальному середовищі) або спостерігати рух матеріальної точки в реальному часі у фізичному середовищі.

Загалом, перспективи подальшого розвитку теми "Комп'ютерне моделювання руху матеріальної точки" є широкими і включають різноманітні аспекти, що дає можливість покращити функціональність, візуалізацію та застосування додатку у різних галузях. І, хоча коливання матеріальної точки в рухомій системі відліку та робототехніка можуть існувати як окремі області, їх можна поєднувати. Наприклад, роботи з механічними руками або ногами можуть мати рухому систему, що дозволяє їм здійснювати коливальні рухи для виконання певних завдань. Отже, розуміння коливань матеріальних точок у рухомій системі відліку може мати практичне значення для розвитку робототехніки, допомагаючи створювати більш ефективні та функціональні роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "Robotics: Modelling, Planning and Control" - Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, Giuseppe Oriolo, 2009.– 696p.
2. Бухгольц Н.Н. Основной курс теоретической механики. Часть 1. Динамика системы материальных точек./Издание 4, переработанное и дополненное Таргом С.М// - М..:.Наука, 1965. – с. 183 -200
3. Яблонский А.А. Курс теоретической механики./ Издательство «Высшая школа» - 1966. – 520 с.
4. Эндрю Троелсен. C# и платформа .NET. / Библиотека программиста/ Москва., Санкт-Петербург: Издательство «Питер». – 2004. – 785 с.
5. Джесс Либерти. Программирование на C#./ 2-е издание.
6. Шилдт Г. Полный справочник по C#. /. М.: Издаетльский дом «Вильямс». 2004 – 740 с.
7. Культин Н.Б. C# в задачах и примерах. – БХВ.-Перетбург, 2007. – 240с. : с иллюстрациями.
8. Том Миллер. DirectX 9 с управляемым кодом. Программирование игр и графика./ М.: Издательский дом «КомБук», 2005. – 395 с.
9. Мещерский И. В. Сборник задач по теоретической механике: Учеб. Пособие.–36-е изд., исправл./ Под ред. Н. В. Бутенина, А. И. Лурье, Д. Р. Меркина.–М.: Наука. Гл. ред. физ. мат. лит., 1986.–448с.

Додаток №1 Головна форма

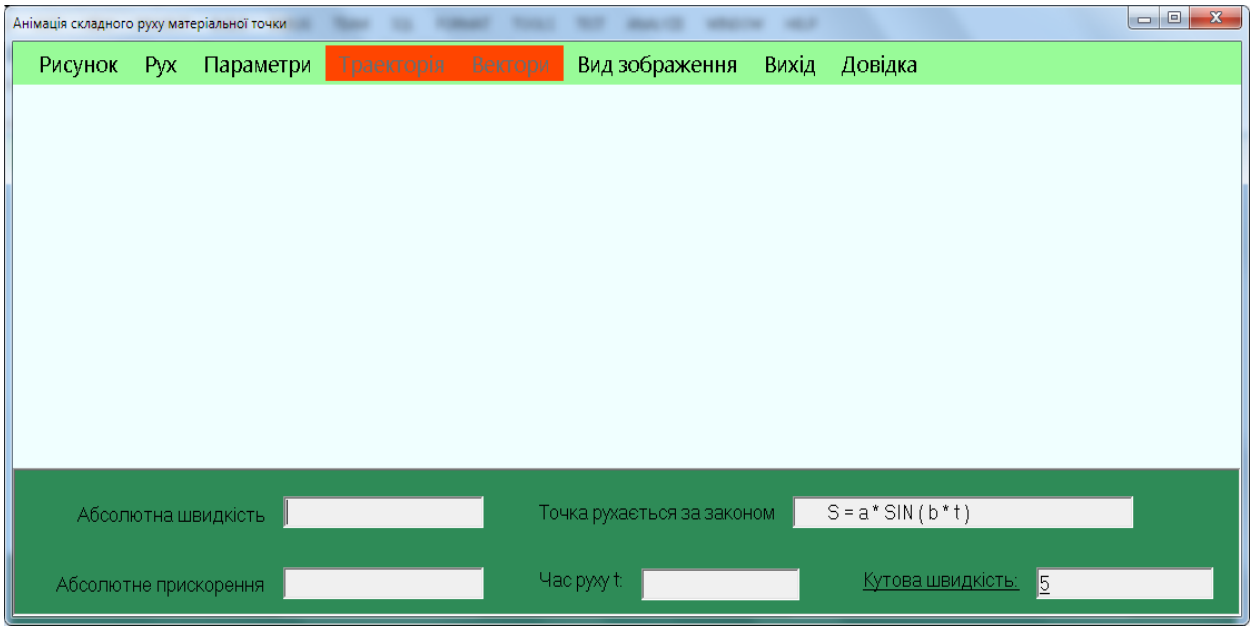

Рисунок 5

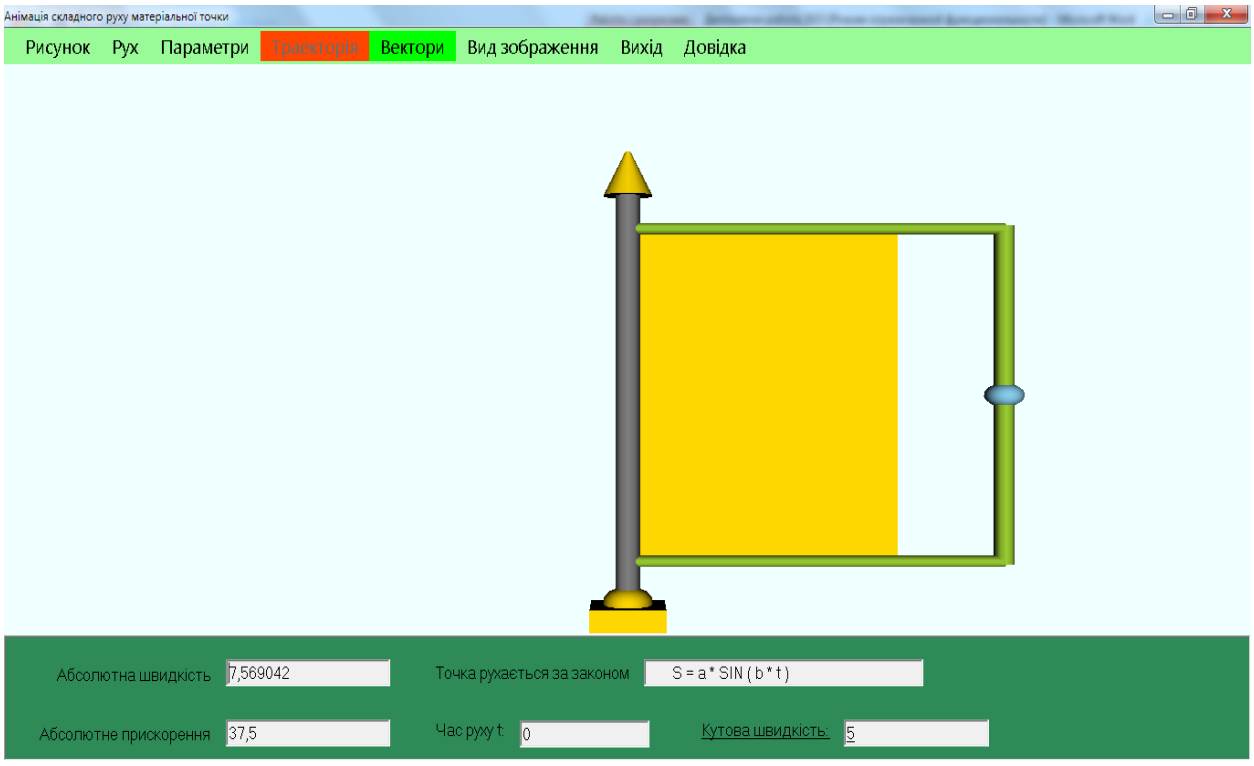

Рисунок 6

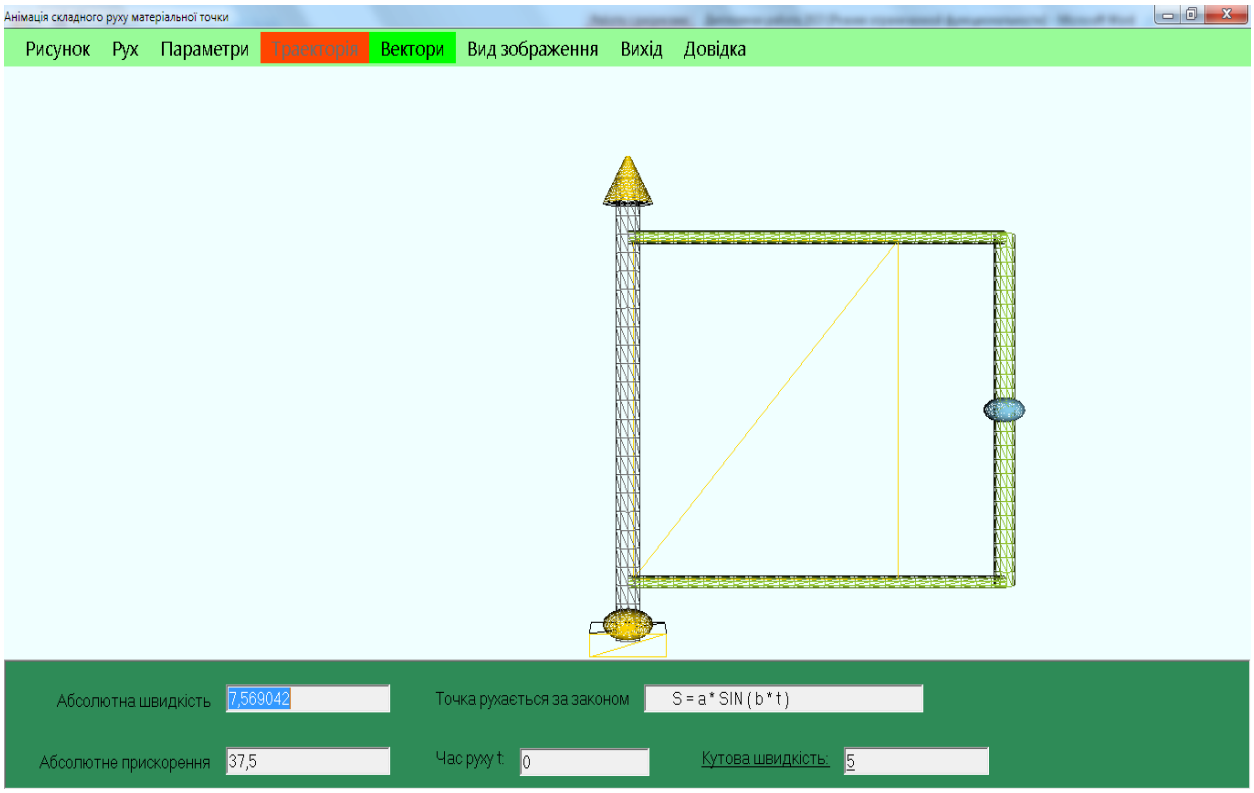

Рисунок 7

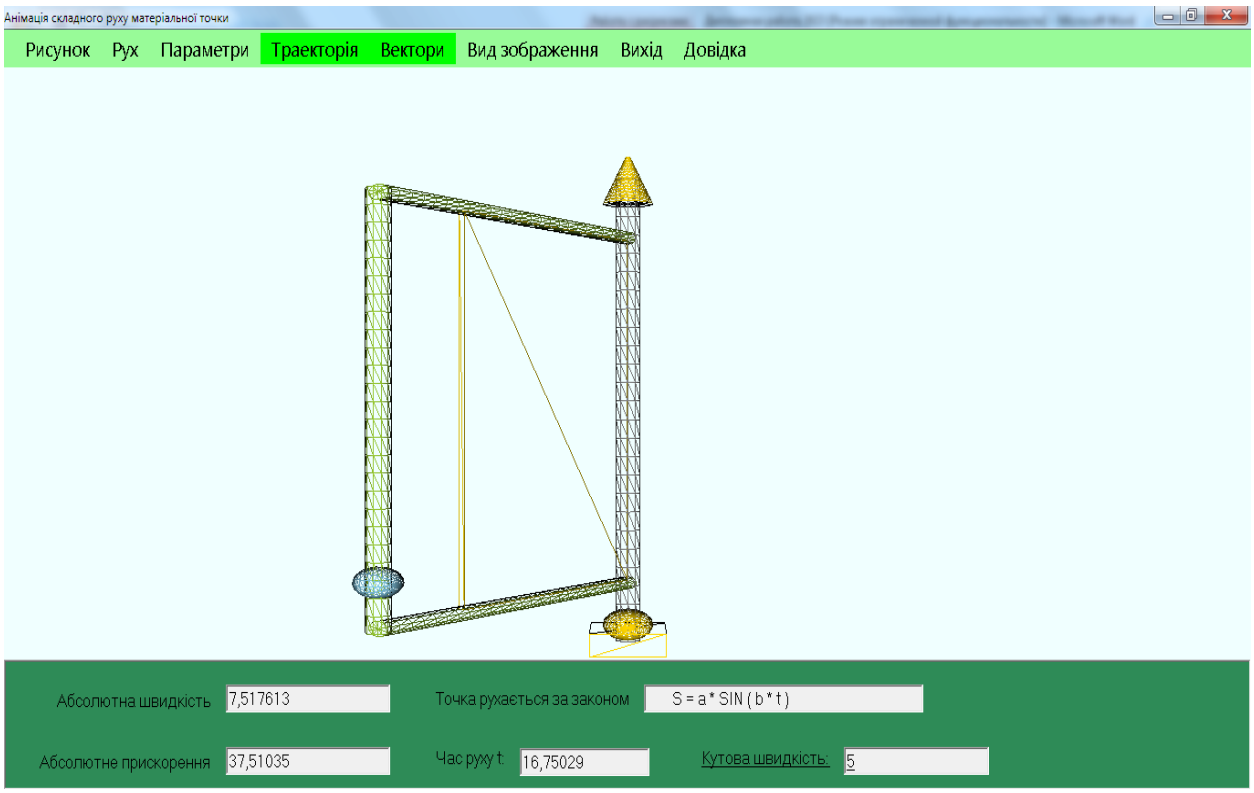

Рисунок 8

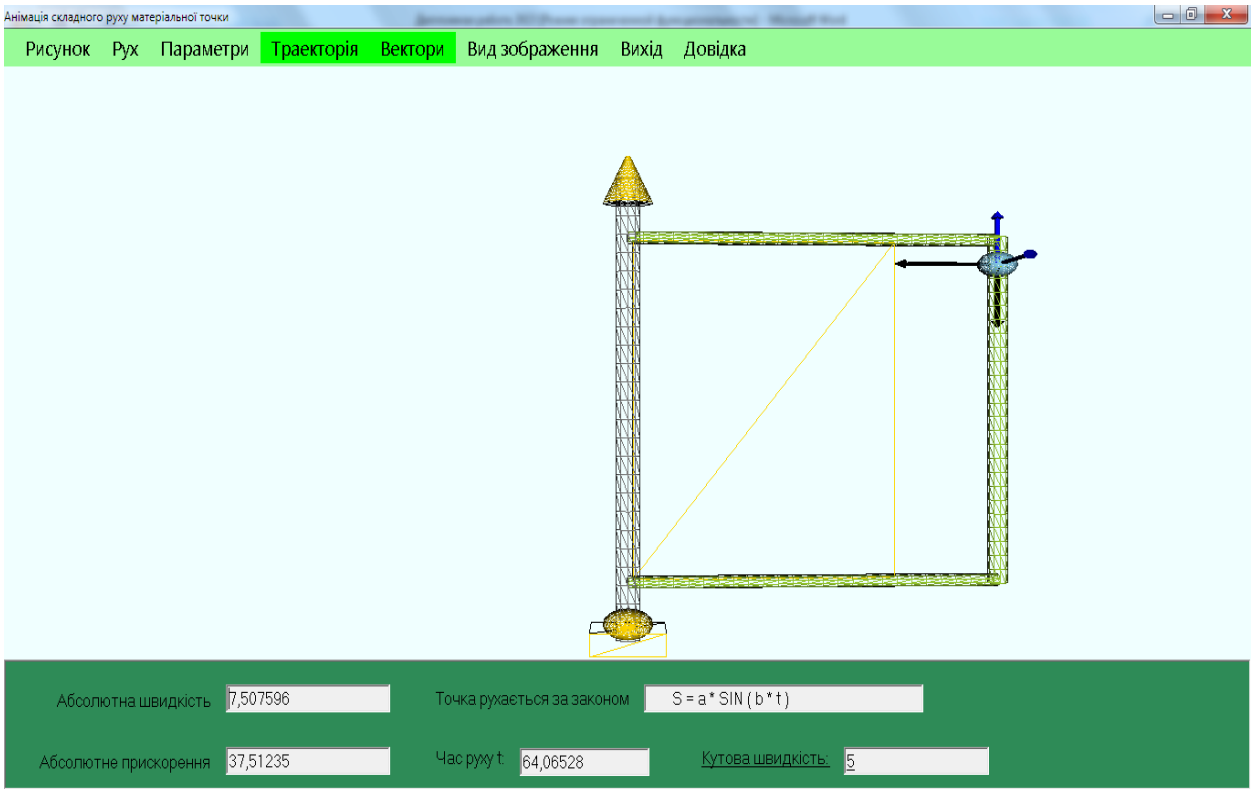

Рисунок 9

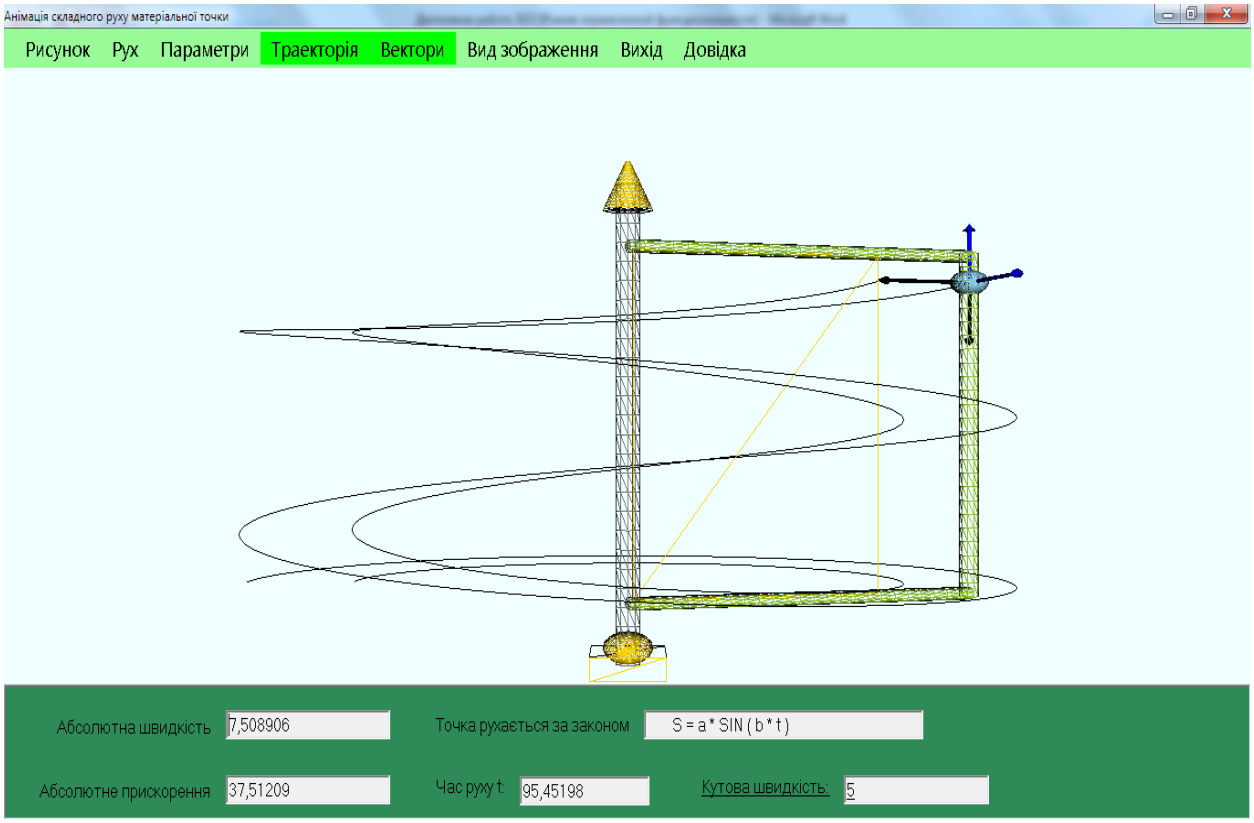

Рисунок 10

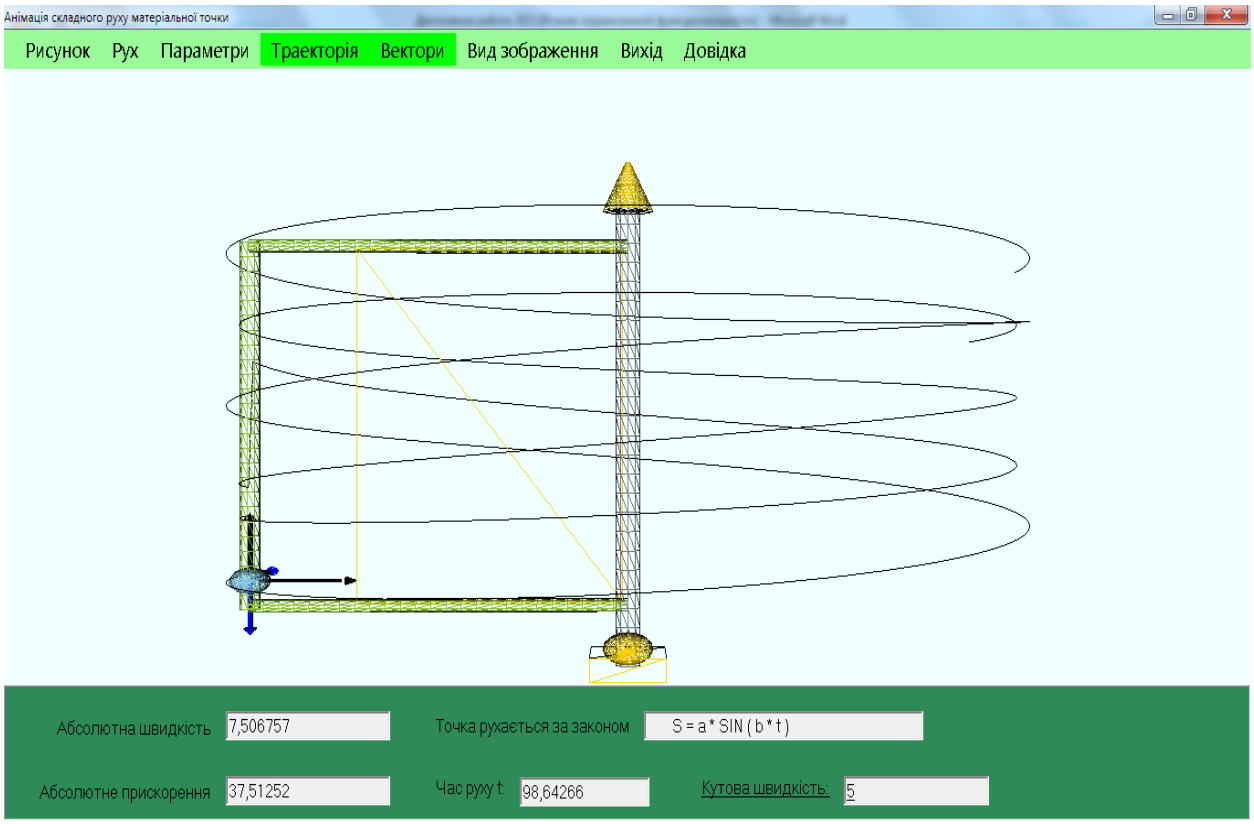

Рисунок 11

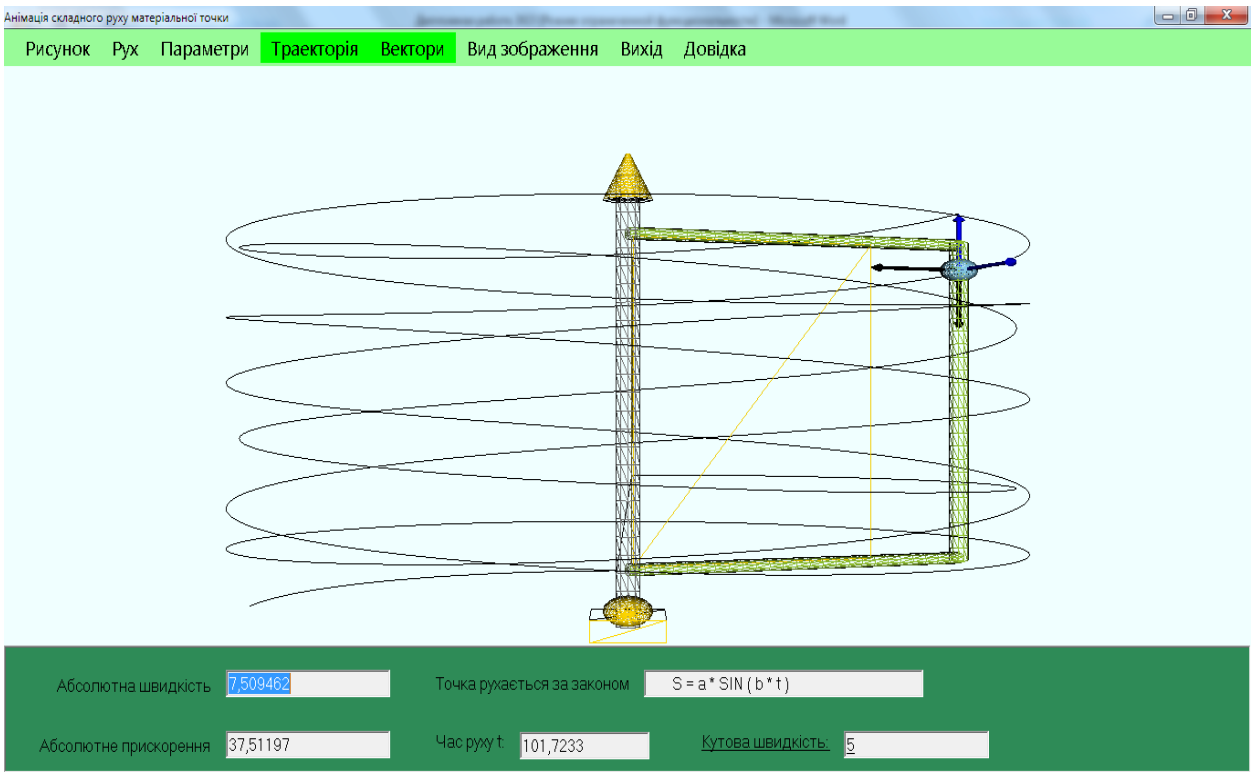

Рисунок 12

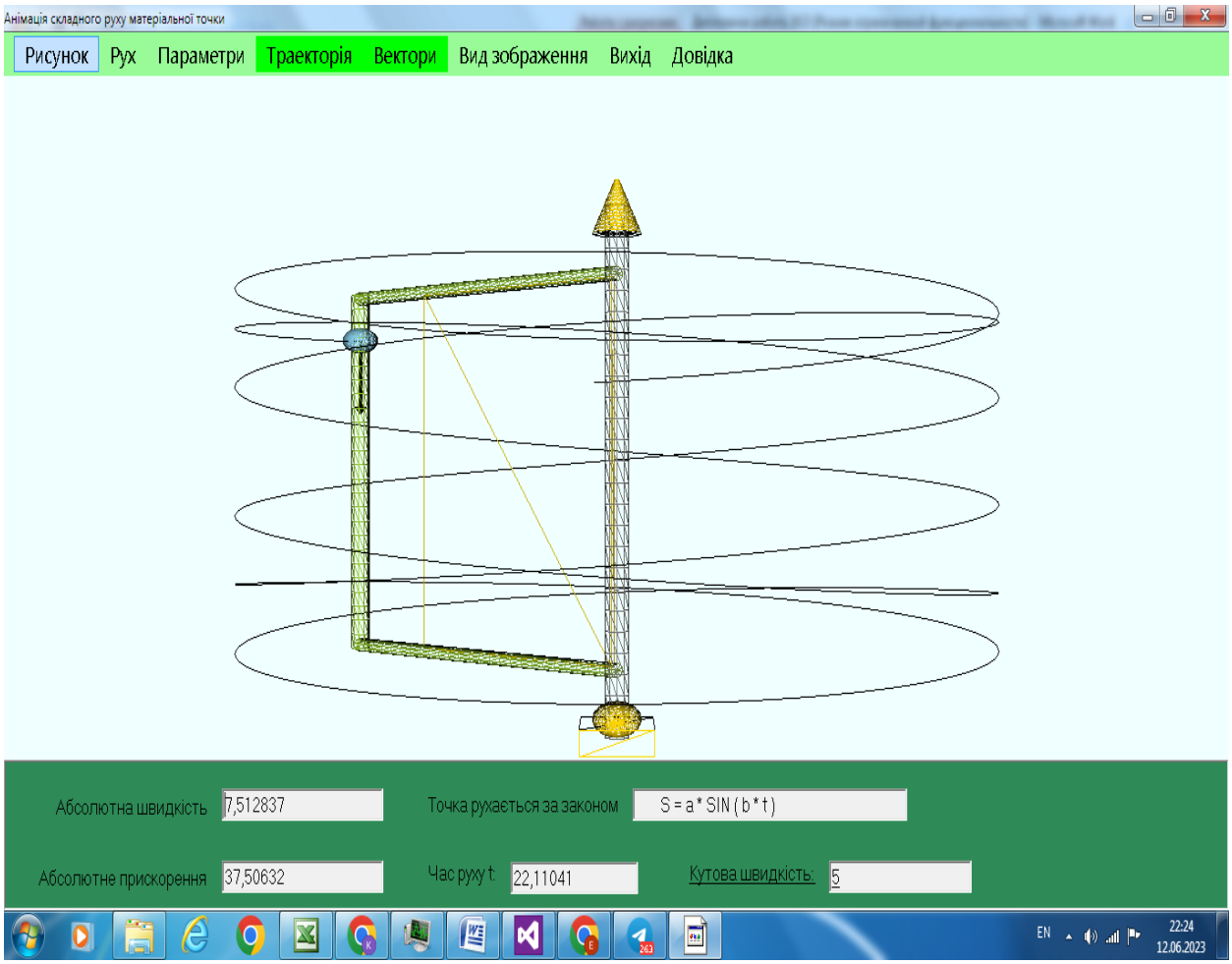

Рисунок 13

Додаток №2 Форма для зміни геометричних параметрів системи

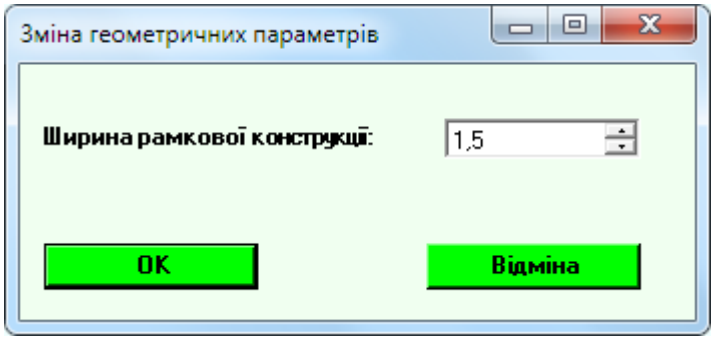

Рисунок 14

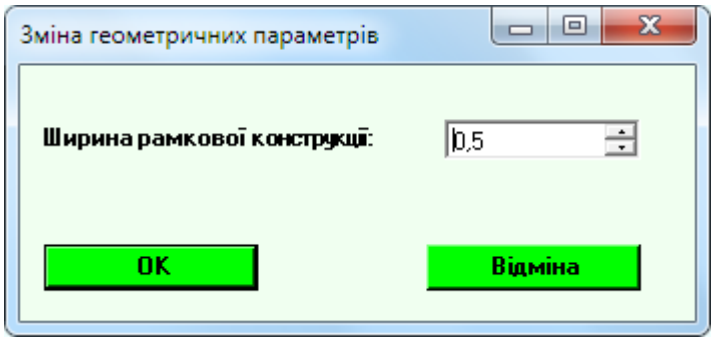

Рисунок 15

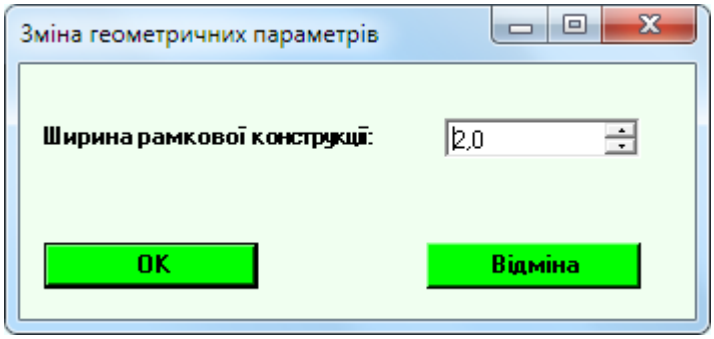

Рисунок 16

Додаток №3 Форма для зміни кінематичних параметрів системи

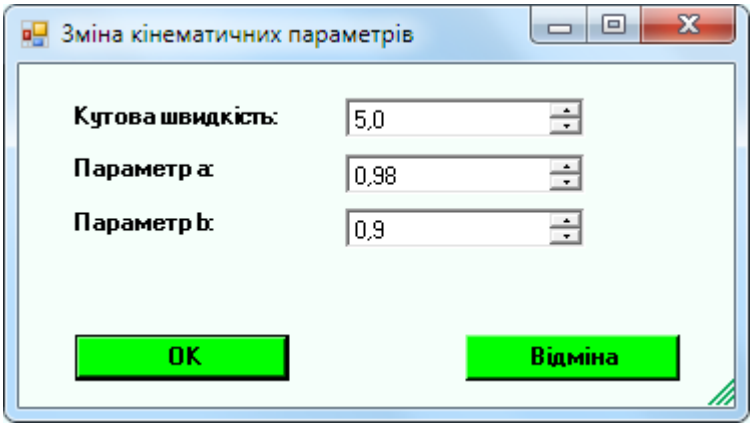

Рисунок 17

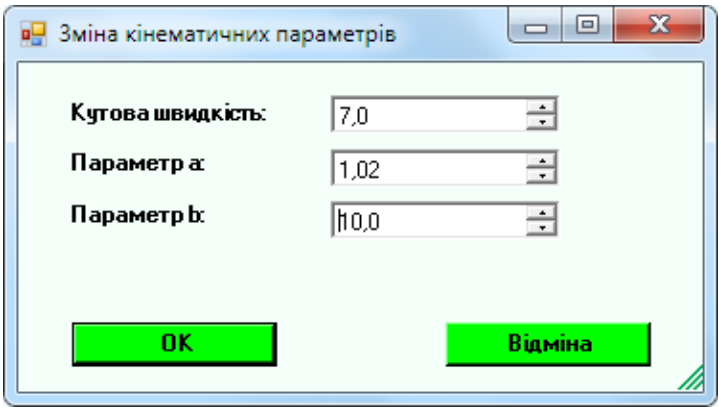

Рисунок 18

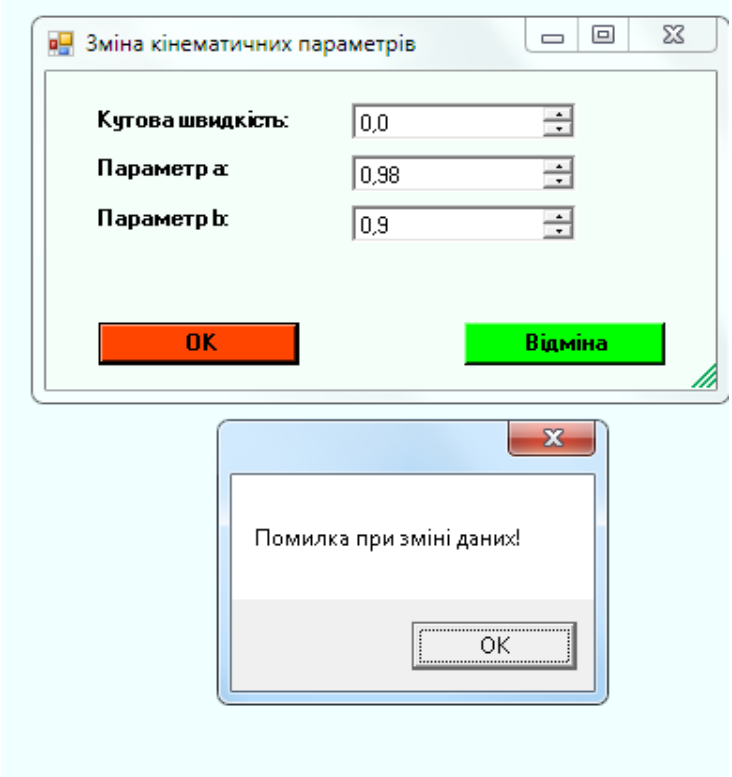

Рисунок 19

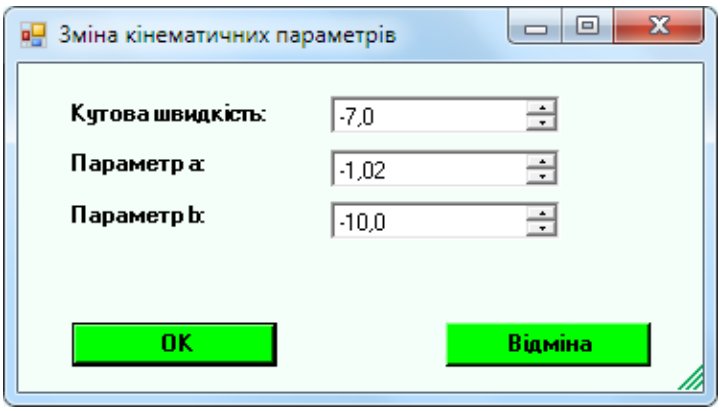

Рисунок 20

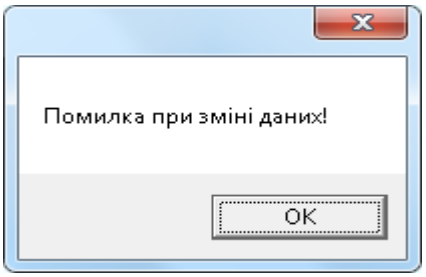

Рисунок 21

Додаток №4 Довідка для користувача

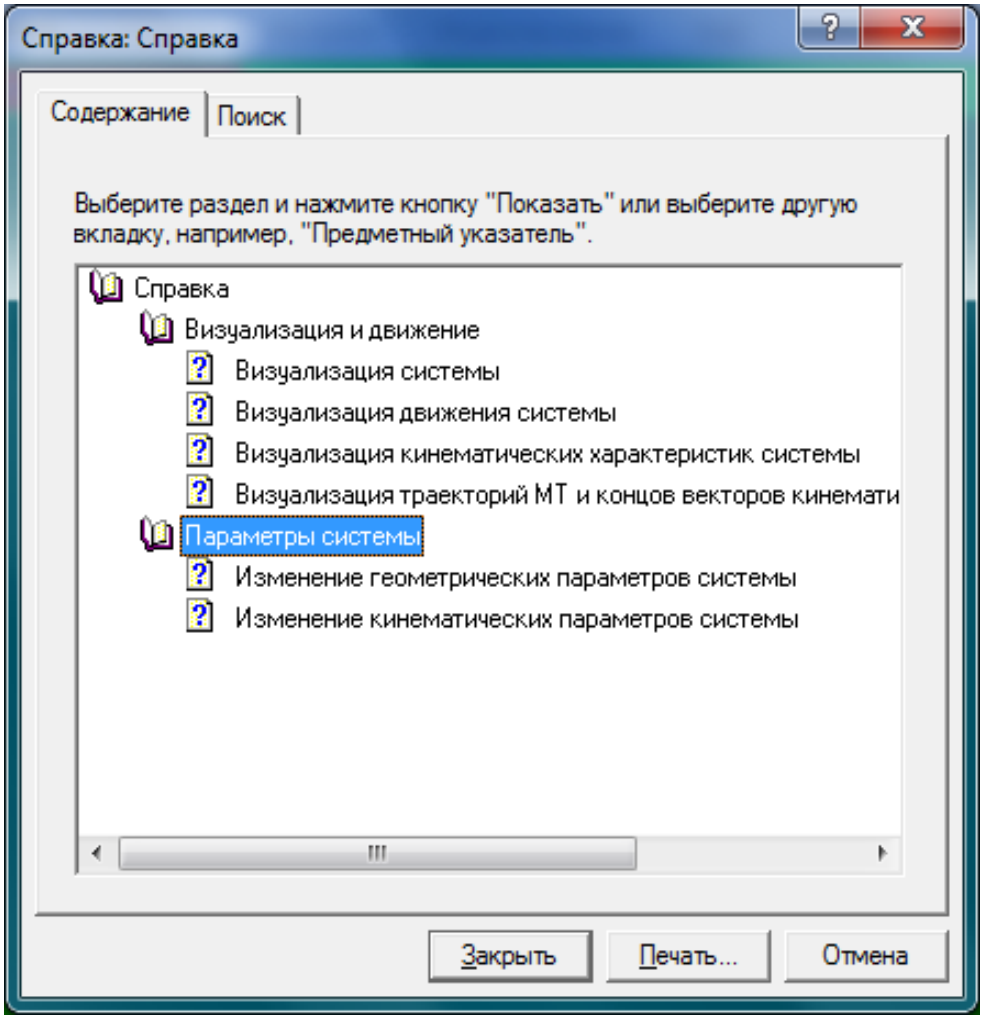

Рисунок 22

Додаток №5 Код програми Windows - додатка

Файл Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
using System.Windows.Forms;
namespace Dvigenie_MT
{
  public partial class Form1 : Form
  {
    float t=0;
    float Y_MT;
    float perenos_z = 6.0f;
    float shirina_vectora = 0.009f;
    float dlina_vectora = 0.3f;
    float dlina_osi = 3f;
    float shirina_CB = 0.04f;
    float shirina_osi = 0.1f;
    public static float dop_dlina_po_AB;
    public static float dlina_CB = 1.5f;
    public static float ugol = 5;
    public static float a = 1.02f;
    public static float b = 1.0f;
    int naprav=1;
    bool flag_time = false;
    List<CustomVertex.PositionColored> MT_List =
new List<CustomVertex.PositionColored>();
    List<CustomVertex.PositionColored> We_List =
new List<CustomVertex.PositionColored>();
    List<CustomVertex.PositionColored> Wr_List =
new List<CustomVertex.PositionColored>();
    List<CustomVertex.PositionColored> Ve_List =
new List<CustomVertex.PositionColored>();
    List<CustomVertex.PositionColored> Vr_List =
new List<CustomVertex.PositionColored>();
    Device d3d;
    Mesh M_tochka;
    Mesh os;
    Mesh cilindr_gorisontal;
    Mesh cilindr_vertical;
    Mesh plastina;
    Mesh verhn_krep;
    Mesh nign_krep;
    Mesh podshipnik;
    Mesh vector;
    Mesh vectorST;
    Material M_tochkaMaterial;
    Material osMaterial;
    Material cilindr_gorisontalMaterial;
    Material plastinaMaterial;
    Material krepMaterial;
    Material podshipnikMaterial;
    Material vectorMaterial;
    public Form1()
    {
      InitializeComponent();
      d3d = null;
      this.SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.Opaque, true);
      os = null;
      cilindr_gorisontal = null;
      cilindr_vertical = null;
      plastina = null;
      M_tochka = null;
      vector = null;
      vectorST = null;
      podshipnik = null;
      verhn_krep = null;
      nign_krep = null;
```

```
    }
    private void Form1_Load(object sender, EventArgs e)
    {
      начатьToolStripMenuItem.Enabled = false;
      начатьToolStripMenuItem.BackColor = Color.OrangeRed;
      остановитьToolStripMenuItem.Enabled = false;
      остановитьToolStripMenuItem.BackColor = Color.OrangeRed;
      ВектораToolStripMenuItem.Enabled = false;
      ВектораToolStripMenuItem.BackColor = Color.OrangeRed;
      стеретьToolStripMenuItem1.Enabled = false;
      стеретьToolStripMenuItem1.BackColor = Color.OrangeRed;
      траекторияToolStripMenuItem.Enabled = false;
      траекторияToolStripMenuItem.BackColor = Color.OrangeRed;
      dop_dlina_po_AB = 1.02f;
          a = dop_dlina_po_AB;
      try
      {
        PresentParameters d3dpp = new PresentParameters();
        d3dpp.BackBufferCount = 1;
        d3dpp.SwapEffect = SwapEffect.Discard;
        d3dpp.Windowed = true;
        d3dpp.MultiSample = MultiSampleType.None;
        d3dpp.EnableAutoDepthStencil = true;
        d3dpp.AutoDepthStencilFormat = DepthFormat.D16;
        d3d = new Device(0, DeviceType.Hardware, this, CreateFlags.SoftwareVertexProcessing,
d3dpp);
      }
        catch(Exception exc)
      {MessageBox.Show(this, exc.Message, " Помилка ініціалізації ");
        Close();
        }
        os = Mesh.Cylinder(d3d, 0.05f, 0.05f, dlina_osi, 10, 25);
        osMaterial = new Material();
        osMaterial.Diffuse = Color.Gray;
        osMaterial.Specular = Color.White;
        cilindr_gorisontal = Mesh.Cylinder(d3d, shirina_CB, shirina_CB, dlina_CB, 10, 25);
        cilindr_gorisontalMaterial = new Material();
        cilindr_gorisontalMaterial.Diffuse = Color.YellowGreen;
        cilindr_gorisontalMaterial.Specular = Color.White;
        cilindr_vertical = Mesh.Cylinder(d3d, shirina_CB, shirina_CB, dlina_osi * 3 / 4 +
shirina_CB, 10, 25);

        plastina = Mesh.Box(d3d, dlina_CB - 11 * 0.027f * dlina_CB, dlina_osi * 3 / 4 - 0.027f *
dlina_CB, 0.027f * dlina_CB / 2);
        plastinaMaterial = new Material();
        plastinaMaterial.Diffuse = Color.Gold;
        plastinaMaterial.Specular = Color.White;
        M_tochka = Mesh.Sphere(d3d, 2 * shirina_CB, 15, 15);
        M_tochkaMaterial = new Material();
        M_tochkaMaterial.Diffuse = Color.SkyBlue;
        M_tochkaMaterial.Specular = Color.White;
        verhn_krep = Mesh.Cylinder(d3d, shirina_osi, 0.1f*shirina_osi, 3*shirina_osi, 15, 15);
        nign_krep = Mesh.Box(d3d, 3*shirina_osi, 1.5f*shirina_osi, 3*shirina_osi);
        krepMaterial = new Material();
        krepMaterial.Diffuse = Color.Gold;
        krepMaterial.Specular = Color.White;
        podshipnik = Mesh.Sphere(d3d, shirina_osi,15, 15);
        podshipnikMaterial = new Material();
        podshipnikMaterial.Diffuse = Color.White;
        podshipnikMaterial.Specular = Color.White;
        vector = Mesh.Cylinder(d3d, shirina_vectora, shirina_vectora, dlina_vectora, 25, 25);
        vectorST = Mesh.Cylinder(d3d, 2.5f * (shirina_vectora + 0.001f), 0.1f * shirina_vectora,
4 * (shirina_vectora + 0.001f), 25, 25);
        vectorMaterial = new Material();
        vectorMaterial.Diffuse = Color.Black;
        vectorMaterial.Specular = Color.White;
    d3d.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Green, 1.0f, 0);
        d3d.Present();
      }
    private void SetupProekcii()
    { d3d.Lights[0].Enabled = true;
      d3d.Lights[0].Diffuse = Color.White;
      d3d.Lights[0].Position = new Vector3(0, 0, 0);
      d3d.Transform.Projection =
        Matrix.PerspectiveFovLH((float)Math.PI / 4, this.Width / this.Height, 1.0f, 50.0f);
    }
    private void Picture()
    {
```

```
      if (flag_time == true)
      {
        t=t+0.01f;
        skor_tB.Text = Convert.ToString(Skorost(t));
        uskor_tB.Text = Convert.ToString(Uskorenie(t));
        time_tB.Text = Convert.ToString(t);
      }
      flag_time = false;
      Y_MT = Координата_Y_MT(t);
    d3d.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Green, 1.0f, 0);
        d3d.BeginScene();
        SetupProekcii();
        skor_tB.Text =Convert.ToString(Skorost(t));
        uskor_tB.Text = Convert.ToString(Uskorenie(t));
        time_tB.Text = Convert.ToString(t);
      if (сплошнойToolStripMenuItem.Checked == true)
      { d3d.RenderState.FillMode = FillMode.Solid; }
      if (каркасныйToolStripMenuItem.Checked == true)
      { d3d.RenderState.FillMode = FillMode.WireFrame; }
      d3d.Transform.World = Matrix.RotationX((float)Math.PI / 2) * Matrix.Translation(0, 0,
perenos_z);
      d3d.Material = osMaterial;
      os.DrawSubset(0);

      d3d.Transform.World = Matrix.RotationY((float)Math.PI / 2) * Matrix.Translation(dlina_CB /
2, dlina_osi * 3 / 8, 0)
                          * Matrix.RotationY(ugol * t)
                          * Matrix.Translation(0, 0, perenos_z);
      d3d.Material = cilindr_gorisontalMaterial;
      cilindr_gorisontal.DrawSubset(0);
      d3d.Transform.World = Matrix.RotationY((float)Math.PI / 2) * Matrix.Translation(dlina_CB /
2, -dlina_osi * 3 / 8, 0)
                          * Matrix.RotationY(ugol * t)
                          * Matrix.Translation(0, 0, perenos_z);
      cilindr_gorisontal.DrawSubset(0);
      d3d.Transform.World = Matrix.RotationX((float)Math.PI / 2) * Matrix.Translation(dlina_CB,
0, 0)
                          * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
      cilindr_vertical.DrawSubset(0);
      d3d.Transform.World = Matrix.Translation((dlina_CB - 10 * 0.027f * dlina_CB) / 2, 0, 0)
                          * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
      d3d.Material = plastinaMaterial;
      plastina.DrawSubset(0);
      d3d.Transform.World = Matrix.Translation(dlina_CB, 0, 0) * Matrix.Translation(0, Y_MT, 0)
                          * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
      d3d.Material = M_tochkaMaterial;
      M_tochka.DrawSubset(0);
      d3d.Transform.World = Matrix.RotationX((float)Math.PI * 3 / 2) * Matrix.Translation(0,
dlina_osi / 2, 0)
                          * Matrix.Translation(0, 0, perenos_z);
      d3d.Material = krepMaterial;
      verhn_krep.DrawSubset(0);
      d3d.Transform.World = Matrix.Translation(0, -dlina_osi / 2, 0)
                         * Matrix.Translation(0, 0, perenos_z);
      nign_krep.DrawSubset(0);
      d3d.Transform.World = Matrix.Translation(0, -dlina_osi / 2 + shirina_osi, 0)
                         * Matrix.Translation(0, 0, perenos_z);
      podshipnik.DrawSubset(0);
      if (траекторияМТToolStripMenuItem.Checked == true)
      {
        float x = Convert.ToSingle(dlina_CB * Math.Cos(-ugol * t));
        float y = Convert.ToSingle(Y_MT + (dlina_osi * 3 / 4 + dlina_osi / 8.0f + 4 * shirina_CB)
/ 2);
        float z = Convert.ToSingle(dlina_CB * Math.Sin(-ugol * t));
        CustomVertex.PositionColored one = new CustomVertex.PositionColored();
        one.Position = new Vector3(x, y, z);
        MT_List.Add(one);
        d3d.VertexFormat = CustomVertex.PositionColored.Format;
        CustomVertex.PositionColored[] verts = new CustomVertex.PositionColored[MT_List.Count];
        vectorMaterial.Diffuse = Color.AliceBlue;
        vectorMaterial.Specular = Color.White;
        d3d.Material = vectorMaterial;
        for (int i = 0; i < MT_List.Count; i++)
          verts[i] = MT_List[i];
    d3d.DrawUserPrimitives(PrimitiveType.LineStrip, verts.Length - 1, verts);
      }
      if (траекторияМТToolStripMenuItem.Checked == false)
      { MT_List.Clear(); }
```

```
        //Годограф We
      if (годографWeToolStripMenuItem.Checked == true)
      {
        float x = Convert.ToSingle((dlina_CB - dlina_vectora - 4 * (shirina_vectora + 0.001f) /
2) * Math.Cos(-ugol * t));
        float y = Convert.ToSingle(Y_MT + (dlina_osi * 3 / 4 + dlina_osi / 8.0f + 4 * shirina_CB)
/ 2);
        float z = Convert.ToSingle((dlina_CB - dlina_vectora) /*- (shirina_vectora + 0.001f) /
2)*/ * Math.Sin(-ugol * t));
        CustomVertex.PositionColored one = new CustomVertex.PositionColored();
        one.Position = new Vector3(x, y, z);
        We_List.Add(one);
        d3d.VertexFormat = CustomVertex.PositionColored.Format;
        CustomVertex.PositionColored[] verts = new CustomVertex.PositionColored[We_List.Count];
        vectorMaterial.Diffuse = Color.Black;
        vectorMaterial.Specular = Color.White;
        d3d.Material = vectorMaterial;
        for (int i = 0; i < We_List.Count; i++)
          verts[i] = We_List[i];
    d3d.DrawUserPrimitives(PrimitiveType.LineStrip, verts.Length - 1, verts);
      }
      if (годографWeToolStripMenuItem.Checked == false)
      { We_List.Clear(); }
        //Годограф Wr
      if (годографWrToolStripMenuItem.Checked == true)
      {  float k = -a * b * b * (float)Math.Sin(b * t);
          if (k > 0)
          { naprav = 1; }
          if (k < 0)
          { naprav = -1; }
        float x = Convert.ToSingle(dlina_CB * Math.Cos(-ugol * t));
        float y = Convert.ToSingle(Y_MT + naprav * dlina_vectora - 0.5f * shirina_vectora
                                   + (dlina_osi * 3 / 4 + dlina_osi / 8.0f + 4 * shirina_CB + 4 *
(shirina_vectora + 0.001f)) / 2);
        float z = Convert.ToSingle(dlina_CB * Math.Sin(-ugol * t));
        CustomVertex.PositionColored one = new CustomVertex.PositionColored();
        one.Position = new Vector3(x, y, z);
        Wr_List.Add(one);

        d3d.VertexFormat = CustomVertex.PositionColored.Format;
        CustomVertex.PositionColored[] verts = new CustomVertex.PositionColored[Wr_List.Count];
        for (int i = 0; i < Wr_List.Count; i++)
          verts[i] = Wr_List[i];
   d3d.DrawUserPrimitives(PrimitiveType.LineStrip, verts.Length - 1, verts);
      }
      if (годографWrToolStripMenuItem.Checked == false)
      { Wr_List.Clear(); }
        //Годограф Ve
      if (годографVeToolStripMenuItem.Checked == true)
      {
        float x = Convert.ToSingle(dlina_CB * Math.Cos(-ugol * t)
                                   + (dlina_vectora - 0.5f * shirina_CB) * Math.Sin(-ugol *
t));
        float y = Convert.ToSingle(Y_MT + (dlina_osi * 3 / 4 + dlina_osi / 8.0f + 4 * shirina_CB
+ 4 * (shirina_vectora + 0.001f)) / 2);
        float z = Convert.ToSingle(dlina_CB * Math.Sin(-ugol * t) - (dlina_vectora
                                    - 0.5f * shirina_CB) * Math.Cos(-ugol * t));
        CustomVertex.PositionColored one = new CustomVertex.PositionColored();
        one.Position = new Vector3(x, y, z);
        Ve_List.Add(one);
        d3d.VertexFormat = CustomVertex.PositionColored.Format;
        CustomVertex.PositionColored[] verts = new CustomVertex.PositionColored[Ve_List.Count];
        for (int i = 0; i < Ve_List.Count; i++)
          verts[i] = Ve_List[i];
    d3d.DrawUserPrimitives(PrimitiveType.LineStrip, verts.Length - 1, verts);
      }
      if (годографVeToolStripMenuItem.Checked == false)
      { Ve_List.Clear(); }

        //Годограф Vr
      if (годографVrToolStripMenuItem.Checked == true)
      {
          float k = a * b * (float)Math.Cos(b * t);
          if (k > 0)
          { naprav = 1; }
          if (k < 0)
          { naprav = -1; }
```

```
        float x = Convert.ToSingle(dlina_CB * Math.Cos(-ugol * t));
        float y = Convert.ToSingle(Y_MT + naprav * dlina_vectora - 0.5f * shirina_vectora
                                   + (dlina_osi * 3 / 4 + dlina_osi / 8.0f + 4 * shirina_CB + 4 *
(shirina_vectora + 0.001f)) / 2);
        float z = Convert.ToSingle(dlina_CB * Math.Sin(-ugol * t));
        CustomVertex.PositionColored one = new CustomVertex.PositionColored();
        one.Position = new Vector3(x, y, z);
        Vr_List.Add(one);
        d3d.VertexFormat = CustomVertex.PositionColored.Format;
        CustomVertex.PositionColored[] verts = new CustomVertex.PositionColored[Vr_List.Count];
        for (int i = 0; i < Vr_List.Count; i++)
          verts[i] = Vr_List[i];
    d3d.DrawUserPrimitives(PrimitiveType.LineStrip, verts.Length - 1, verts);
      }
      if (годографVrToolStripMenuItem.Checked == false)
      { Vr_List.Clear(); }
      //вектор We
      if (wnToolStripMenuItem.Checked == true)
      {
          d3d.Transform.World = Matrix.RotationY((float)Math.PI / 2) *
Matrix.Translation(dlina_CB - 0.5f * dlina_vectora, 0, 0)
                              * Matrix.Translation(0, Y_MT, 0)
                              * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
        vectorMaterial.Diffuse = Color.Black;
        vectorMaterial.Specular = Color.White;
        d3d.Material = vectorMaterial;
        vector.DrawSubset(0);

        d3d.Transform.World = Matrix.RotationY(-(float)Math.PI / 2) * Matrix.Translation(dlina_CB
- dlina_vectora, 0, 0)
                            * Matrix.Translation(0, Y_MT, 0)
                            * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
        vectorST.DrawSubset(0);
      }
      if (wnToolStripMenuItem.Checked == false)
      {
          We_List.Clear();
      }
      //вектор Wr
      if (wrToolStripMenuItem.Checked == true)
      {  float k = -a * b * b *(float)Math.Sin(b * t);
          if (k > 0)
          { naprav = 1; }
          if (k < 0)
          { naprav = -1; }
          d3d.Transform.World = Matrix.RotationX((float)Math.PI / 2) *
Matrix.Translation(dlina_CB, 0, 0)
                              * Matrix.Translation(0, naprav * 0.5f * dlina_vectora, 0)
                              * Matrix.Translation(0, Y_MT, 0)
                              * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
        vectorMaterial.Diffuse = Color.Black;
        vectorMaterial.Specular = Color.White;
        d3d.Material = vectorMaterial;
        vector.DrawSubset(0);
        d3d.Transform.World = Matrix.RotationX(3 * naprav *(float)Math.PI / 2) *
Matrix.Translation(dlina_CB, 0, 0)
                            * Matrix.Translation(0, naprav * dlina_vectora, 0) *
Matrix.Translation(0, Y_MT, 0)
                            * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
        vectorST.DrawSubset(0);
      }
      if (wrToolStripMenuItem.Checked == false)
      {
          Wr_List.Clear();
      }
      //вектор Vr
      if (vrToolStripMenuItem.Checked == true)
      {
        float k = a *b*(float) Math.Cos(b * t);
        if (k > 0)
        {  naprav=1;}
        if (k < 0)
        { naprav = -1; }
        d3d.Transform.World = Matrix.RotationX( (float)Math.PI / 2) *
Matrix.Translation(dlina_CB, 0, 0)
                            * Matrix.Translation(0, naprav * 0.5f * dlina_vectora, 0)
                            * Matrix.Translation(0, Y_MT, 0)
                            * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
```

```
        vectorMaterial.Diffuse = Color.Blue;
        vectorMaterial.Specular = Color.White;
        d3d.Material = vectorMaterial;
        vector.DrawSubset(0);
        d3d.Transform.World = Matrix.RotationX(3 * naprav *(float)Math.PI / 2) *
Matrix.Translation(dlina_CB, 0, 0)
                            * Matrix.Translation(0, naprav * dlina_vectora, 0) *
Matrix.Translation(0, Y_MT, 0)
                            * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
        vectorST.DrawSubset(0);
      }
      if (vrToolStripMenuItem.Checked == false)
      {
          Vr_List.Clear();
      }
      //вектор Ve
      if (veToolStripMenuItem.Checked == true)
      {
        d3d.Transform.World = Matrix.Translation(dlina_CB, 0, 0) * Matrix.Translation(0, Y_MT, 0)
                            * Matrix.Translation(0, 0, -0.5f * dlina_vectora)
                            * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
        vectorMaterial.Diffuse = Color.Blue;
        vectorMaterial.Specular = Color.White;
        d3d.Material = vectorMaterial;
        vector.DrawSubset(0);
        d3d.Transform.World = Matrix.RotationY((float)Math.PI) * Matrix.Translation(dlina_CB, 0,
0)
                            * Matrix.Translation(0, 0, -dlina_vectora - 0.5f * shirina_CB) *
Matrix.Translation(0, Y_MT, 0)
                            * Matrix.RotationY(ugol * t) * Matrix.Translation(0, 0, perenos_z);
        vectorST.DrawSubset(0);
      }
      if (veToolStripMenuItem.Checked == false)
      {
          Ve_List.Clear();
      }
      d3d.EndScene();
      d3d.Present();
    }
    private void Ochis()
    {
     d3d.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Green, 1.0f, 0);
        d3d.BeginScene();
        SetupProekcii();
        d3d.EndScene();
        d3d.Present();
        skor_tB.Text = "";
        uskor_tB.Text = "";
        time_tB.Text = "";
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
      flag_time = true;
      Picture();
    }
    public void OnIdle(object sender, EventArgs e)
    {
      Invalidate();
    }
    private void начатьToolStripMenuItem_Click(object sender, EventArgs e)
    {
      flag_time = true;
        нарисоватьToolStripMenuItem.Enabled = false;
        нарисоватьToolStripMenuItem.BackColor = Color.OrangeRed;
        геометрическиеToolStripMenuItem.Enabled = false;
        геометрическиеToolStripMenuItem.BackColor = Color.OrangeRed;
        кинематическиеToolStripMenuItem.Enabled = false;
        кинематическиеToolStripMenuItem.BackColor = Color.OrangeRed;
        начатьToolStripMenuItem.Enabled = false;
        начатьToolStripMenuItem.BackColor = Color.OrangeRed;
        остановитьToolStripMenuItem.Enabled = true;
        остановитьToolStripMenuItem.BackColor = Color.Lime;
        стеретьToolStripMenuItem1.Enabled = false;
        стеретьToolStripMenuItem1.BackColor = Color.OrangeRed;
        ВектораToolStripMenuItem.Enabled = true;
        ВектораToolStripMenuItem.BackColor = Color.Lime;
        траекторияToolStripMenuItem.Enabled = true;
        траекторияToolStripMenuItem.BackColor = Color.Lime;
```

```
        Ochis();
        timer1.Enabled = true;
    }
   private void остановитьToolStripMenuItem_Click(object sender, EventArgs e)
    {
      flag_time = false;
        стеретьToolStripMenuItem1.Enabled = true;
        стеретьToolStripMenuItem1.BackColor = Color.Lime;
        кинематическиеToolStripMenuItem.Enabled=true;
        кинематическиеToolStripMenuItem.BackColor = Color.Lime;
        остановитьToolStripMenuItem.Enabled=false;
        остановитьToolStripMenuItem.BackColor = Color.OrangeRed;
        начатьToolStripMenuItem.Enabled=true;
        начатьToolStripMenuItem.BackColor = Color.Lime;
        ВектораToolStripMenuItem.Enabled = true;
        ВектораToolStripMenuItem.BackColor = Color.Lime;
                timer1.Enabled=false;
    }
    private void геометрическиеToolStripMenuItem_Click(object sender, EventArgs e)
    { MT_List.Clear();
      We_List.Clear();
      Wr_List.Clear();
      Vr_List.Clear();
      Ve_List.Clear();
      Form3 forma3 = new Form3();
      forma3.ShowDialog();
      plastina.Dispose();
      plastina = Mesh.Box(d3d, dlina_CB - 11 * 0.027f * dlina_CB, dlina_osi * 3 / 4 - 0.027f *
dlina_CB, 0.027f * dlina_CB / 2);
      os.Dispose();
      os = Mesh.Cylinder(d3d, 0.05f, 0.05f, dlina_osi, 10, 25);
      cilindr_gorisontal.Dispose();
      cilindr_gorisontal = Mesh.Cylinder(d3d, shirina_CB, shirina_CB, dlina_CB, 10, 25);
      cilindr_vertical.Dispose();
      cilindr_vertical = Mesh.Cylinder(d3d, shirina_CB, shirina_CB, dlina_osi * 3 / 4 +
shirina_CB, 10, 25);
      M_tochka.Dispose();
      M_tochka = Mesh.Sphere(d3d, 2 * shirina_CB, 15, 15);
      verhn_krep.Dispose();
      verhn_krep = Mesh.Cylinder(d3d, shirina_osi, 0.1f * shirina_osi, 3 * shirina_osi, 15, 15);
      nign_krep.Dispose();
      nign_krep = Mesh.Box(d3d, 3 * shirina_osi, 1.5f * shirina_osi, 3 * shirina_osi);
      podshipnik.Dispose();
      podshipnik = Mesh.Sphere(d3d, shirina_osi, 15, 15);

    }
    private void кинематическиеToolStripMenuItem_Click(object sender, EventArgs e)
    { MT_List.Clear();
      We_List.Clear();
      Wr_List.Clear();
      Vr_List.Clear();
      Ve_List.Clear();
      Form2 forma2 = new Form2();
      forma2.ShowDialog();
    }
    private float Координата_Y_MT(float t1)
    {
        float y = 0;
      y = a * (float)Math.Sin(b *  t1);
      return y;
    }
    private void wnToolStripMenuItem_Click(object sender, EventArgs e)
    { We_List.Clear();
      flag_time = false;
      wnToolStripMenuItem.Checked = !wnToolStripMenuItem.Checked;
      if (wnToolStripMenuItem.Checked == true)
      {
          годографWeToolStripMenuItem.Enabled = true;
          годографWeToolStripMenuItem.BackColor = Color.Lime;
      }
      if (wnToolStripMenuItem.Checked != true)
      {
          годографWeToolStripMenuItem.Checked = false;
          годографWeToolStripMenuItem.Enabled = false;
          годографWeToolStripMenuItem.BackColor = Color.OrangeRed;
      }
      Picture();
    }
```

```
private void wrToolStripMenuItem_Click(object sender, EventArgs e)
{ Wr_List.Clear();
  flag_time = false;
  wrToolStripMenuItem.Checked = !wrToolStripMenuItem.Checked;
  if (wrToolStripMenuItem.Checked == true)
  {
      годографWrToolStripMenuItem.Enabled = true;
      годографWrToolStripMenuItem.BackColor = Color.Lime;
  }
  if (wrToolStripMenuItem.Checked != true)
  {
      годографWrToolStripMenuItem.Checked = false;
      годографWrToolStripMenuItem.Enabled = false;
      годографWrToolStripMenuItem.BackColor = Color.OrangeRed;
  }
  Picture();
}
private void veToolStripMenuItem_Click(object sender, EventArgs e)
{ Ve_List.Clear();
  flag_time = false;
  veToolStripMenuItem.Checked = !veToolStripMenuItem.Checked;
  if (veToolStripMenuItem.Checked == true)
  {
      годографVeToolStripMenuItem.Enabled = true;
      годографVeToolStripMenuItem.BackColor = Color.Lime;
  }
  if (veToolStripMenuItem.Checked != true)
  {
      годографVeToolStripMenuItem.Checked = false;
      годографVeToolStripMenuItem.Enabled = false;
      годографVeToolStripMenuItem.BackColor = Color.OrangeRed;
  }
  Picture();
}
private void vrToolStripMenuItem_Click(object sender, EventArgs e)
{ Vr_List.Clear();
  flag_time = false;
  vrToolStripMenuItem.Checked = !vrToolStripMenuItem.Checked;
  if (vrToolStripMenuItem.Checked == true)
  {
      годографVrToolStripMenuItem.Enabled = true;
      годографVrToolStripMenuItem.BackColor = Color.Lime;
  }
  if (vrToolStripMenuItem.Checked != true)
  {
      годографVrToolStripMenuItem.Checked = false;
      годографVrToolStripMenuItem.Enabled = false;
      годографVrToolStripMenuItem.BackColor = Color.OrangeRed;
  }
  Picture();
}
private void сплошнойToolStripMenuItem_Click(object sender, EventArgs e)
{
  сплошнойToolStripMenuItem.Checked = true;
  каркасныйToolStripMenuItem.Checked = false;
    this.Invalidate();
}

private void каркасныйToolStripMenuItem_Click(object sender, EventArgs e)
{
  каркасныйToolStripMenuItem.Checked = true;
  сплошнойToolStripMenuItem.Checked = false;
    this.Invalidate();
}
private void выходToolStripMenuItem1_Click(object sender, EventArgs e)
{
  Close();
}
private void Form1_Paint(object sender, PaintEventArgs e)
{
  if (нарисоватьToolStripMenuItem.Enabled == false)
    Picture();
  else
  {
 d3d.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Green, 1.0f, 0);
    d3d.Present();
  }
  a_TB.Text = Convert.ToString(a);
```

```
      b_TB.Text = Convert.ToString(b);
      textBox1.Text = Convert.ToString(ugol);
    }
    private float Skorost(float t1)
    {
      return Convert.ToSingle(Math.Sqrt(Math.Pow((a * b * Math.Cos(b * t1)), 2)
                            + Math.Pow((dlina_CB * ugol), 2)));
    }
    private float Uskorenie(float t1)
    {
      return Convert.ToSingle(Math.Sqrt(Math.Pow(-a * b * b * Math.Sin(b * t1), 2)
                            + Math.Pow((dlina_CB * ugol * ugol), 2)));
    }
    private void Form1_Shown(object sender, EventArgs e)
    {
      a_TB.Text = Convert.ToString(a);
      b_TB.Text = Convert.ToString(b);
      textBox1.Text = Convert.ToString(ugol);
    }
    private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
    {
      Help.ShowHelp(this, @" D:\Diplom 2023\Сімченко К\Сімченко К 2023\HELPOVICH\Help_for_Diplom.hlp");
    }
    private void стеретьToolStripMenuItem1_Click(object sender, EventArgs e)
    { MT_List.Clear();
        We_List.Clear();
        Wr_List.Clear();
        Vr_List.Clear();
        Ve_List.Clear();
        Ochis();
        нарисоватьToolStripMenuItem.Enabled = true;
        нарисоватьToolStripMenuItem.BackColor = Color.Lime;
        начатьToolStripMenuItem.Enabled = false;
        начатьToolStripMenuItem.BackColor = Color.OrangeRed;
        стеретьToolStripMenuItem1.Enabled = false;
        стеретьToolStripMenuItem1.BackColor = Color.OrangeRed;
        ВектораToolStripMenuItem.Enabled = false;
        ВектораToolStripMenuItem.BackColor = Color.OrangeRed;
        геометрическиеToolStripMenuItem.Enabled = true;
        геометрическиеToolStripMenuItem.BackColor = Color.Lime;
        траекторияToolStripMenuItem.Enabled = false;
        траекторияToolStripMenuItem.BackColor = Color.OrangeRed;
    }
  private void нарисоватьToolStripMenuItem_Click(object sender, EventArgs e)
    {
        t = 0;
        Picture();
        стеретьToolStripMenuItem1.Enabled = true;
        стеретьToolStripMenuItem1.BackColor = Color.Lime;
        начатьToolStripMenuItem.Enabled = true;
        начатьToolStripMenuItem.BackColor = Color.Lime;
        остановитьToolStripMenuItem.Enabled = false;
        остановитьToolStripMenuItem.BackColor = Color.OrangeRed;
        нарисоватьToolStripMenuItem.Enabled = false;
        нарисоватьToolStripMenuItem.BackColor = Color.OrangeRed;
        ВектораToolStripMenuItem.Enabled = true;
        ВектораToolStripMenuItem.BackColor = Color.Lime;
        геометрическиеToolStripMenuItem.Enabled = false;
        геометрическиеToolStripMenuItem.BackColor = Color.OrangeRed;
    }
private void траекторияMTToolStripMenuItem_Click(object sender, EventArgs e)
    {       траекторияMTToolStripMenuItem.Checked = !траекторияMTToolStripMenuItem.Checked;
    }
  private void годографWeToolStripMenuItem_Click(object sender, EventArgs e)
    {        годографWeToolStripMenuItem.Checked = !годографWeToolStripMenuItem.Checked;
    }
  private void годографWrToolStripMenuItem_Click(object sender, EventArgs e)
    {        годографWrToolStripMenuItem.Checked = !годографWrToolStripMenuItem.Checked;
    }
  private void годографVeToolStripMenuItem_Click(object sender, EventArgs e)
    {        годографVeToolStripMenuItem.Checked = !годографVeToolStripMenuItem.Checked;
    }
  private void годографVrToolStripMenuItem_Click(object sender, EventArgs e)
    {        годографVrToolStripMenuItem.Checked = !годографVrToolStripMenuItem.Checked;
    }
    }  }
```

Файл Form2.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Dvigenie_MT
{    public partial class Form2 : Form
    {  public Form2()
        { InitializeComponent();        }
        private void button1_Click(object sender, EventArgs e)
        {if (ugol_UpD.Value==Convert.ToDecimal( 0.0))
          {MessageBox.Show("Помилка при зміні даних!");
            return;}
          Form1.ugol = Convert.ToSingle(ugol_UpD.Value);
          Form1.a = Convert.ToSingle(a_UpD.Value);
          Form1.b = Convert.ToSingle(b_UpD.Value);
        }
        private void Form2_Shown(object sender, EventArgs e)
        {
          a_UpD.Maximum = Convert.ToDecimal(Form1.dop_dlina_po_AB);
          a_UpD.Minimum = Convert.ToDecimal(-Form1.dop_dlina_po_AB);
          a_UpD.Value = Convert.ToDecimal(Form1.a);
          b_UpD.Value = Convert.ToDecimal(Form1.b);
          ugol_UpD.Value = Convert.ToDecimal(Form1.ugol);
        }
        private void ugol_UpD_ValueChanged(object sender, EventArgs e)
        {
            if (ugol_UpD.Value == Convert.ToDecimal(0.0))
                button1.BackColor = Color.OrangeRed;
            else
                button1.BackColor = Color.Lime;
        }    }}
```

Файл Form3.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Dvigenie_MT
{ public partial class Form3 : Form
    { public Form3()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        { Form1.dlina_CB=Convert.ToSingle(dlina_UpD.Value);
        }
        private void Form3_Shown(object sender, EventArgs e)
        {  dlina_UpD.Value = Convert.ToDecimal(Form1.dlina_CB);
        }
    }}
```