Форма № 7

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»

«Використання віртуальної лабораторії для розв'язання задач теорії удару»

(тема кваліфікаційної роботи українською мовою)

«The use of a virtual laboratory for solving the problems of impact theory»

(тема кваліфікаційної роботи англійською мовою)

Виконав: здобувач денної форми навчання
спеціальності 126 Інформаційні системи та технології
(код, назва спеціальності)
Освітня програма «Інформаційні системи та технології»
(назва)
Гордєєв Костянтин Юрійович
(прізвище, ім'я, по-батькові здобувача)

Керівник к.ф.-м.н., доц. Рачинська А.Л. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент Палій К.С.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 20___ р.

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

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/____/_____
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК
_____ _____
(підпис) (прізвище, ім'я)

Одеса 2024

АНОТАЦІЯ

Ця кваліфікаційна робота присвячена темі «Використання віртуальної лабораторії для вирішення задач теорії удару». Віртуальні лабораторії дають можливість моделювати різноманітні фізичні явища, проводити експерименти та підтверджувати гіпотези, які важко або неможливо перевірити в реальних умовах.

Метою роботи є оцінка ефективності застосування віртуальної лабораторії для моделювання ударних взаємодій, зокрема на прикладі абсолютно непружного зіткнення кулі з призмою з квадратичною основою. Для цього було розроблено програму мовою C# з використанням Windows Forms у середовищі Visual Studio. Програма дозволяє користувачам налаштовувати параметри експерименту, спостерігати за процесом зіткнення та аналізувати отримані результати.

Результати дослідження показали, що віртуальні лабораторії значно полегшують підготовку та проведення експериментів. Вони дозволяють виконувати велику кількість експериментів без додаткових витрат, сприяючи підвищенню якості досліджень та освітнього процесу. Використання віртуальних лабораторій також забезпечує гнучкість у керуванні умовами експериментів, що є важливим для глибокого розуміння складних фізичних процесів.

ABSTRACT

This qualification work is devoted to the topic “Using a virtual laboratory to solve problems of impact theory”. Virtual laboratories make it possible to simulate various physical phenomena, conduct experiments and confirm hypotheses that are difficult or impossible to test in real conditions.

The aim of this work is to evaluate the effectiveness of using a virtual laboratory to model impact interactions, in particular, on the example of a completely inelastic collision of a ball with a prism with a square base. For this purpose, a C# program was developed using Windows Forms in the Visual Studio environment. The program allows users to configure the parameters of the experiment, observe the collision process, and analyze the results.

The study results showed that virtual laboratories greatly facilitate the preparation and conduct of experiments. They allow performing a large number of experiments at no additional cost, contributing to the quality of research and the educational process. The use of virtual laboratories also provides flexibility in controlling the conditions of experiments, which is important for a deep understanding of complex physical processes.

ЗМІСТ

стор.

ВСТУП

Актуальність теми. Теорія удару є важливою частиною механіки з багатою історією, що охоплює кілька століть. Вона розвивалася паралельно з загальним прогресом у фізиці та математиці.

Сучасні дослідження теорії удару використовують як традиційні методи, так і передові технології. Традиційні підходи включають фізичні експерименти, які вимагають значних витрат ресурсів і часу. Ці експерименти потребують підготовки спеціалізованого обладнання, точного налаштування апаратури та ретельної обробки даних. Наприклад, краш-тести автомобілів проводяться в реальних умовах для оцінки безпеки пасажирів, що вимагає створення реалістичних умов і використання дорогих манекенів із датчиками.

На противагу цьому, віртуальні лабораторії відкривають нові можливості для досліджень у теорії удару. Вони дозволяють проводити високоточні симуляції в реальному часі без значних витрат на обладнання та матеріали. Такі симуляції можуть точно моделювати поведінку матеріалів і конструкцій при ударах, дозволяючи дослідникам швидко і ефективно аналізувати різні сценарії. Це особливо важливо в аерокосмічній галузі, де тестування на реальних зразках може бути надзвичайно дорогим і складним.
Віртуальні експерименти мають численні переваги:

- Економічність: зниження витрат на матеріали та обладнання.
- Швидкість: можливість проведення великої кількості експериментів за короткий час.
- Безпека: відсутність ризиків, пов'язаних із фізичними експериментами.
- Точність: здатність моделювати складні фізичні процеси з високою точністю.

Однак віртуальні лабораторії також мають певні недоліки:

- Залежність від моделей: точність симуляцій залежить від якості математичних моделей.
- Необхідність порівняння результатів: доводиться порівнювати віртуальні дані з реальними експериментами для підтвердження їх достовірності.
- Високі вимоги до обчислювальних ресурсів: складні симуляції можуть вимагати потужних комп'ютерів та спеціалізованого програмного забезпечення.

У цій роботі проводиться аналіз переваг та недоліків віртуальних експериментів. Також наведено приклад розв'язання конкретної задачі за допомогою розроблених та вбудованих інструментів віртуальної лабораторії. Це демонструє, як віртуальні технології можуть ефективно доповнювати традиційні методи, забезпечуючи більш ефективні та доступні дослідження в галузі теорії удару.

Мета та задачі дослідження.

Метою роботи є розробка та реалізація програми, що моделює зіткнення сфери з призмою та аналіз їх взаємодії.

Перед нами представлено декілька задач:

- Розробити математичну модель рухів сфери та призми.
- Розробити алгоритм, який моделює непружне зіткнення з призмою з урахуванням законів механіки.
- Реалізувати графічний інтерфейс для зображення зіткнення і подальшого руху об’єктів.
- Провести декілька досліджень змінюючи швидкість кулі, довжину ребра квадратної основи призми та інших параметрів.
- Визначити мінімальну швидкість кулі при якій призма зможе перевернутись.

1 ІСНУЮЧІ МОДЕЛІ РУХІВ СИСТЕМИ МАТЕРІАЛЬНИХ ТОЧОК В ТЕОРІЇ УДАРУ

1.1 Огляд літератури по темі роботи

У природі нерідко спостерігаються такі явища, за яких окремі матеріальні точки системи раптово змінюють свої швидкості за дуже короткий проміжок часу, у той час як вся система за цей проміжок часу майже не змінює свого становища. Такі явища називаються ударом. Вони викликають дуже великими силами, що діють на точки системи протягом короткого проміжку часу.

Розглянемо спочатку явище удару для однієї матеріальної точки, масу якої позначимо через m. Рівняння запишемо у векторному вигляді

$$m\frac{d\mathbf{V}}{dt}=\mathbf{F}$$

Під дією великих за величиною сил точка отримає велике прискорення Розглядаючи сили, що діють на точку як функції часу, проінтегруємо рівняння руху. Отримаємо закон зміни кількості руху точки

$$m\mathbf{V}_1-m\mathbf{V}_0=\int_{t_\partial}^{t_1}\mathbf{F}dt$$

де $\mathbf{V}_1$ - швидкість точки в момент t, $m\mathbf{V}_0$ - швидкість точки в момент t_0. Вираз, що стоїть у правій частині отриманій рівності називається імпульсом сили за час t_1-t_0. Якщо інтервал часу t_1-t_0 дуже малий, а чинні на точку сили кінцеві, то інтеграл правої частини буде величиною малої, порядку t_1-t_0, а зміна кількості руху виявляється мало помітною за час t_1-t_0. Так

відбуваються всі безперервні процеси Якщо ж сила **F** виявляється досить великою, порядку $\frac{1}{t_1 - t_0}$, то інтеграл правої частини буде величиною кінцевої, і за малий проміжок часу відбудеться помітна зміна кількості руху, тобто відбудеться явище удару.

Якщо через г позначити радіус-вектор точки, то матимемо

$$\mathbf{r}_1 - \mathbf{r}_0 = \int_{t_0}^{t_1} \mathbf{V} dt$$

звідки випливає, що при кінцевій зміні швидкості за час $t_1 - t_0$ інтеграл правої частини є величиною малої, і їм можна знехтувати щодо малих рухів точки.

У всіх випадках, коли проміжок часу $t_1 - t_0$ можна роздивитись практично як нескінченно малий і коли сила, що на точку за цей час, має порядок $\frac{1}{t_1 - t_0}$, будемо казати, що відбувається явище удару, а величину

$$\mathbf{P} = \int_{t_0}^{t_1} \mathbf{F} dt$$

називатимемо ударом. Рівняння

$$m\mathbf{V}_1 - m\mathbf{V}_0 = \mathbf{P}$$

дозволяє вимірювати удар по виробленому їм ефекту. Його можна переписати у вигляді

$$\Delta\left(m\mathbf{V}_1\right) = \mathbf{P}\,.$$

Надалі це рівняння називатимемо основним рівнянням теорії удару для однієї матеріальної точки.

Припускатимемо, що рух матеріальної точки підпорядковується неутримуючому зв'язку

$$f\left(x,y,z,t\right)\le 0,$$

що знаходиться в даний момент у ненатягнутому стані, і що в деякий момент часу t_0 частка потрапляє на цю зв'язок, так що виконується умова

$$f\left(x,y,z,t_0\right)=0,$$

і момент t_0 визначається як найменший додатній корінь рівняння

$$f\left(x(t),y(t),z(t),t\right)=0,$$

де функції $x(t), y(t), z(t)$ визначаються законом руху точки до удару. У момент удару швидкість точки не може мати довільного значення, а задовольняє умові

$$\frac{df}{dt}\le 0,$$

або

$$\frac{df}{dx}x'+\frac{df}{dy}y'+\frac{df}{dz}z'\le 0,$$

де x', y', z' - проекції швидкості точки на нерухомі осі координат.

Нехай швидкість $\mathbf{V}_0$ у момент t_0 задовольняє умові

$$\left.\frac{df}{dt}\right|_{t_0} = 0,$$

тоді, коли всі вищі похідні задовольняють умові

$$\left.\frac{d^k f}{dt^k}\right|_{t_0} = 0,$$

то матеріальна точка після удару рухатиметься у зв'язку. Якщо а хоча б одна з похідних відмінна від нуля, і, отже, негативна, після удару точка знову залишає зв'язок.

Якщо в момент удару t_0 виконується умова

$$\left.\frac{df}{dt}\right|_{t_0} > 0,$$

то в момент удару зв'язок вплине на точку. Ми будемо говорити, що на точку діє ударна реакція зв'язку $\mathbf{R}$, що у момент часу t_0 змінює швидкість $\mathbf{V}_0$ на $\mathbf{V}_1$ яка задовольняє умові

$$\left.\frac{df}{dt}\right|_{t_0+\tau} \le 0.$$

При цьому передбачається, що: 1) час τ дії сили реакції $\mathbf{R}$ нескінченно малий; 2) за час τ удару матеріальна точка і поверхню не встигають змінити свого становища; 3) за час удару імпульс будь-якої кінцевої сили дорівнює нулю.

Зроблені припущення представляють ідеалізацію дійсного руху. Насправді за час удару поверхні тіл, що соударяються, деформуються і

вступають в дію ударні сили. Але процес деформації протікає дуже швидко і для спрощення ми змушені запроваджувати гіпотезу миттєвого удару.

При зроблених припущеннях матимемо

$$\left.\frac{df}{dt}\right|_{t=t_0} > 0 \qquad \left.\frac{df}{dt}\right|_{t=t_0+\tau} \le 0,$$

і якщо вважати, що за час τ швидкість матеріальної точки змінюється безперервно, то існуватиме момент t_1:

$$t_0 < t_1 \le t_0 + \tau ,$$

в який виконується рівність

$$\left.\frac{df}{dt}\right|_{t=t_1} = 0 .$$

Тому удар можна поділити на дві стадії. Перша відбувається за час (t_0, t_1) - стиск, друга за час $(t_1, t_0 + \tau)$ - відбиток. Удар називають абсолютно непружним, якщо друга стадія відсутня і $\tau = t_1 - t_0$. Інакше удар називають пружним. Швидкість $\mathbf{V}_0$, з якою матеріальна точка приходить у дотикання зі зв'язком, називають швидкістю падіння матеріальною точки, а швидкість $\mathbf{V}_1$ - швидкістю відображення. Кут α між отриманим напрямком швидкості $\mathbf{V}_0$ і нормаллю до поверхні зв'язку називають кутом падіння, а кут β між нормаллю і напрямом швидкості $\mathbf{V}_1$ - кутом відбиття.

Зв'язок, що накладається на матеріальну точку, називають ідеальної, якщо робота ударної реакції $\mathbf{R}$ на будь-якому можливому Переміщення

точки дорівнює нулю. Нехай t деякий момент часу у проміжку t_0 та $t_0 + \tau$. Тоді

$$\Delta(m\mathbf{V}) = \int_{t_\partial}^{t_1} \mathbf{R}dt = \mathbf{n}^0 \int_{t_\partial}^{t_1} \mathbf{R}dt ,$$

де $\mathbf{n}^0$ - одиничний вектор нормалі до поверхні в точці зіткнення. Звідси випливає, що збільшення швидкості за час удару завжди спрямовано колінеарно з додатною нормаллю до поверхні в точці зіткнення, а швидкість падіння та швидкість відображення розташовані у площині, нормальної до поверхні $f(x, y, z, t) = 0$ і

$$V_0 \sin\alpha = V_1 \sin\beta .$$

В механіці розрізняють пружний і непружні удари. Непружним називають такий удар, при якому матеріальна точка як би прилипає до зв'язку і після удару не залишає поверхні зв'язку. При пружному ударі крапки після удару звільняються від зв'язку. На практиці частіше доводиться зустрічатися з явищами не цілком пружного удару, при якому відбувається втрата енергії і тіла, що ударяются, не повністю відновлюють свою форму. При розрахунку явищ удару для таких тіл приходиться вводити досвідчені гіпотези. Одна з основних таких гіпотез була введена Ньютоном, який припустив, що при зіткненні тіл відношення величин проекцій швидкостей точок після і до удару на напрямок загальної нормалі до поверхні тіл, що ударяються в точці дотику цих тіл, є величина постійна, яка залежить лише від матеріалу соударядних тіл.

Якщо через V_{in} позначити проекції швидкості точки на нормаль до поверхні зв'язку, а через $V_{i\tau}$ проекції швидкості на дотичну площу до поверхні зв'язку, то відношення

$$V_{in} : V_{0\tau} = e$$

називають коефіцієнтом відновлення. Коефіцієнт відновлення показує, наскільки відновлюється нормальна складова швидкості після удару. Якщо $e = 1$, то удар називають абсолютно пружним. У загальному випадку

$$0 \le e \le 1.$$

При гладких зв'язках дотична складова швидкості точки за час удару не змінюється. Але внаслідок неповної гладкості буде змінюватись і ця складова. Тому можна ввести коефіцієнт λ миттєвого тертя, так що

$$V_{i\tau} : V_{0\tau} = 1 - \lambda$$

Коефіцієнти e і λ зазвичай визначаються досвідченим шляхом. Іноді приймається гіпотеза Кулона, тобто робиться припущення, що дотичний удар пов'язаний із нормальним. Як показує експериментальна перевірка, вплив тертя на удар значно менше, ніж вплив пружності.

Розглянемо систему матеріальних точок $m_\nu \left(x_\nu, y_\nu, z_\nu \right)$, переміщення яких стиснуті голономними зв'язками. Зв'язки у відповідності з однією з основних аксіом механіки можна замінити силами, які діють на точки системи. Під дією ударних сил виникатимуть дуже великі реакції. Позначимо через $\mathbf{R}$ вектор реактивного удару, що діє на матеріальну точку, з проекціями R_x, R_y, R_z, маючи на увазі під цим граничні значення імпульсів сил реакцій. Тоді для кожної точки системи рівняння удару можна записати у вигляді

$$\Delta m_i \mathbf{V}_i = \mathbf{P}_i + \mathbf{R}_i .$$

Дотримуючись ідеї Лагранжа, введемо аксіому ідеальних зв'язків для удару. За визначенням, ідеальними називатимемо такі зв'язки, для яких сума робіт сил реактивного удару на будь-якому можливе переміщення системи дорівнює нулю, тобто

$$\sum_{\nu=1}^{n}\left(\mathbf{R}_\nu \delta\mathbf{r}_\nu\right)=0.$$

Підставляючи сюди значення реактивного удару, знайденого з основного рівняння удару для точки, отримаємо загальне рівняння ня теорії удару для системи матеріальних точок

$$\sum_{\nu=1}^{n}\left(\Delta m_\nu \mathbf{V}_\nu-\mathbf{P}_\nu\right)\delta\mathbf{r}_\nu=0,$$

або у проекціях

$$\sum_{\nu=1}^{n}\left\{\left(\Delta m_\nu x'_\nu-P_{x\nu}\right)\delta x_\nu+\left(\Delta m_\nu y'_\nu-P_{y\nu}\right)\delta y_\nu+\left(\Delta m_\nu z'_\nu-P_{z\nu}\right)\delta z_\nu\right\}=0.$$

Це рівняння має таке саме значення, як і загальне рівняння динаміки системи.

Якщо через $\mathbf{N}$ позначити силу реакції в деякі торій точці, то ця реакція буде нормальною до поверхні зв’язку, якщо відсутня тертя. За час удару тіла, що ударяються не переміщаються, тому за час удару не змінюється і напрям реакції. Звідси випливає, що під час удару виконується умова $\mathbf{N}\|\mathbf{R}$, а це означає, що ідеальні зв'язки залишаються ідеальними та для ударних реакцій.

За час удару зв'язку можуть зберігатися, але можуть і не зберігатися. Будемо називати зв'язок, що зберігається, якщо він існує під час удару та зберігається після удару. В цьому випадку дійсне переміщення буде допускатися зв'язками та після удару. Зв'язки називатимемо такими, що не

зберігаються, якщо вони існують під час удару, але зникають одразу після удару. Тоді дійсне переміщення, яке має місце після удару, не буде належати до переміщень, що допускаються цим зв'язком.

Розглянемо деякі наслідки із загального рівняння теорії удару системи матеріальних точок.

Нехай зв'язки, накладені на систему матеріальних точок, допускають поступальне переміщення всієї системи як одного цілого вздовж деякої нерухомої осі. Не порушуючи спільності, можна припускати, що зв'язки допускають поступальне переміщення всієї системи вздовж нерухомої осі x. Тоді серед усіх можливих переміщень системи буде переміщення, що задовольняє умовам

$$\delta x_\nu = \alpha,\ \delta y_\nu = 0,\ \delta z_\nu = 0$$

Підставляючи ці значення переміщень у загальне рівняння теорії удару, матимемо

$$\alpha \sum_{\nu=1}^{n} \left(\Delta m_\nu x'_\nu - P_{x\nu} \right) = 0.$$

Але так як

$$\sum_{\nu=1}^{n} \left(\Delta m_\nu x'_\nu \right) = \Delta \sum_{\nu=1}^{n} \left(m_\nu x'_\nu \right) = \sum_{\nu=1}^{n} \left(\Delta m_\nu x_\nu \right)' = \Delta \left(M \xi' \right),$$

де ξ - координата центру тяжіння по осі x. Отриманий результат можна уявити у вигляді

$$\Delta \left(M \xi' \right) = \sum_{\nu} P_{x\nu}$$

Якщо серед можливих переміщень системи є поступальне переміщення вздовж осі x, то зміна проекції кількості руху системи вздовж осі x дорівнює сумі проекцій ударів прикладених до точок системи, на цю вісь x.

Припустимо, що серед можливих переміщень системи є поворот всієї системи як одного цілого навколо нерухомої осі z. У цьому випадку серед можливих переміщень системи будуть переміщення

$$\delta x_\nu = -y_\nu \delta\varphi\,,\ \delta y_\nu = x_\nu \delta\varphi\,,\ \delta z_\nu = 0\,.$$

Підставляючи ці значення переміщень у загальне рівняння теорії удару, матимемо

$$\delta\varphi \sum_{\nu=1}^{n} \left\{ -\left(\Delta m_\nu x'_\nu - P_{x\nu}\right) y_\nu + \left(\Delta m_\nu y'_\nu - P_{y\nu}\right) x_\nu \right\} = 0\,,$$

що легко привести до вигляду

$$\sum_{\nu=1}^{n} \left\{ -\Delta\left(m_\nu x'_\nu\right) y_\nu + \Delta\left(m_\nu y'_\nu\right) x_\nu \right\} = \sum_{\nu=1}^{n} \left\{ -\left(P_{x\nu}\right) y_\nu + \left(-P_{y\nu}\right) x_\nu \right\}.$$

За час удару швидкості точок системи змінюються на кінцеву величину. Тому з рівняння

$$\frac{dx_\nu}{dt} = x'_\nu$$

отримаємо

$$x_{\nu 1} - x_{\nu 0} = \int_{t_0}^{t_0+\varepsilon} x'_\nu dt\,.$$

Але

$$\lim_{\varepsilon\to 0}\int_{t_0}^{t_0+\varepsilon} x'_\nu dt = 0,$$

тому що x'_ν - кінцева величина. Отже, протягом часу - мені удару має місце умова

$$x_{\nu_1} - x_{\nu_0} \to 0,$$

тобто координати точок системи не змінюються Приймаючи цю постановку до уваги, перепишемо отримане рівняння у вигляді

$$\sum_{\nu=1}^{n}\left\{\Delta\left(m_\nu y'_\nu x_\nu\right) - \Delta\left(m_\nu x'_\nu y_\nu\right)\right\} = \sum_{\nu=1}^{n}\left\{P_{y\nu}x_\nu - P_{x\nu}y_\nu\right\},$$

або

$$\Delta\sum_{\nu=1}^{n}\left\{m_\nu\left(y'_\nu x_\nu - x'_\nu y_\nu\right)\right\} = \sum_{\nu=1}^{n}\left\{P_{y\nu}x_\nu - P_{x\nu}y_\nu\right\}.$$

Позначаючи момент кількості руху системи щодо осі z через K_z

$$K_z = \sum_\nu m_\nu\left(y'_\nu x_\nu - x'_\nu y_\nu\right),$$

перепишемо рівняння

$$\Delta K_z = \sum_{\nu=1}^{n}\left\{P_{y\nu}x_\nu - P_{x\nu}y_\nu\right\}.$$

У результаті приходимо до наступного: Якщо серед можливих переміщень системи є поворот навколо нерухомої осі z, зміна моменту кількості руху системи щодо цієї осі за час удару дорівнює сумі моментів ударних імпульсів щодо осі z.

Розглянемо деякі наслідки з основного рівняння теорії удару, аналогічні теоремі живих сил. Будемо припускати, що на точки системи не діють зовнішні удари, тобто виконують умови

$$\mathbf{P}_z = 0 \qquad (\nu = 1,2,...,k).$$

Удар у разі відбувається лише з допомогою накладення чи зняття зв'язків.

Припустимо спочатку, що удар походить від раптового накладення зв'язків, які залежать явно від часу. Нехай, окрім того, що раптово накладаються на систему зв'язку залишаються у всьому подальший рух системи. Такі зв'язки називаються утриманими, або непружними, зв'язками. Серед можливих переміщень системи перебуватиме переміщення, сумісні зі зв'язками (зберігаються зв'язки). Оскільки зв'язки не залежать явно від часу, то серед можливих переміщень знаходяться дійсні переміщення, які будуть мати точки системи після удару, тобто переміщення, пропорційні швидкостям точок системи після удару:

$$\delta x_\nu = \theta x'_\nu,\ \delta y_\nu = \theta y'_\nu,\ \delta z_\nu = \theta z'_\nu.$$

Підставляючи ці значення основне рівняння теорії удару

$$\sum_{\nu=1}^{n}\left\{\left(\Delta m_\nu x'_\nu\right)\delta x_\nu + \left(\Delta m_\nu y'_\nu\right)\delta y_\nu + \left(\Delta m_\nu z'_\nu\right)\delta z_\nu\right\} = 0,$$

будемо мати

$$\sum_{\nu=1}^{n}\left\{\Delta\left(m_\nu x'_\nu\right)x'_\nu+\Delta\left(m_\nu y'_\nu\right)y'_\nu+\Delta\left(m_\nu z'_\nu\right)z'_\nu\right\}=0,$$

або

$$\sum_{\nu=1}^{n}m_\nu\left\{\left(x'_\nu-x'_{\nu0}\right)x'_\nu+\left(y'_\nu-y'_{\nu0}\right)y'_\nu+\left(z'_\nu-z'_{\nu0}\right)z'_\nu\right\}=0.$$

Після перетворень виду

$$\left(x'_\nu-x'_{\nu0}\right)x'_\nu=\frac{1}{2}\left(x'^2_\nu-2x'_{\nu0}x'_\nu+x'^{\,2}_{\nu0}\right)+\frac{1}{2}x'^2_\nu-\frac{1}{2}x'^{\,2}_{\nu0}=$$

$$=\frac{1}{2}\left(x'_\nu-x'_{\nu0}\right)^2+\frac{1}{2}x'^2_\nu-\frac{1}{2}x'^{\,2}_{\nu0}$$

отримаємо

$$\sum_{\nu=1}^{n}\left\{\frac{m_\nu}{2}\left(\left(x'_\nu-x'_{\nu0}\right)^2+\left(y'_\nu-y'_{\nu0}\right)^2+\left(z'_\nu-z'_{\nu0}\right)^2\right)+\right.$$

$$\left.+\sum\frac{m_\nu}{2}\left(x'^{\,2}_\nu+y'^{\,2}_\nu+z'^{\,2}_\nu\right)-\sum\frac{m_\nu}{2}\left(x'^{\,2}_{\nu0}+y'^{\,2}_{\nu0}+z'^{\,2}_{\nu0}\right)=0.\right.$$

Введемо поняття вектору втраченої швидкості

$$\mathbf{w}_\nu\left(x'_\nu-x'_{\nu0},y'_\nu-y'_{\nu0},z'_\nu-z'_{\nu0}\right).$$

Тоді отримане рівняння можна буде переписати у вигляді

$$\sum_\nu\frac{m_\nu W_\nu^2}{2}=T_0-T_1,$$

тут T_0-T_1 представляє зміну живої сили під час удару.

Маємо, що втрата живої сили системи під час накладення зв'язків дорівнює живій силі втрачених швидкостей.

Розглянемо випадок, коли на систему матеріальних точок накладені ідеальні, які не залежать явно від часу, голономні зв'язку. Припустимо, хто в деякий момент часу зв'язку раптово знімаються (наприклад, при вибуху снаряда, що летить). За час удару відбувається звільнення від зв'язків. На протязі часу удару можливі переміщення системи знаходяться в відповідно до накладених зв'язків. При цьому серед можливих переміщень перебувають і дійсні переміщення до удару (відповідні рівнянням зв'язку), тобто переміщення, пропорціональні швидкостям точок системи до удару:

$$\delta x_\nu = \theta x'_{\nu 0},\ \delta y_\nu = \theta y'_{\nu 0},\ \delta z_\nu = \theta z'_{\nu 0}.$$

Підставляючи ці значення в основне рівняння теорії удару, будемо мати

$$\theta \sum_{\nu=1}^{n} m_\nu \left\{ \left(x'_\nu - x'_{\nu 0} \right) x'_{\nu 0} + \left(y'_\nu - y'_{\nu 0} \right) y'_{\nu 0} + \left(z'_\nu - z'_{\nu 0} \right) z'_{\nu 0} \right\} = 0.$$

Перетворимо вирази виду

$$\left(x'_\nu - x'_{\nu 0} \right) x'_{\nu 0} = -\frac{1}{2}\left(x'^2_\nu - 2x'_{\nu 0} x'_\nu + x'^{\,2}_{\nu 0} \right) + \frac{1}{2} x'^2_{\nu 0} - \frac{1}{2} x'^{\,2}_{\nu 0} =$$

$$= -\frac{1}{2}\left(x'_\nu - x'_{\nu 0} \right)^2 + \frac{1}{2} x'^2_\nu - \frac{1}{2} x'^{\,2}_{\nu 0}.$$

Проробляючи аналогічні перетворення для координат y_ν та z_ν, будемо мати із загального рівняння динаміки

$$\sum_{\nu=1}^{n}\frac{m_\nu}{2}\left(\left(x'_\nu-x'_{\nu0}\right)^2+\left(y'_\nu-y'_{\nu0}\right)^2+\left(z'_\nu-z'_{\nu0}\right)^2\right)+$$

$$+\sum\frac{m_\nu}{2}\left(x'^{\,2}_\nu+y'^{\,2}_\nu+z'^{\,2}_\nu\right)-\sum\frac{m_\nu}{2}\left(x'^{\,2}_{\nu0}+y'^{\,2}_{\nu0}+z'^{\,2}_{\nu0}\right)=0.$$

або, вводячи вектор втраченої швидкості $\mathbf{w}$,

$$-\sum_{\nu=1}^{n}\frac{m_\nu w_\nu^2}{2}+\sum_{\nu=1}^{n}\frac{m_\nu v_\nu^2}{2}-\sum_{\nu=1}^{n}\frac{m_\nu v_{\nu0}^2}{2}=0.$$

Звідки

$$T_1-T_0=\sum_{\nu=1}^{n}\frac{m_\nu w_\nu^2}{2}.$$

Ми отримали: При звільненні системи, від зв'язків придбана жива сила дорівнює живій силі набутих швидкостей. Це застосовується лише до непружних систем.

Переходячи до вивчення не цілком пружного удару системи матеріальних точок, зауважимо, що швидкості точок системи до і після удару пов'язані співвідношеннями, які з гіпотези Ньютони:

$$e\mathbf{v}_{n0}=-\mathbf{v}_{n1},\ \mathbf{v}_{\tau0}=\mathbf{v}_{\tau1},$$

де $\mathbf{v}_{n0}$ і $\mathbf{v}_{n1}$ - нормальні складові вектору швидкості до і після удару; $\mathbf{v}_{\tau0}$ і $\mathbf{v}_{\tau1}$ - дотичні, що становлять швидкості до та після удару. Введемо вектор

$$\sigma=e\mathbf{v}_0+\mathbf{v}_1=e\mathbf{v}_{n0}+e\mathbf{v}_{\tau0}+\mathbf{v}_{n1}+\mathbf{v}_{\tau1}=$$
$$=-\mathbf{v}_{\tau1}+e\mathbf{v}_{\tau1}+\mathbf{v}_{n1}+\mathbf{v}_{\tau1}=\left(1+e\right)\mathbf{v}_{\tau1}=\left(1+e\right)\mathbf{v}_{\tau0},$$

розташований в спільній дотичній площині, що стикаються поверхонь. Якщо удар відбувається завдяки накладенню і зняття зв'язків, то серед можливих переміщень системи будуть знаходитися переміщення, сумісні з накладеними на час удару зв'язками, що незберігаються. Серед усіх переміщень будуть переміщення, пропорційні векторам σ_ν, тобто

$$\delta x_\nu = \theta\left(x'_\nu + e x'_{\nu 0}\right),\ \delta y_\nu = \theta\left(y'_\nu + e y'_{\nu 0}\right),$$
$$\delta z_\nu = \theta\left(z'_\nu + e z'_{\nu 0}\right)\ \left(\nu = 1,2,...,n\right),$$

де θ - коефіцієнт пропорційності. Підставляючи ці значення в загальне рівняння теорії удару, матимемо

$$\theta\sum_{\nu=1}^{n} m_\nu \left\{\left(x'_\nu - x'_{\nu 0}\right)\left(x'_\nu + e x'_{\nu 0}\right) + \left(y'_\nu - y'_{\nu 0}\right)\left(y'_\nu + e y'_{\nu 0}\right) + \left(z'_\nu - z'_{\nu 0}\right)\left(z'_\nu + e z'_{\nu 0}\right)\right\} = 0.$$

Перетворимо вираз

$$\left(x'_\nu - x'_{\nu 0}\right)\left(x'_\nu + e x'_{\nu 0}\right) = \frac{1}{2}\left(x'^2_\nu - 2x'_{\nu 0}x'_\nu + {x'_{\nu 0}}^2\right) + \frac{1}{2}x'^2_\nu - \frac{1}{2}{x'_{\nu 0}}^2 +$$
$$+e\left[-\frac{1}{2}\left(x'^2_\nu - 2x'_{\nu 0}x'_\nu + {x'_{\nu 0}}^2\right) + \frac{1}{2}x'^2_\nu - \frac{1}{2}{x'_{\nu 0}}^2\right] =$$
$$= \frac{1-e}{2}\left(x'_\nu - x'_{\nu 0}\right)^2 + \frac{1+e}{2}x'^2_\nu - \frac{1+e}{2}{x'_{\nu 0}}^2.$$

Після аналогічних перетворень для інших координат загальне рівняння теорії удару можна привести до вигляду

$$(1-e)\sum_{\nu=1}^{n}\frac{m_\nu}{2}\left(\left(x'_\nu - x'_{\nu 0}\right)^2 + \left(y'_\nu - y'_{\nu 0}\right)^2 + \left(z'_\nu - z'_{\nu 0}\right)^2\right) +$$
$$+(1+e)\sum\frac{m_\nu}{2}\left({x'_\nu}^2 + {y'_\nu}^2 + {z'_\nu}^2\right) - (1+e)\sum\frac{m_\nu}{2}\left({x'_{\nu 0}}^2 + {y'_{\nu 0}}^2 + {z'_{\nu 0}}^2\right) = 0.$$

або

$$(1-y)\sum_{\nu=1}^{n}\frac{m_\nu w_\nu^2}{2}+(1+e)T_1-(1+e)T_0=0,$$

тобто

$$T_0-T_1=\frac{1-e}{1+e}\sum_{\nu=1}^{n}\frac{m_\nu w_\nu^2}{2}.$$

Отримане рівняння визначає втрату кінетичної енергії при не цілком пружному ударі. При $e=1$ втрати живої сили не відбувається.

1.2 Методи дослідження руху тіл в теорії удару

Розглянемо методи розв'язання задачі про центр удару. Нехай тверде тіло може вільно обертатися навколо нерухомої вісі, закріпленої в точках O і O', відстань між якими дорівнює h. Будемо припускати, що тверде тіло в начальний момент перебуває у спокої.

Нерухому систему координат $Oxyz$ виберемо так, щоб центр важче-твердого тіла знаходився в плоскості $Oxyz$: $G(1,0,\eta)$. Припустимо ще, що удар виробляється в точку $P(a,0,c)$ тієї ж площини ударним імпульсом $\Phi(0,B,0)$.

Положення твердого тіла не змінюється за час удару, але при цьому тверде тіло придбає кутову швидкість $\boldsymbol{\omega}$ обертання навколо осі z, так що швидкості точок твердого тіла будуть визначатися із співвідношень

$$x_\nu'=-\omega y_\nu,\ y_\nu=\omega x_\nu,\ z_\nu=0.$$

Звільнивши тверде тіло від зв'язків у точках O та O' та замінивши їх дію реактивними ударами R_x, R_y, R_z можна при змінити теорему про рух центру мас. Теорема дає наступні рівняння для швидкостей точок тіла після удару:

$$\sum mx' = R_x + R'_x,\ \sum my' = B + R_y + R'_y,\ 0 = R_z + R'_z,$$

але

$$\sum mx' = -\omega \sum my = 0,\ \sum my' = \omega \sum mx = \omega M \xi,$$

тому рівняння удару можна буде переписати у вигляді

$$R_x + R'_x = 0,\ \omega M \xi = B + R_y + R'_y,\ 0 = R_z + R'_z,$$

Звернемося тепер до теореми про зміну моменту кількості руху під час удару. Спочатку визначимо момент кількості рухи після удару за допомогою співвідношення

$$\sum_\nu m_\nu \begin{Vmatrix} x_\nu & y_\nu & z_\nu \\ x'_\nu & y'_\nu & z'_\nu \end{Vmatrix} = \omega \sum_\nu m_\nu \begin{Vmatrix} x_\nu & y_\nu & z_\nu \\ -y_\nu & x_\nu & 0 \end{Vmatrix},$$

яке дає

$$K_x = -\omega \sum_\nu m_\nu x_\nu z_\nu,\ K_y = -\omega \sum_\nu m_\nu y_\nu z_\nu,\ K_z = \omega \sum_\nu m_\nu \left(x_\nu^2 + y_\nu^2 \right)$$

Таким чином,

$$. \; K_z = J_\nu \omega$$

Оскільки момент кількості руху системи до удару дорівнює нулю, будемо мати:

$$-\omega \sum_\nu mxz = -Bc - hR'_y \, , \; -\omega \sum_\nu myz = hR'_x \, , \; \omega J_z = aB \, .$$

Якщо удар прикладено в деякій точці P і не впливає на підшипники, то така точка P називається центром удару.

В цьому випадку

$$R_x = R'_x = R_y = R'_y = 0 \, ,$$

і рівняння набудуть вигляду

$$\omega M \xi = B \, , \; \omega \sum_\nu m_\nu x_\nu z_\nu = Bc \, , \; -\omega \sum_\nu m_\nu y_\nu z_\nu = 0 \, ,$$

$$J\omega = aB \, .$$

Звідси матимемо

$$a = \frac{J}{M\xi} \, ,$$

тобто центр удару знаходиться з того ж боку, що і центр тяжіння тіла, і відстань від осі обертання до центру удару залежить тільки від розподілу мас на твердому тілі. Крім того,

$$\omega \sum_\nu m_\nu x_\nu z_\nu = c\omega M \xi \, ,$$

або

$$c = \frac{\sum_{\nu} m_\nu x_\nu z_\nu}{M\xi}$$

Розглянемо властивості точки $S(0,0,c)$. Якщо перенести початок координат у точку S, то отримаємо

$$\sum_{\nu} m_\nu x_\nu (z_\nu - c) = \sum_{\nu} m_\nu x_\nu z_\nu = 0,$$

тобто вісь z для точки S є головною віссю інерції. Звідси відразу отримуємо спосіб побудови центру удару: На осі обертання твердого тіла необхідно знайти точку для якою ця вісь є головною віссю інерції, та відкласти (у бік центру мас) відстань a.

Розглянемо методи розв’язання задачі про балістичний маятником називають апарат, призначений для вимірювання швидкості снарядів. Він складається з труби, заповненою піском і підвішеною до горизонтальної осі. Снаряд, проникаючи в трубу, застряє у піску. Відбувається дуже швидко втрата швидкості снаряда - удар. Цей удар відбувається завдяки накладенню зв'язку.

Припустимо, що центр тяжкості маятника знаходиться у точці G на відстані l від осі обертання. Масу маятника позначим через M, а момент інерції його відносно осі обертання через $J = Mk^2$.

Для простоти вважатимемо, що снаряд застряє в циліндричній трубі на продовженні лінії OG на відстані a від осі обертання маятника. Після удару маятник набуває швидкості і починає робити коливання біля осі обертання. Застосовуючи теорему Карно для випадку накладання зв'язків будемо мати

$$T_1 - T_0 = \sum_{\nu=1}^{n} \frac{m_\nu w_\nu^2}{2},$$

де

$$T_0 = \frac{m w_0^2}{2},\ T_1 = \frac{1}{2}\left(Mk^2 + ma^2\right)\omega^2.$$

Тут m - маса снаряда; ω - швидкість обертання маятника після удару; v_0 - швидкість влучення снаряда. Звідси

$$\sum_{\nu} \frac{m_\nu w_\nu^2}{2} = \frac{m}{2}\left(v_0 - a\omega\right)^2 + \frac{Mk^2}{2}\omega^2. \qquad (1.1)$$

Підставляючи ці значення (1.1), отримаємо

$$m v_0^2 - \left(Mk^2 + ma^2\right)\omega^2 = m\left(v_0 - a\omega\right)^2 + Mk^2\omega^2,$$

звідки маємо

$$v_0 = \frac{ma^2 + Mk^2}{ma}\omega\ . \qquad (1.2)$$

Надана сама собі після удару, система буде здійснювати рух відповідно до інтегралу живих сил

$$T = U + h$$

де

$$U = (Ml + ma) g \cos \vartheta,$$

так що

$$\frac{Mk^2 + ma^2}{2} \omega^2 = (Ml + ma) g \cos \vartheta + h.$$

Постійну h слід визначати за умови, що швидкість кінці руху перетворюється на нуль, тобто

$$h = -(Ml + ma) g \cos \vartheta$$

Тоді, оскільки на початку руху $h = 0$, маємо

$$\frac{Mk^2 + ma^2}{2} \omega^2 = (Ml + ma) g - (Ml + ma) g \cos \vartheta$$

або

$$(Mk^2 + ma^2) \omega^2 = 4(Ml + ma) g \sin^2 \frac{\vartheta}{2},$$

звідки

$$\omega = 2 \sqrt{\frac{(Ml + ma) g}{Mk^2 + ma^2}} \sin \frac{\vartheta}{2}.$$

Підставляючи це значення в (1.2), отримаємо

$$v_0 = 2 \frac{\sqrt{(Ml + ma)(Mk^2 + ma^2) g}}{ma} \sin \frac{\vartheta}{2}.$$

Знаючи кут відхилення маятника ϑ, можна знайти швидкість снаряда при попаданні його в маятник.

Розглянемо голономну механічну систему, стан руху якої визначається узагальненими координатами $q_1, q_2, ..., q_k$. Рух такої системи задовольняє рівнянням Лагранжа другого роду

$$\frac{d}{dt}\frac{\partial T}{\partial q_i'} - \frac{\partial T}{\partial q_i} = Q_i \ \left(i = 1, 2, ..., k\right).$$

Припустимо, що механічна система отримує удар за дуже короткий час τ, причому зв'язки, накладені на систему до удару, залишаються після удару. Проінтегруємо рівняння руху за час удару:

$$\int_{t_0}^{t_0+\tau} d\left(\frac{\partial T}{\partial q_i'}\right) - \int_{t_0}^{t_0+\tau}\left(\frac{\partial T}{\partial q_i}\right)dt = \int_{t_0}^{t_0+\tau} Q_i dt \qquad (1.3)$$

Зважаючи на те, що $\partial T/\partial q_i$ — кінцева величина, матимемо

$$\lim_{\tau\to 0}\int_{t_0}^{t_0+\tau}\left(\frac{\partial T}{\partial q_i}\right)dt = 0.$$

Тоді в межі рівняння (1.3) набудуть вигляду

$$\Delta\left(\frac{\partial T}{\partial q_i'}\right) = \lim_{\tau\to 0}\int_{t_0}^{t_0+\tau} Q_i dt.$$

Отримані рівняння є основними рівняннями теорії удару у вигляді Лагранжа.

Розглянемо рух твердого тіла з однією нерухомою точкою. Вибираючи за осі рухомий системи координат головні осі інерції тіла, запишемо динамічні рівняння Ейлера:

$$A\frac{dp}{dt}+\left(C-B\right)qr=L,\ B\frac{dq}{dt}+\left(A-C\right)pr=M,\ C\frac{dr}{dt}+\left(B-A\right)qr=N.$$

Нехай у момент t_0 до твердого тіла додається зовнішній удар. Тоді будемо мати

$$\lim_{\tau\to 0}\int_{t_0}^{t_0+\tau}\left(A-C\right)prdt=0,\ \lim_{\tau\to 0}\int_{t_0}^{t_0+\tau}\left(B-A\right)qpdt=0,\ \lim_{\tau\to 0}\int_{t_0}^{t_0+\tau}\left(C-B\right)qrdt=0.$$

З рівнянь Ейлера отримаємо

$$A\Delta p=\lim_{\tau\to 0}\int_{t_0}^{t_0+\tau}Ldt,\ B\Delta q=\lim_{\tau\to 0}Mdt,\ C\Delta r=\lim_{\tau\to 0}\int_{t_0}^{t_0+\tau}Ndt.$$

2 ДОСЛІДЖЕННЯ НЕПРУЖНОГО ЗІТХНЕННЯ КУЛІ З ПРИЗМОЮ

2.1 Математична модель

Момент імпульсних сил відносно осі Oy_1 рівний нулю, тому момент кількості руху до удару і після зберігається, тобто:

$$\frac{2}{3}amv = J_{y_1}\omega \tag{2.1}$$

Обчислимо момент інерції системи відносно осі O_1y_1, складаючи його із суми моментів інерції кулі і призми відносно цієї осі. Момент інерції кулі масою m рівний:

$$J_{y_1} = mh^2 = m\left(a^2 + \frac{9}{4}a^2\right) = \frac{13}{4}ma^2 \ . \tag{2.2}$$

Для обчислення моменту інерції призми J_{2y_1}, попередньо обчислимо центральний момент інерції відносно осі O_y , паралельної осі O_1y_1:

$$J_y = \int_{(M)}(x^2 + z^2)dm \ , \tag{2.3}$$

де $dm = \tau dx * dy * dz$.

Так як

$$J = \tau(\int_{-\frac{a}{2}}^{\frac{a}{2}}\int_{\frac{a}{2}}^{\frac{a}{2}}\int_{-\frac{3a}{2}}^{\frac{3a}{2}}x^2dxdydz + \int_{-\frac{a}{2}}^{\frac{a}{2}}\int_{-\frac{a}{2}}^{\frac{a}{2}}\int_{-\frac{3a}{2}}^{\frac{3a}{2}}z^2dxdydz) = \frac{5ma^2}{2} \tag{2.4}$$

Тоді на основі теореми про момент інерції відносно осі, паралельної центральній осі, знаходимо:

$$J_{y_1} = \frac{13}{4}ma^2 + 10ma^2 = \frac{53}{4}ma^2. \quad (2.5)$$

З рівнянь (2.2) і (2.5) випливає, що:

$$J_{2y_1} = J_y + 3mh^2 = \frac{5ma^2}{2} + 3m\left(\frac{a^2}{4} + \frac{9a^2}{4}\right) = 10ma^2. \quad (2.6)$$

Формули (2.1) і (2.6) дозволяють визначити кутову швидкість обертання системи тіл навколо осі O_1y_1 після удару:

$$\omega = \frac{6}{53} * \frac{v}{a}. \quad (2.7)$$

Для визначення найменшої швидкості v, при якій призма обертається, застосовуємо теорему про зміну кінетичної енергії:

$$-J_{y_1}\frac{\omega^2}{2} = A, \quad (2.8)$$

де А – робота діючих сил тяжкості.

В початковий момент (в момент удару) центр інерції має координати:

$$\begin{cases} x_{1c} = \frac{ma + \frac{3}{2}ma}{4m} \\ z_{1c} = \frac{3}{2}a \end{cases} \quad (2.9)$$

Спрощуємо вираз:

$$\begin{cases} x_{1c} = \frac{5}{8}a \\ z_{1c} = \frac{3}{2}a \end{cases} \quad (2.10)$$

Та знаходиться на відстані:

$$r_{1c} = a\sqrt{\frac{9}{4} + \frac{25}{64}} = \frac{13}{8}a.$$

Для того щоб призма могла обертатися навколо ребра AB, центр інерції системи повинен піднятися на висоту:

$$h = (1.625 - 1.5)a = 0.125a. \quad (2.11)$$

Таким чином, робота діючих сил тяжкості, виконана при обертанні призми відносно ребра AB до положення, в якому центр інерції системи займає найвище положення, рівна:

$$A = -4mgh = -0.5mga. \quad (2.12)$$

На основі відношення (2.8) і (2.12), знаходимо шукану швидкість:

$$v = \frac{1}{3}\sqrt{53ag}\,. \quad (2.13)$$

Таким чином, ми отримали формулу для визначення мінімальної швидкості кулі, при якій призма зможе перевернутися. Це дозволяє більш точно моделювати фізичну поведінку об'єктів при їх взаємодії, що є важливим аспектом для успішної реалізації проекту.

Переходячи до практичної частини, ми зосередимося на реалізації даних теоретичних положень у середовищі програмування. Наступні кроки включають побудову моделей об'єктів, налаштування фізичних параметрів і симуляцію їх взаємодії за допомогою відповідних алгоритмів. Ми розпочнемо з опису створення об'єктів за допомогою технології Direct3D,

включаючи налаштування матеріалів, освітлення та інших параметрів, необхідних для реалістичної візуалізації та точної фізичної симуляції.

Ми будемо використовувати отримані математичні моделі для створення програмного забезпечення, яке дозволить проводити комплексні симуляції взаємодії кулі та призми. Це дозволить не лише перевірити правильність теоретичних розрахунків, але й забезпечити наочну демонстрацію фізичних явищ, що відбуваються при таких взаємодіях.

2.2 Платформа для розробки системи дослідження

Для розробки дипломного проекту використовувалася Visual Studio з мовою програмування C# через декілька причин:

- C# має чистий і зрозумілий синтаксис, що робить код легко читаємим і підтримується. Це дозволяє розробникам швидко орієнтуватися в коді та проводити його аналіз
- C# повністю підтримує концепції ООП, такі як інкапсуляція, спадкування та поліморфізм. Це дозволяє організовувати код у логічні сутності та підвищує його структурованість та гнучкість
- .NET Framework включає великий набір класів і бібліотек для вирішення різних завдань, включаючи роботу з файлами, мережею, базами даних, графікою і багатьом іншим. Це спрощує розробку додатків, оскільки багато стандартних завдань вже реалізовано
- Завдяки тому, що .NET Framework та .NET Core підтримуються на різних платформах, програми, написані на C#, можуть бути легко портовані на інші операційні системи, такі як Linux і macOS, що розширює їхню аудиторію та збільшує їх доступність
- Visual Studio надає повноцінне інтегроване середовище розробки (IDE) з багатьма інструментами для створення, налагодження та тестування програмного забезпечення. Завдяки цьому розробники можуть ефективно працювати над проектами та підвищувати їх якість

- Visual Studio надає візуальні редактори для створення графічних інтерфейсів. Наприклад, Windows Forms Designer дозволяє легко створювати та налаштовувати форми додатків, додаючи на них елементи керування та налаштовуючи їх властивості без необхідності писати код вручну.

Для моделювання використовувався DirectX, який є комплексним набором API, розробленим Microsoft, який надає потужні інструменти для створення та керування графікою, звуком та іншими мультимедійними елементами на платформах Windows. Розглянемо детальніше цю технологію та її застосування.

DirectX складається з кількох ключових компонентів, серед яких Direct3D, DirectDraw, DirectSound, DirectInput та інші. Основний компонент для роботи з 3D-графікою — це Direct3D, який надає високопродуктивні можливості для відтворення тривимірних сцен. Основні можливості та компоненти DirectX включають:

- Direct3D: Головний інструмент для роботи з тривимірною графікою, що забезпечує детальний контроль над графічним апаратним забезпеченням, дозволяючи створювати складні сцени з реалістичним освітленням та деталізацією.
- Direct2D: Використовується для високопродуктивного відтворення двовимірної графіки, що є корисним для інтерфейсів користувача та візуалізації графічних елементів.
- DirectSound: Забезпечує можливості відтворення звукових ефектів і музики, що є важливим для створення комплексних мультимедійних додатків.
- DirectInput: Надає можливість отримання даних від різних пристроїв введення, таких як клавіатури, миші, джойстики та інші контролери.

Переваги Використання DirectX:

- Реалістичність візуалізації: Однією з основних переваг DirectX є можливість створення високоякісної графіки. Це дозволяє точно

моделювати фізичні процеси, такі як непружне зіткнення кулі з призмою, з урахуванням всіх необхідних деталей. Використовуючи шейдери, можна реалізувати складні ефекти освітлення, тіней та текстурування, що додає реалістичності сценам.

- Висока продуктивність: DirectX забезпечує прямий доступ до апаратного забезпечення, що дозволяє максимально ефективно використовувати можливості сучасних графічних процесорів (GPU). Це особливо важливо для обробки фізичних взаємодій у реальному часі, де кожен кадр має бути оброблений з високою швидкістю та точністю.
- Гнучкість та масштабованість: DirectX підтримує широкий спектр графічних карт та пристроїв, що дозволяє створювати масштабовані рішення, які працюють на різних апаратних конфігураціях. Це забезпечує сумісність вашого додатка з великою кількістю систем, від офісних комп'ютерів до ігрових станцій.
- Візуалізація Фізичних Процесів: для вирішення задач теорії удару необхідно створити точну та наочну візуалізацію процесу зіткнення. DirectX дозволяє реалізувати складні фізичні моделі та відобразити їх у тривимірному просторі з високою точністю. Використовуючи Direct3D, можна відтворювати траєкторії руху, деформації при зіткненні та інші важливі аспекти.
- Анімація та симуляція: DirectX підтримує анімацію об'єктів та симуляцію фізичних процесів, що є ключовими елементами для створення віртуальних лабораторій. Ви можете використовувати ці можливості для детального моделювання динаміки зіткнення, включаючи розрахунок імпульсів, сил та енергій, що беруть участь у процесі.

2.3 Декомпозиція системи

Інформаційна система дипломної роботи розробляється з метою максимально спростити процес дослідження руху системи тіл, що стикаються. За допомогою побудованої системи вирішуються дві основні задачі: візуалізація руху системи та аналіз параметрів моделей системи.

Для реалізації розв'язку першої задачі повинні бути виділені наступні складові: інформаційна технологія розрахунку рівнянь руху системи тіл, та інформаційна технологія моделювання руху системи тіл у тривимірному просторі.

Для реалізації розв'язку другої задачі повинні бути виділена наступна складова: інструментарій для аналізу отриманих результатів.

На рис. 2.1 представлено схему декомпозиції інформаційної системи кваліфікаційної роботи.

У якості вхідних даних задаються:

- Геометричний розмір призми.
- Вага кулі.
- Швидкість кулі.

Інформаційна технологія 3-D візуалізації руху системи тіл включає розробку тривимірних моделей тіл. У якості вхідних даних є значення координати кулі та кут відхилення призми від вертикалі у поточний момент часу. У якості вихідних даних є змодельована сцена тривимірної візуалізації. Інформаційна технологія аналізу даних означає, що для отриманої моделі системи тіл визначається швидкість кулі необхідна для перекидання призми.

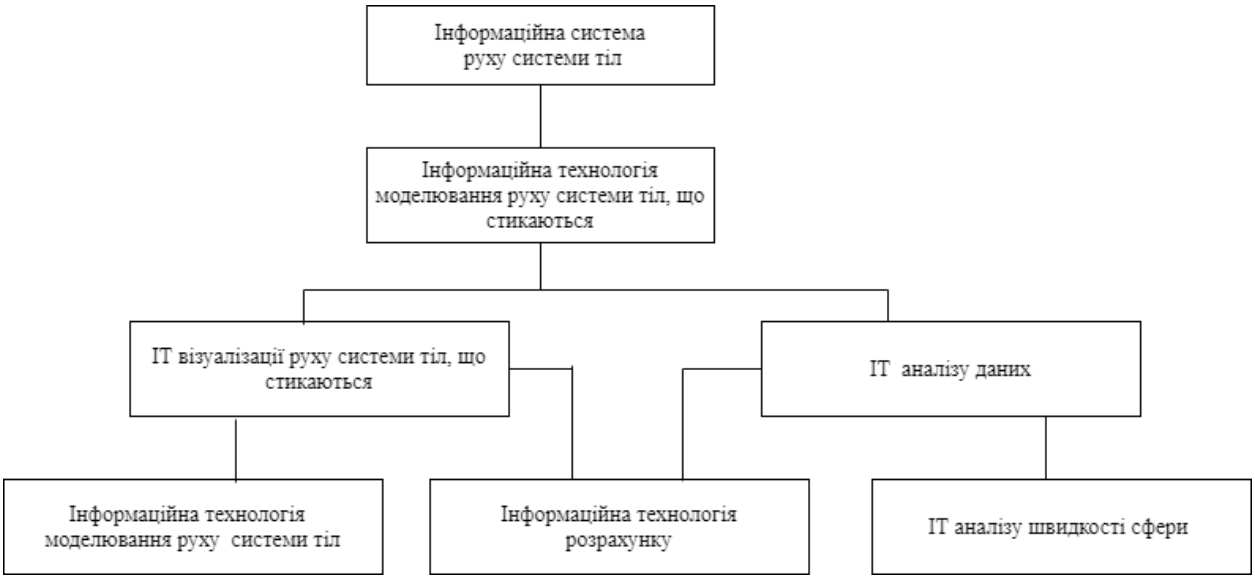

Рисунок 2.1- Декомпозиція ІС

Для програмної реалізації системи розроблено два модулі проекту. На рис. 2.2 представлено схему структури системи. На рис. 2.3 представлено схему застосунку інформаційної системи.

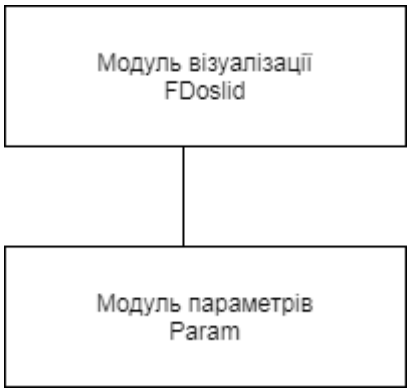

Рисунок 2.2 – Структура системи

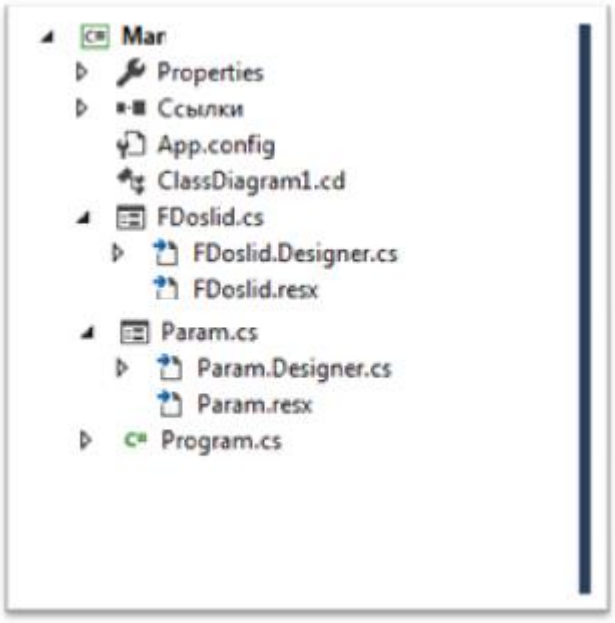

Рисунок 2.3 – Структура застосунку

2.4 Модуль візуалізації руху системи тіл, що стикаються

Модуль візуалізації - клас FDoslid (див. Додаток А), призначений для тривимірного моделювання руху системи.

Ми визначаємо змінні для пристрою Direct3D (Device), трьох об'єктів типу Mesh (sphera, prism і base_box), матеріалів для цих об'єктів, координат і параметрів об'єктів.

Конструктор FDoslid() виконує початкову ініціалізацію форми і встановлює її властивості. Він викликає метод InitializeComponent(), який створює і ініціалізує всі компоненти форми, такі як кнопки, меню та інші елементи керування, визначені у візуальному дизайнері форми. Це забезпечує належне відображення інтерфейсу користувача.

Після ініціалізації компонентів встановлюються значення null для змінних d3d, sphera, і prism. Це необхідно для того, щоб пізніше можна було перевірити, чи були ці об'єкти успішно створені і чи не виникли помилки під час їх ініціалізації. Ініціалізація значенням null також допомагає уникнути

випадкових звернень до не ініціалізованих об'єктів, що може призвести до виключень і збоїв у роботі програми.

Device d3d	Посилання на пристрій Direct3D
Mesh sphera	Меш об’єкт сфери
Mesh prism	Меш об’єкт призми
Mesh base_box	Меш об’єкт онови
Material sfera_material	Матеріал сфери
Material prism_material	Матеріал призми
Material base_box_material	Матеріал основи
Vector3 a	Вектор координат сфери
Vector3 b	Вектор координат призми
float speed_sphera	Горизонтальна швидкість сфери
double rebr	Ребро основи призми
double height_prism	Висота призми
double mas_prism	Маса призми
double mas_sphera	Маса сфери
double min_speed_sphera	Мінімальна швидкість сфери для перевертання призми

Таблиця 1- Список глобальних змінних проекту

Також у конструкторі використовується метод SetStyle(ControlStyles.ResizeRedraw, true), який встановлює стиль керування, що дозволяє перерисовувати форму при зміні її розміру. Це гарантує, що всі графічні елементи на формі будуть правильно перерисовуватись і зберігати свою коректну позицію і розміри, навіть якщо користувач змінює розмір вікна програми.

Файл FDoslid.Designer:

Також нам потрібно додати код, який буде звільняти ресурси Direct3D, що були зайнятті моделлю:

```
protected override void Dispose(bool disposing)
{
    if (disposing && (components != null))
    {
        components.Dispose();

        if(sphera != null)
            sphera.Dispose();

        if (base_box != null)
            base_box.Dispose();

        if (prism != null)
            prism.Dispose();

        if (d3d != null)
            d3d.Dispose();
    }
    base.Dispose(disposing);
}
```

Рисунок 2.4 – Звільняємо ресурси Direct3D

Метод FDoslid _Load виконується, коли форма завантажується і стає видимою для користувача. Він починає з налаштування параметрів пристрою Direct3D у блоці try/catch через об'єкт PresentParameters, який визначає різні параметри відображення, такі як кількість буферів, режим відображення, глибина буфера, та інші.

Наприклад:

d3dpp.Windowed = true;

встановлює віконний режим відображення, що дозволяє програмі працювати у вікні, а не в повноекранному режимі.

Після налаштування параметрів створюється пристрій Direct3D за допомогою конструктора Device, який приймає ці параметри і намагається створити пристрій. Якщо створення пристрою не вдається, програма «ловить» виключення і показує повідомлення з помилкою, а потім закриває

форму. Це гарантує, що користувач отримає зворотний зв'язок у разі проблем з ініціалізацією графіки.

Наступним кроком є створення об'єктів sphera, prism і base_box, а також встановлення їхніх матеріалів. Ці об'єкти будуть використовуватися для рендерингу тривимірних форм на екрані. Властивості матеріалів, такі як дифузний і дзеркальний колір, визначають, як ці об'єкти будуть виглядати при відображенні і взаємодії зі світлом.

Метод SetupProekcii відповідає за налаштування проекції та освітлення сцени в Direct3D. Перший крок в методі - це включення першого джерела світла:

```
d3d.Lights[0].Enabled = true;
```

Це джерело світла буде використовуватися для освітлення об'єктів на сцені, що дозволить їм виглядати більш реалістичною завдяки відблискам і тіням.

Колір джерела світла задається як білий, що забезпечує нейтральне освітлення сцени:

```
d3d.Lights[0].Diffuse = Color.White;
```

Позиція джерела світла встановлюється у точку (0, 0, 0):

```
(d3d.Lights[0].Position = new Vector3(0, 0, 0)) ;
```

Основна частина методу налаштовує матрицю проекції (d3d.Transform.Projection). Використовуючи метод Matrix.PerspectiveFovLH, створюється матриця перспективної проекції з кутом поля зору в 60 градусів ($\pi/3$ радіан), співвідношенням сторін, яке дорівнює відношенню ширини

форми до її висоти, та відстанями до ближньої і дальньої площини відсікання (1.0f та 50.0f відповідно):

```
Matrix.PerspectiveFovLH((float)Math.PI / 3, this.Width / this.Height, 1.0f, 50.0f);
```

Це налаштування визначає, як тривимірні об'єкти будуть проектуватися на двовимірну площину екрану.

Метод Mathdd призначений для обчислення мінімальної швидкості сфери, при якій призма перевернеться.

Формула для обчислення vmin базується на фізичних властивостях об'єктів і законах динаміки. Вона враховує гравітаційне прискорення та розміри призми, щоб визначити швидкість, при якій момент сили, створений рухомою сферою, буде достатнім для перевертання призми. Це може бути використано для моделювання фізичної взаємодії між об'єктами в тривимірному просторі.

Повернення значення vmin з методу дозволяє використовувати його в інших частинах програми для перевірки умови перевертання призми. Цей метод є важливим для реалізації логіки моделювання і перевірки фізичних законів в програмі.

Метод DrawPrism відповідає за відображення об'єктів сцени на екрані. Спочатку очищуються буфер екрану та буфер глибини, що гарантує, що попередні малюнки не залишаться на екрані і нові зображення будуть відображені на чистому тлі:

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

Цей код складається з:

1. ClearFlags.Target – відповідає за очищення екранного буферу
2. ClearFlags.ZBuffer – відповідає за очищення глибинного буферу

3. Color.Azure– встановлює колір екрану на світло-голубий після очищення екранного буферу
4. 1.0f– значення яке ми передаємо в буфер глибини після його очистки для коректного відображення об’єктів на екрані

Потім починається сцена (d3d.BeginScene()), і налаштовуються параметри проекції та освітлення за допомогою методу SetupProekcii.

Після налаштування сцени встановлюється режим відображення, який визначає, як будуть заповнюватися полігони об'єктів (суцільні або каркасні).

```
d3d.RenderState.FillMode = FillMode.Solid;
```

Далі по черзі задаються матеріали для кожного з об'єктів (бази, призми та сфери) і встановлюються їхні матриці трансформації. Ці матриці визначають позицію, обертання і масштабування об'єктів у тривимірному просторі.

Після налаштування матеріалів і трансформацій, об'єкти малюються за допомогою методу DrawSubset, який виконує рендеринг конкретного підмножини об'єкта (у даному випадку, підмножини 0).

Після завершення рендерингу всіх об'єктів сцена завершується (d3d.EndScene()) і результат відображається на екрані за допомогою d3d.Present(). Таким чином, метод DrawPrism забезпечує повний цикл рендерингу об'єктів на сцені.

Метод timer1_Tick:

Метод timer1_Tick виконується при кожному «тіку» таймера і оновлює стан об'єктів на екрані. Викликається метод DrawPrism, який відображає об'єкти з використанням оновлених значень обертання.

Після відображення призми оновлюються координати сфери. Якщо сфера ще не досягла центру призми, її координати a.X і a.Z збільшуються на значення speed_sphera. Це забезпечує плавний рух сфери у напрямку призми.

Коли сфера досягає центру призми:

a.X >= prism_centX && a.Z >= prism_centZ

Тоді обчислюється мінімальна швидкість, необхідна для перевертання призми (v = Mathdd()). Якщо поточна швидкість сфери перевищує це значення, призма перевертається і метод DrawPrism знову викликається для відображення оновленої сцени.

Якщо поточна швидкість сфери не досягає необхідного значення, призма починає хитатися і метод DrawPrism знову викликається для відображення оновленої сцени.

В кінці таймер вимикається, щоб зупинити подальше оновлення, оскільки завдання завершено:

timer1.Enabled = false;

Метод запускToolStripMenuItem_Click:

Виконується при натисканні на пункт меню "Запуск". Першим кроком є відновлення початкових координат сфери:

a = new Vector3(-4.5f, 0, 4.5f);

Це потрібно для того, щоб сфера завжди починала свій рух з визначеної початкової позиції, незалежно від попередніх змін її координат.

Далі перезапускається таймер. Спочатку він вимикається щоб зупинити будь-які поточні «тіки», а потім знову включається:

timer1.Enabled = false;
timer1.Enabled = true;

Це гарантує, що таймер починає свій відлік з нуля і всі події оновлення будуть виконуватися з самого початку. Таймер використовується для періодичного виклику методу timer1_Tick, який оновлює позиції об'єктів на екрані.

Таким чином, метод запускToolStripMenuItem_Click забезпечує правильний запуск і перезапуск симуляції руху сфери і взаємодії між об'єктами, гарантуючи, що всі початкові умови встановлені правильно перед початком нового циклу симуляції.

Метод настройкиПараметровОбъектовToolStripMenuItem_Click:

Метод настройкиПараметровОбъектовToolStripMenuItem_Click виконується при натисканні на пункт меню "Настройки параметрів об'єктів". Він створює новий екземпляр Windows Forms SetupF, який, є діалоговим вікном для налаштування параметрів об'єктів сцени.

Метод setup.ShowDialog() викликає це діалогове вікно у модальному режимі, що означає, що користувач не зможе взаємодіяти з головним вікном програми, поки не закрив діалогове вікно. Це зручно для забезпечення того, що користувач завершить налаштування параметрів перед продовженням роботи з основною програмою.

Таким чином, цей метод забезпечує механізм для налаштування параметрів об'єктів, що дозволяє користувачеві змінювати параметри безпосередньо під час виконання програми. Це може включати зміну розмірів, кольорів, швидкості та інших властивостей об'єктів, що робить програму більш гнучкою та інтерактивною.

2.5 Модуль параметрів системи тіл, що стикаються

Модуль параметрів - клас FParm, що має вигляд, представлений на рис. 2.5. Всі компоненти NumericUpDown мають значення властивості Value, що відповідає значенню параметра моделі за замовченням.

Користувач має функціональну можливість змінювати параметри моделі для проведення нового. Зміна значень параметрів моделі не впливає на прорисовку системи тіл з точку зору геометрії системи. Для зв'язку даних між модуля використовуються відкриті поля класу FParm (див. Додаток Б).

Для зручності користувача на формі модуля розташовано дві кнопки Ок та Відміна, яка мають відповідні значення властивості DialogResult.

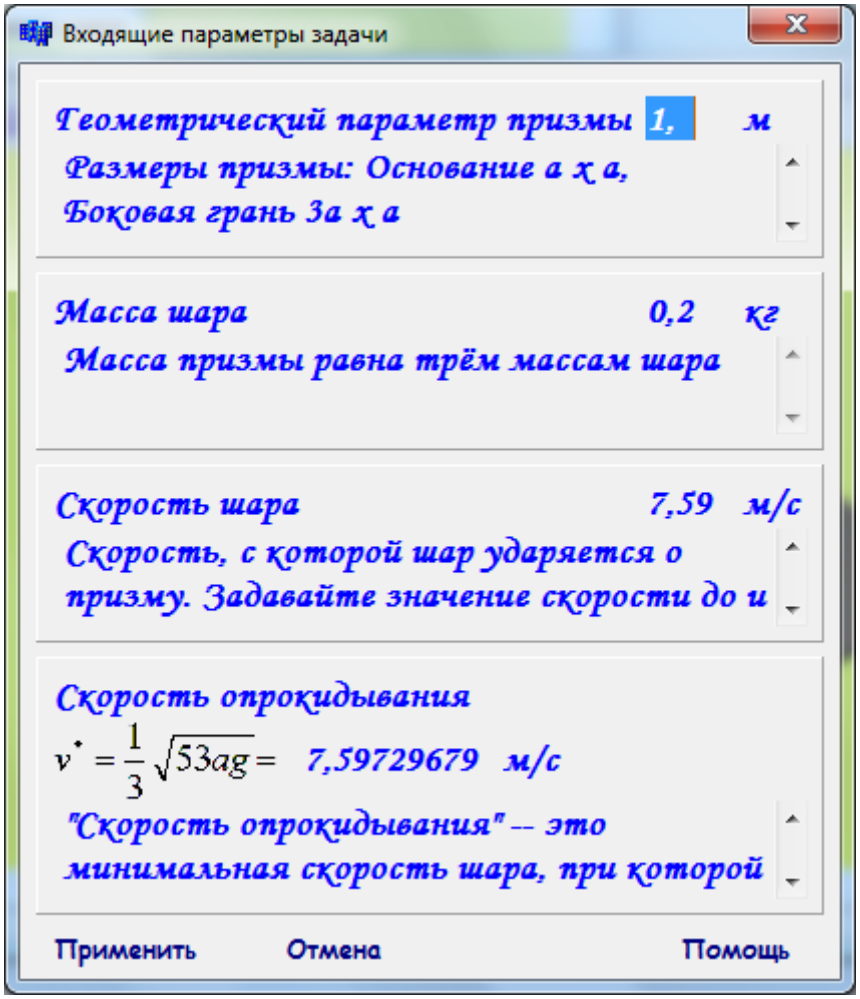

Рисунок 2.5 – Вікно модуля параметрів

Конструктор FParm викликається при створенні нової форми налаштувань. Він приймає параметри типу float, які представляють змінні величини. Це дозволяє передавати початкові значення для цих параметрів із головної форми до форми налаштувань.

У тілі конструктора викликається метод InitializeComponent(), який автоматично генерується дизайнером форм Visual Studio. Цей метод

ініціалізує всі компоненти користувацького інтерфейсу форми, такі як текстові поля, кнопки тощо. Він обов'язково має бути викликаний першим у конструкторі, щоб гарантувати коректну ініціалізацію форми.

Після ініціалізації компонентів конструктор присвоює значення параметрів відповідним властивостям. Це дозволяє формі налаштувань відображати передані значення у відповідних текстових полях, коли форма завантажується.

Метод SetupF_Load обробляє подію завантаження форми. Він автоматично викликається, коли форма SetupF відображається на екрані. Основною метою цього методу є ініціалізація текстових полів значеннями властивостей, які ми передаємо.

На початку методу значення властивостей конвертується в рядок за допомогою методу ToString() і присвоюється відповідним текстовим полям. Це забезпечує відображення поточних значень у відповідному текстовому полі форми, що дозволяє користувачеві побачити і, за необхідності, змінити це значення. Таким чином метод забезпечує початкову синхронізацію між внутрішніми властивостями форми та інтерфейсом користувача.

Метод buttonSave_Click обробляє подію натискання на кнопку Save. Його основна функція полягає у збереженні введених користувачем значень і закритті форми.

Спочатку метод намагається конвертувати текст із текстових полей у число типу float за допомогою методу float.TryParse. Якщо конвертація успішна, значення присвоюється відповідній властивості. У випадку, якщо конвертація не вдається, відображається повідомлення про помилку, і виконання методу припиняється за допомогою оператора return.

Аналогічно обробляється значення з текстового поля textBoxPrismRebrLenght. Після успішної конвертації значення присвоюється властивості PrismRebrLenght. Якщо обидві конвертації успішні, метод встановлює результат діалогу форми на DialogResult.OK, що сигналізує про успішне завершення операції. Потім форма закривається.

3 ФУНКЦІОНАЛ ВІРТУАЛЬНОЇ ЛАБОРАТОРІЇ

При відкриті запуску застосування за рахунок модуля візуалізації відкривається вікно як показано на рис. 3.1.

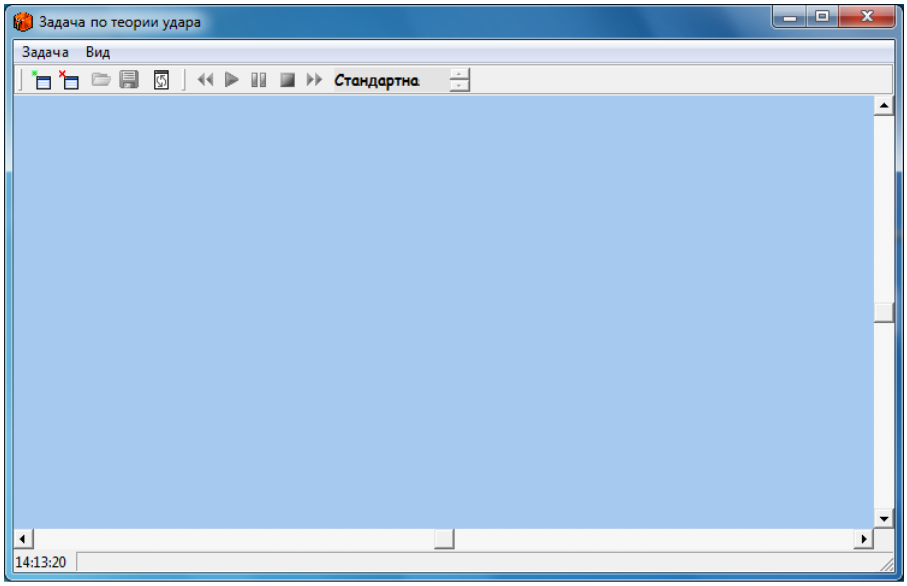

Рисунок 3.1 - Головне вікно лабораторії

Користувач має функціонал для зміни кольору вікна лабораторії – рис. 3.2.

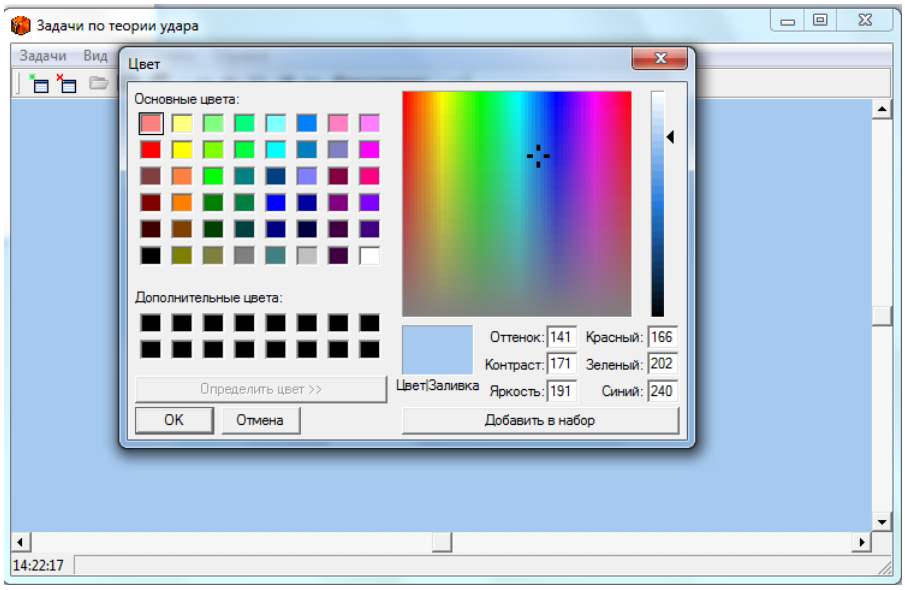

Рисунок 3.2 – Функціонал вибору кольору вікна лабораторії

Для зображення 3-D моделі призми користувач обирає відповідну задачу в головному меню.

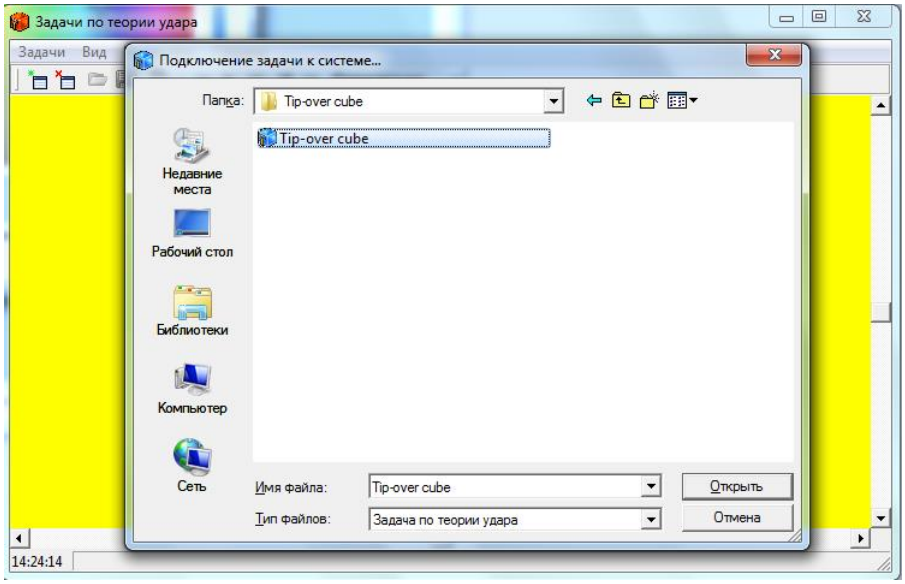

Рисунок 3.3 – Підключення задачі до віртуальної лабораторії

Модель призми і платформи завантажуються на сцену, як показано на рис. 3.4.

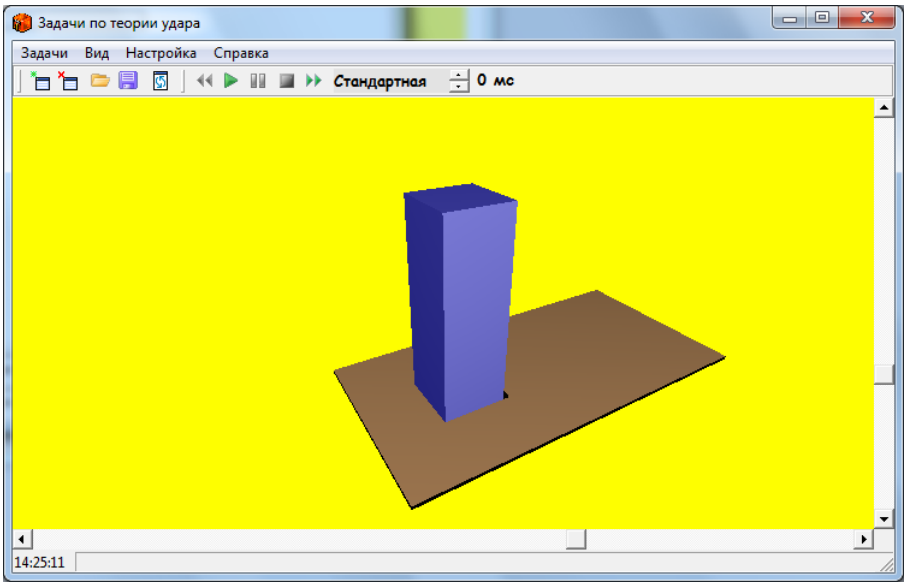

Рисунок 3.4 – Сцена лабораторії з нерухомою призмою

У користувача є можливість змінити масштаб зображення за допомогою функціонала меню (рис. 3.5).

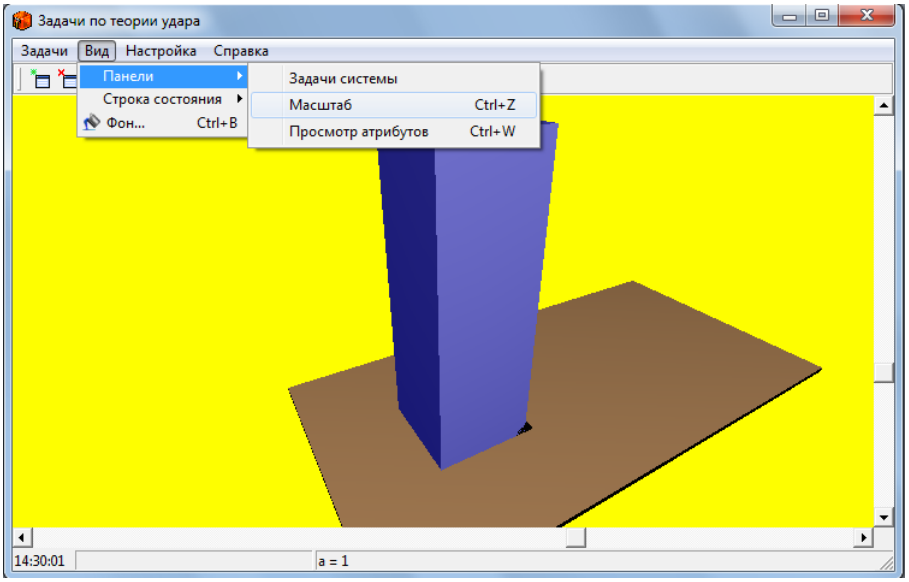

Рисунок 3.5 – Функціонал зміни масштабу сцени

Крім того за бажанням користувача за допомогою головного меню форми можна додати вивід необхідної інформації на інформативну панель застосування, як показано на рисунку 3.6.

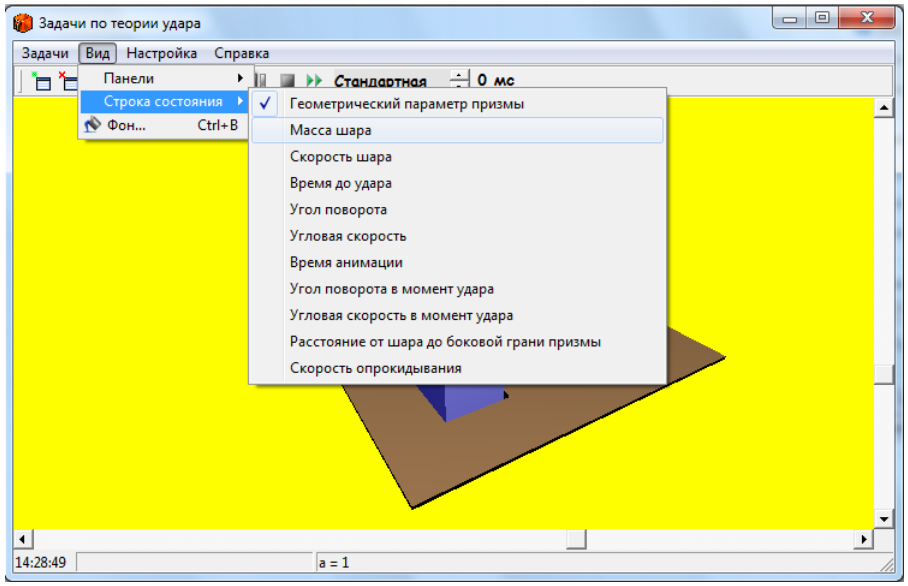

Рисунок 3.6 – Функціонал зміни строки стану лабораторії

Згідно рис. 2.5 для того, щоб сфера перекинула призму необхідно мати швидкість більше, ніж 7.59729679 м/с. За замовченням параметри системи тіл забезпечують швидкість кульки в момент стикання кулька має швидкість 7.59 м/с, що є недостатньою величиною для перекинення призми. Результат такого розрахунку представлено на рис. 3.7.

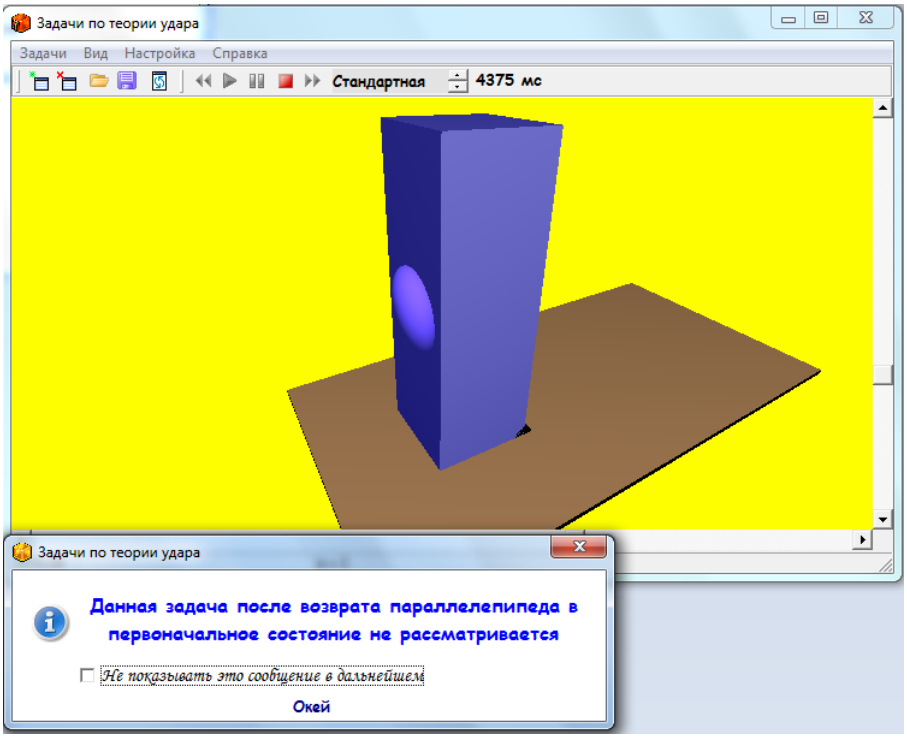

Рисунок 3.7 – Візуалізація кінцевого положення системи після зіткнення тіл при недостатній швидкості кулі

Якщо користувач змінює параметри моделей системи, щоб забезпечити швидкість кулі на момент зіткнення більше 7.59729679 м/с, наприклад 7.6 м/с, то результат такого розрахунку представлено на рис. 3.8.

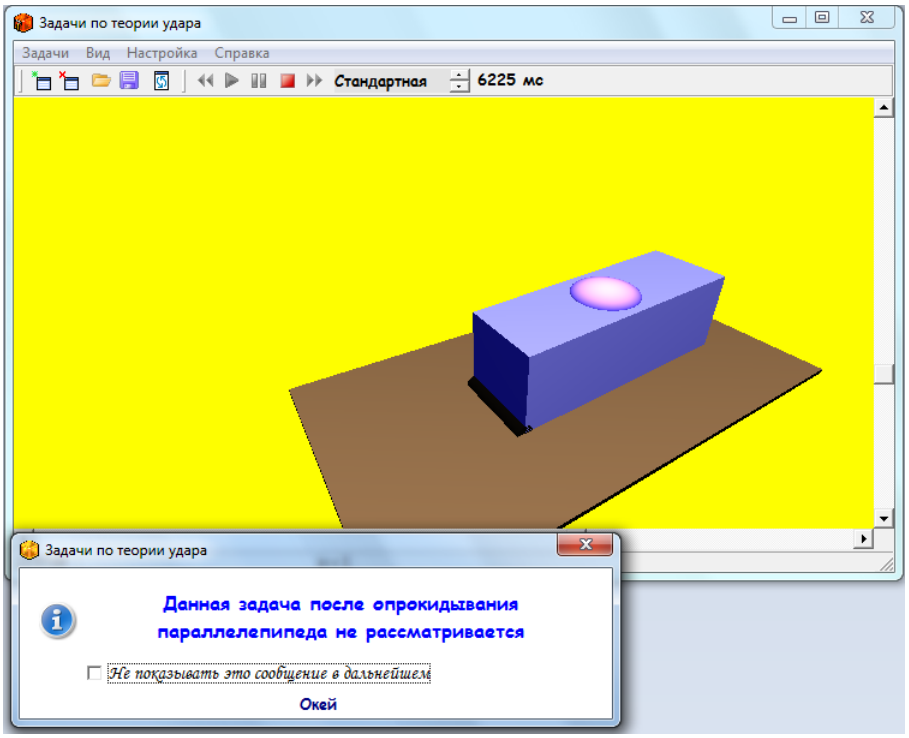

Рисунок 3.8 – Візуалізація кінцевого положення системи після зіткнення тіл при достатній швидкості кулі

ВИСНОВОК

В результаті виконання кваліфікаційної роботи з дослідження використання віртуальної лабораторії для розв’язання задач з теорії удару були розглянуті теоретичні основи механіки та динаміки твердих тіл. Було розроблено програмне забезпечення для моделювання та візуалізації процесу непружного удару кулі з призмою.

Програмою було продемонстровано співпадіння програмної реалізації з теоретичними розрахунками та реальними даними. Поведінка тіл при зіткненні та траєкторії руху після зіткнення співпадають з очікуваними значеннями. Програма також має змогу враховувати можливості зміни параметрів кулі та призми.

Розроблене програмне забезпечення з подальшою модифікацією може бути використано для рішення та моделювання інших задач теорії удару, які є важливими в сучасних реаліях.

Результатом виконання роботи є можливість проведення анімації руху системи тіл, що стикаються.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Павловський М.А., Теоретична Механіка // Київ: «ТЕХНІКА», 2002 – 512 с.

2. Апостолюк О. С., Теоретична механіка: Збірник задач / О. С. Апостолюк, В. М. Воробйов, Д. І. Ільчишина та ін.// Київ: «Техніка», 2007 – 400 с.

3. Березін Л.М., Теоретична механіка / Березін Л.М., Кошель С.О.// Київ: «Центр учбової літератури», 2020 – 218 с.

4. Кузьо І.В., Теоретична механіка // Київ: «Фоліо», 2017 – 780 с.

5. Булгаков В.М., Теоретична механіка / В.М. Булгаков, В.В. Яременко, О.М. Черниш, М.Г.Березовий// Київ: «Центр учбової літератури», 2021 – 640 с.

6. Яскілка М.Б., Збірник завдань для розрахунково-графічних робіт з теоретичної механіки/ Київ: Вища школа: «Веселка», 1999. – 351 с.

7. Бублик В.В. Об'єктно-орієнтоване програмування// Київ: Підручник: «ІТ-книга», 2015 – 624 с.

8. Кравець П.О., Об'єктно-орієнтоване програмування// Львів: Видавництво Львівської політехніки, 2012 – 624 с.

9. Дібрівний О.А., Вступ до об’єктно-орієнтованого програмування C#/ Дібрівний О.А., Гребенюк В.В.// Київ: Державний університет телекомунікацій, 2018. – 190 с.

10. Коноваленко І.В., Програмування мовою C# 6.0// Тернопіль: ТНТУ, 2016. – 227 с.

ДОДАТОК А

Код модуля візуалізації

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace Mar
{
   public partial class FDoslid : Form
   {

GraphPane myPane = null;
      Parametrs ParametrsModel = null;
      List<double[]> resultCulc = null;


      public FMain()
      {
         InitializeComponent();
      }
      #region Релизация интерфейса

      public void Run()
      {
         Application.Run(this);

      }

      public Parametrs GetParametrs()
      {
         return ParametrsModel;
      }

      public void SetRez(List<double[]> result)
      {
         resultCulc = result;
      }
      public event EventHandler MyRun;
      public event EventHandler CulcOk;
      #endregion

      private void FMain_Shown(object sender, EventArgs e)
      {
         Labels();

         if (MyRun != null)

            MyRun(sender, e);
         myPane = zedGraphControl1.GraphPane;
      }


      void Labels()
      {
         label1.Text = Convert.ToString('\u03B4');
         label2.Text = Convert.ToString('\u03BB');
         label3.Text = Convert.ToString('\u03BD');
         label4.Text = "e";
         label5.Text = Convert.ToString('\u03C0');
         label6.Text = Convert.ToString('\u03C0');
         label7.Text = Convert.ToString('\u03C0');
         label8.Text = "A" + Convert.ToString('\u2082');
         label9.Text = "A" + Convert.ToString('\u2083');
         label10.Text = "k" + Convert.ToString('\u00B2');
         label11.Text = Convert.ToString('\u03C7');
```

```
    label12.Text = Convert.ToString('\u03B5');
    label13.Text = Convert.ToString('\u03C4') + Convert.ToString('\u2091');
    this.BackgroundImage =new Bitmap("2.jpg");
    chBoxDelta.Text = Convert.ToString('\u03B4');
    chBoxLaymda.Text = Convert.ToString('\u03BB');
    chBoxNu.Text = Convert.ToString('\u03BD');
    chBoxk2.Text = "k" + Convert.ToString('\u00B2');
    myPane = zedGraphControl1.GraphPane;
    myPane.Title.Text = "";
    myPane.XAxis.Title.Text = "";
    myPane.YAxis.Title.Text = "";
    label14.Text = "A" + Convert.ToString('\u2081');
    label15.Text = "I" + Convert.ToString('\u2081') + Convert.ToString('\u2081');
    label17.Text = "I" + Convert.ToString('\u2082') + Convert.ToString('\u2082');
    label16.Text = "I" + Convert.ToString('\u2083') + Convert.ToString('\u2083');

}

private void butOk_Click(object sender, EventArgs e)
{
    ParametrsModel = new Parametrs();
    ParametrsModel.A1 = Convert.ToDouble(nUpA1.Value);
    ParametrsModel.A2 = Convert.ToDouble(nUpA2.Value);
    ParametrsModel.A3 = Convert.ToDouble(nUpA3.Value);
    ParametrsModel.I11 = Convert.ToDouble(nUpI11.Value);
    ParametrsModel.I22 = Convert.ToDouble(nUpI22.Value);
    ParametrsModel.I33 = Convert.ToDouble(nUpI33.Value);
    ParametrsModel.e = Convert.ToDouble(nUpe.Value);
    ParametrsModel.nu0 = Convert.ToDouble(nUpnu.Value);
    ParametrsModel.G0 = 1;
    ParametrsModel.eps = Convert.ToDouble(nUpeps.Value);
    ParametrsModel.delta0 = Convert.ToDouble(nUpdelta.Value);
    ParametrsModel.laymda0 = Convert.ToDouble(nUplaymda.Value);
    ParametrsModel.k2 = Convert.ToDouble(nUpk2.Value);
    ParametrsModel.xi = Convert.ToDouble(nUpxi.Value);
    ParametrsModel.t_k = Convert.ToDouble(nUptk.Value);
    if (CulcOk != null)
    {
        CulcOk(sender, e);
        Graf();
    }

}

void Graf()
{

    myPane.CurveList.Clear();

    if (chBoxG.Checked)
    {
        PointPairList list = new PointPairList();
        for (int i = 0; i < resultCulc.Count; i++)
            list.Add(resultCulc[i][0], resultCulc[i][1]);
        LineItem myCurve = myPane.AddCurve("", list, Color.Red);
        myCurve.Symbol.IsVisible = false;
    }
    if (chBoxT.Checked)
    {
        PointPairList list = new PointPairList();
        for (int i = 0; i < resultCulc.Count; i++)
            list.Add(resultCulc[i][0], resultCulc[i][2]);
        LineItem myCurve = myPane.AddCurve("", list, Color.Blue);
        myCurve.Symbol.IsVisible = false;
    }
    if (chBoxNu.Checked)
    {
        PointPairList list = new PointPairList();
        for (int i = 0; i < resultCulc.Count; i++)
            list.Add(resultCulc[i][0], resultCulc[i][3]);
        LineItem myCurve = myPane.AddCurve("", list, Color.Yellow);
        myCurve.Symbol.IsVisible = false;
    }
    if (chBoxDelta.Checked)
    {
        PointPairList list = new PointPairList();
        for (int i = 0; i < resultCulc.Count; i++)
            list.Add(resultCulc[i][0], resultCulc[i][4]);
```

```
            LineItem myCurve = myPane.AddCurve("", list, Color.FromArgb(192, 0, 192));
            myCurve.Symbol.IsVisible = false;
        }
        if (chBoxLaymda.Checked)
        {
            PointPairList list = new PointPairList();
            for (int i = 0; i < resultCulc.Count; i++)
                list.Add(resultCulc[i][0], resultCulc[i][5]);
            LineItem myCurve = myPane.AddCurve("", list, Color.Lime);
            myCurve.Symbol.IsVisible = false;
        }
        if (chBoxk2.Checked)
        {
            PointPairList list = new PointPairList();
            for (int i = 0; i < resultCulc.Count; i++)
                list.Add(resultCulc[i][0], resultCulc[i][6]);
            LineItem myCurve = myPane.AddCurve("", list, Color.Black);
            myCurve.Symbol.IsVisible = false;
        }
        zedGraphControl1.AxisChange();
        zedGraphControl1.Invalidate();

    }

    private void butGraf_Click(object sender, EventArgs e)
    {
        Graf();
    }

}


class Model1
  {
    Parametrs p;

    List<double[]> rez = new List<double[]>();

    //G y[0]
    public double f1(double x, double[] y, int n)
    {
        double K, E;
        ClassCulc.Raschet((1 - y[5]), 5e-7, out K, out E);
        double R_k = p.A1 * (p.A2 - p.A3) + p.A3 * (p.A1 - p.A2);
        double W_k = 1 - E / K;
        return -y[0] / R_k * (p.I22 * (p.A1 - p.A3) * W_k + p.I33 * (p.A1 - p.A2) * (y[5] - W_k) + p.I11 * (p.A2 - p.A3) * (1 - W_k));
    }
    //T y[1]
    public double f2(double x, double[] y, int n)
    {
        double K, E;
        ClassCulc.Raschet((1 - y[5]), 5e-7, out K, out E);
        double R_k = p.A1 * (p.A2 - p.A3) + p.A3 * (p.A1 - p.A2);
        double W_k = 1 - E / K;
        double S_k = p.A2 - p.A3 + (p.A1 - p.A2) * y[5];
        return -2 * y[1] / R_k * (p.I22 * (p.A1 - p.A3) * W_k + p.I33 * (p.A1 - p.A2) * (y[5] - W_k) +
            (p.A1 - p.A2) * (p.A1 - p.A3) * (p.A2 - p.A3) / S_k * (p.I33 * (y[5] - W_k) / p.A3 + p.I22 * (1 - y[5]) * W_k / p.A2) + p.I11 * (p.A2 -
p.A3) * R_k / S_k / p.A1 * (1 - W_k));
    }
    //nu y[2]
    public double f3(double x, double[] y, int n)
    {
        return Math.Pow(1 + p.e * Math.Cos(y[2]), 2) / Math.Pow(1 - p.e * p.e, 1.5);
    }
    //delta y[3]
    public double f4(double x, double[] y, int n)
    {
        return 0;
    }
    //laymda y[4]
    public double f5(double x, double[] y, int n)
    {
        double K, E;
        ClassCulc.Raschet((1 - y[5]), 5e-7, out K, out E);
        double N = p.A2 + p.A3 - 2 * p.A1 + 3 * (2 * p.A1 * y[1] / y[0] / y[0] - 1) * (p.A3 + (p.A2 - p.A3) * (K - E) / K / y[5]);
        double h_e = Math.Pow(1 - p.e * p.e, 1.5);

        return 3 * N * Math.Cos(y[3]) / 4 / y[0] / h_e;
```

```
        }
        //k2 y[5]
        public double f6(double x, double[] y, int n)
        {
            double K, E;
            ClassCulc.Raschet((1 - y[5]), 5e-7, out K, out E);

            return (p.I33 * p.A1 - p.I11 * p.A3) / p.A1 / p.A3 * ((1 - p.xi) * (1 - y[5]) - ((1 - p.xi) + (1 + p.xi) * y[5]) * E / K);
        }

        public void Culc()
        {
            rez.Clear();
            const int n = 6;
            double[] y = new double[n+1];
            y[0] = p.G0;
            y[1] = p.G0 * p.G0 * (p.k2 * (p.A1 - p.A2) + (p.A2 - p.A3)) / 2 / ((p.A2 - p.A3) * p.A1 + p.A3 * p.k2 * (p.A1 - p.A2));
            y[2] = p.nu0;
            y[3] = p.delta0;
            y[4] = p.laymda0;
            y[5] = p.k2;

            double[] masrez = new double[n+1];
            for (int i = 0; i < n; i++)
                masrez[i + 1] = y[i];

            rez.Add(masrez);

            ClassCulc.myF[] masFunc = new ClassCulc.myF[n];
            masFunc[0] = f1;
            masFunc[1] = f2;
            masFunc[2] = f3;
            masFunc[3] = f4;
            masFunc[4] = f5;
            masFunc[5] = f6;

            for (double t = 0; t < p.t_k; t = t + p.eps)
            {
                ClassCulc.Runge(t, y, masFunc, n, p.eps);
                masrez = new double[n+1];
                masrez[0] = t + p.eps;
                for (int i = 0; i < n; i++)
                    masrez[i + 1] = y[i];

                rez.Add(masrez);
            }
        }

        public List<double[]> result
        {
            get
            {
                return rez;
            }
        }

       public void SetParametrs(Parametrs P)
        {
            p = P;
        }

    }


class PresenterMain
    {

        private Model1 _model;
        private IMain _mview;

        public PresenterMain(IMain v, Model1 m)
        {
            _model = m;
            _mview = v;

            _mview.MyRun += new EventHandler(_mview_MyRun);
            _mview.CulcOk += new EventHandler(_mview_CulcOk);
        }
```

```
        void _mview_CulcOk(object sender, EventArgs e)
        {
            _model.SetParametrs(_mview.GetParametrs());
            _model.Culc();
            _mview.SetRez(_model.result);
        }

        void _mview_MyRun(object sender, EventArgs e)
        {
            if (ShowSplash != null)
                ShowSplash(this, EventArgs.Empty);
        }

        public void Run()
        {
            _mview.Run();
        }

        public event EventHandler ShowSplash;
    }

interface ISplash
    {
        void RunModal();
        event EventHandler FormClose;
    }

public partial class FSplashScreen : Form, ISplash
    {

        public FSplashScreen()
        {
            InitializeComponent();
        }
        #region Релизация интерфейса

        public void RunModal()
        {
            timer1.Enabled = true;
            ShowDialog();
        }
        public event EventHandler FormClose;
        #endregion

        private void FSplashScreen_FormClosed(object sender, FormClosedEventArgs e)
        {
            if (FormClose != null)
                FormClose(sender, e);
        }

        private void FSplashScreen_MouseClick(object sender, MouseEventArgs e)
        {
            Close();
        }

        private void FSplashScreen_Shown(object sender, EventArgs e)
        {
            label2.Location = new Point(this.Width - label2.Width - 10, this.Height - label2.Height - 5);
            this.BackgroundImage = new Bitmap("sputnik-Cropped.jpg");
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            Close();
        }

        private void FSplashScreen_KeyPress(object sender, KeyPressEventArgs e)
        {
            Close();
        }


    }


class PresenterSplash
    {
```

```
        private ISplash _mview;

        public PresenterSplash(ISplash v)
        {

            _mview = v;
            v.FormClose += new EventHandler(v_FormClose);

        }

        void v_FormClose(object sender, EventArgs e)
        {
            if (FormClose != null)
                FormClose(sender, e);
        }

        public void RunModal()
        {
            _mview.RunModal();
        }
        public event EventHandler FormClose;
    }

class ApplicationController
    {
        PresenterMain pMain;
        Model1 m = new Model1();
        FMain fMain;
        FSplashScreen fSpl;
        PresenterSplash pSplash;

        public ApplicationController()
        {
            fMain = new FMain();
            pMain = new PresenterMain(fMain, m);
            pMain.ShowSplash += new EventHandler(pMain_ShowSplash);

        }

        void pMain_ShowSplash(object sender, EventArgs e)
        {
            FSplashScreen fSpl=new FSplashScreen();
            PresenterSplash pSplash = new PresenterSplash(fSpl);
            pSplash.FormClose += new EventHandler(pSplash_FormClose);
            pSplash.RunModal();
        }

        void pSplash_FormClose(object sender, EventArgs e)
        {
            fMain.WindowState = System.Windows.Forms.FormWindowState.Maximized;

        }
        public void Run()
        {
            pMain.Run();
        }


    }
}
```

ДОДАТОК Б
Код модуля параметрів

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Mar
{
   public partial class Param : Form
   {
      public double r2 = 0.1, R3 = 0.15, r3 = 0.05, ro3 = 0.1, dl0 = 1.04;
      public double    fi1n = 0.2;
      public double fi1tn = -0.5, fi2tn = -0.16;
      public double   m1 = 100, m2 = 30, m3 = 40, m4 = 100;

      public Param()
      {
         InitializeComponent();
      }

      private void Param_Load(object sender, EventArgs e)
      {
         nUDL.Value =Convert.ToDecimal(dl0);
         nUDL.Increment = Convert.ToDecimal(0.01);
         nUDR.Value = Convert.ToDecimal(r2);
         nUDR.Increment = Convert.ToDecimal(0.01);
         nUDr2.Value = Convert.ToDecimal(r3);
         nUDr2.Increment = Convert.ToDecimal(0.01);
         nUDRR2.Value = Convert.ToDecimal(R3);
         nUDRR2.Increment = Convert.ToDecimal(0.01);
         nUDI3.Value = Convert.ToDecimal(ro3);
         nUDI3.Increment = Convert.ToDecimal(0.01);
         nUDFi.Value = Convert.ToDecimal(fi1n);
         nUDFi.Increment = Convert.ToDecimal(0.01);
         nUDV1.Value = Convert.ToDecimal(fi1tn);
         nUDV1.Increment = Convert.ToDecimal(0.01);
         nUDV2.Value = Convert.ToDecimal(fi2tn);
         nUDV2.Increment = Convert.ToDecimal(0.01);
         nUDm1.Value= Convert.ToDecimal(m1);
         nUDm2.Value = Convert.ToDecimal(m2);
         nUDm3.Value = Convert.ToDecimal(m3);
         nUDm4.Value = Convert.ToDecimal(m4);

      }

      private void button1_Click(object sender, EventArgs e)
      {
         r2 = Convert.ToDouble(nUDR.Value);
         R3 = Convert.ToDouble(nUDRR2.Value);
         r3 = Convert.ToDouble(nUDr2.Value);
         ro3 = Convert.ToDouble(nUDI3.Value);
         dl0 = Convert.ToDouble(nUDL.Value);
         fi1n = Convert.ToDouble(nUDFi.Value);
         fi1tn = Convert.ToDouble(nUDV1.Value);
         fi2tn = Convert.ToDouble(nUDV2.Value);
         m1 = Convert.ToDouble(nUDm1.Value);
         m2 = Convert.ToDouble(nUDm2.Value);
         m3 = Convert.ToDouble(nUDm3.Value);
         m4 = Convert.ToDouble(nUDm4.Value);
      }
   }
}
```