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»

«Візуалізація руху механічної системи в середовищі Microsoft Visual Studio»

«Visualization of the movement of a mechanical system in the Microsoft Visual Studio environment»

Виконав: здобувач денної (очної) форми навчання
спеціальності 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Федорів Костянтин Віталійович
(прізвище, ім'я, по-батькові здобувача)

Керівник доцент Іщенко О.В. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент ст.викл. Косирева Л.А.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
механіки, автоматизації та інформаційних технологій
№ ___ від ____.____. 2025 р.

Завідувачка кафедри
_____ Алла РАЧИНСЬКА
(підпис)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.2025 р.

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

Голова ЕК
_____ Микола МАЛАКСІАНО
(підпис)

Одеса 2025

АНОТАЦІЯ

У кваліфікаційній роботі розроблена тема «Візуалізація руху механічної системи в середовищі Microsoft Visual Studio».

Мета роботи – створити Windows-додаток в середовищі Microsoft Visual Studio із застосуванням 3D-графіки, який відображає візуалізацію руху механічної системи з одним ступенем свободи у режимі віртуального часу. Метод дослідження – аналітичний з використанням комп'ютерних технологій.

У кваліфікаційній роботі розроблений Windows-додаток, який здійснює візуалізацію руху механізму, основою якого є механічна система з одним ступенем свободи, у режимі віртуального часу. Модель розроблена за законами механіки – виведено диференціальне рівняння руху механізму за допомогою рівняння Лагранжа II роду, яке згодом вирішувалось чисельно із застосуванням методу Рунге-Кутта IV порядку точності. Додаток створений за допомогою 3D-графіки із застосуванням бібліотеки DirectX і розроблений із використанням комп'ютерного моделювання на мові C# в інтегрованому середовищі Microsoft Visual Studio 19. Проведений ряд комп'ютерних експериментів з різними значеннями параметрів моделі.

ANNOTATION

The qualification work develops the topic "Visualization of the movement of a mechanical system in the Microsoft Visual Studio environment".

The purpose of the work is to create a Windows application in the Microsoft Visual Studio environment using 3D graphics, which displays the visualization of the movement of a mechanical system with one degree of freedom in the virtual time mode. The research method is analytical using computer technology.

In the qualification work has been developed a Windows application that visualizes the movement of a mechanism based on a mechanical system with one degree of freedom in the virtual time mode. The model was developed according to the laws of mechanics - the differential equation of the mechanism's motion was derived using the Lagrange equation II, which was subsequently solved numerically using the Runge-Kutt method of the fourth order of accuracy. The application was created using 3D graphics using the DirectX library and developed using computer modeling in the C# language in the Microsoft Visual Studio 19 integrated environment. A number of computer experiments were carried out with the various values of the model parameters.

ЗМІСТ

ВСТУП

Незамінним ефективним методом сучасного теоретичного дослідження фізичних явищ і технологічних процесів є чисельне моделювання.

Цей підхід нині інтенсивно розвивається: будуються вдосконалені моделі, конструюються нові чисельні алгоритми, проводяться різноманітні обчислювальні експерименти, проектуються кластерні системи.

Бурхливий розвиток комп'ютерної техніки, доступність обчислювальних потужностей, прогрес в операційних системах персональних комп'ютерів, у тому числі обчислювальних процесів, що забезпечують паралельність, тільки посилюють інтерес до проектування і реалізації чисельних експериментів у фізиці і механіці.

Чисельне моделювання може бути використане для передбачення і вивчення поведінки (руху) складних фізичних і механічних систем. Для того, щоб дати кількісно правильні передбачення, моделювання повинне адекватно описувати як окремі процеси, що діють в системі, так і їх взаємодію..

Після того, як модель побудована і випробувана, її можна використати для інтерпретації вимірів і спостережень, для поширення теоретичних моделей на область нових режимних параметрів, для оцінки нових ідей і рішень. Найважливішим застосуванням чисельних моделей є перевірка точності і повноти нашого розуміння фізико-механічної системи шляхом порівняння результатів розрахунків з експериментальними даними або спрощеними аналітичними рішеннями.

Навіть прості натурні експерименти частенько виявляються складними і ресурсоємними, у зв'язку з чим модельні рівняння, що описують їх, зазвичай не мають аналітичних рішень і повинні вирішуватися чисельно.

У дипломній роботі виконано чисельне моделювання руху для механічної системи з одним ступенем свободи. Моделювання супроводжується процесом візуалізації руху механізму із застосуванням 3D-графіки за допомогою графічної бібліотеки DirectX.

1 ПОСТАНОВКА ЗАВДАННЯ

До кола диска радіуса R шарнірно приєднаний важіль, який несе на своїх кінцях зосереджені маси m_1 і m_2. Відстані мас від шарніра C відповідно рівні l_1 і l_2. Диск обертається у вертикальній площині навколо горизонтальної вісі, перпендикулярної його площині, з кутовою швидкістю ω. Масою важеля знехтувати. Вісь обертання важеля паралельна осі обертання диска.

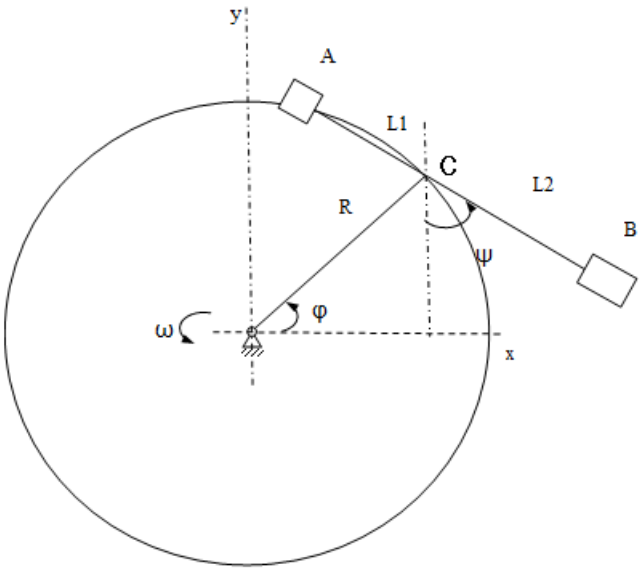


Рисунок 1.1 – Схема механізму

Необхідно:

1) Скласти диференціальне рівняння руху важеля.

2) Створити Windows-додаток на мові C# в середовищі Microsoft Visual Studio, який відображає візуалізацію руху механізму на екрані монітора у режимі віртуального часу із застосуванням 3D-графіки.

2 РІВНЯННЯ РУХУ МЕХАНІЗМУ І ЙОГО ЧИСЕЛЬНИЙ РОЗВ'ЯЗОК

2.1 Виведення рівняння руху механізму

Система тіл, що рухаються, має один ступінь свободи, так як її положення визначається одним параметром. В якості цього параметру можна вважати кут ψ повороту важеля, який відраховується від вертикалі за проти годинникової стрілки. Цей кут ми приймемо за узагальнену координату системи. Тоді рух системи можна описати рівнянням Лагранжа II роду [1 - 3]:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\psi}}\right) - \frac{\partial T}{\partial \psi} = Q_{\psi}, \tag{2.1}$$

де ψ і $\dot{\psi}$ – узагальнена координата і узагальнена швидкість відповідно;

Т – кінетична енергія механічної системи;

Q_{ψ} - узагальнена сила, що відповідає координаті ψ. Вона обчислюється за формулою:

$$\mathrm{Q}_{\psi} = -\frac{\partial П}{\partial \psi}, \tag{2.2}$$

де П – потенційна енергія системи.

Обчислимо кінетичну та потенційну енергії системи. Кінетична енергія T системи визначається як сума кінетичних енергій T_1 вантажу 1 и T_2 вантажу 2:

$$T = T_1 + T_2 \ . \tag{2.3}$$

Вантажі приймемо за матеріальні точки. Кінетична енергія вантажу 1, який зображений на рисунку 1 точкою А, визначається за формулою:

$$T_1 = \frac{m_1 v_1^2}{2} , \qquad (2.4)$$

де V_1 - абсолютна швидкість тіла 1. Це тіло здійснює абсолютний рух, тому його швидкість обчислюється за формулою:

$$\bar{v}_1 = \bar{v}_1^r + \bar{v}_1^e .$$

де v_1^r і v_1^e - відносна і переносна швидкості тіла 1відповідно.

На рисунку 2.1 зображений механізм та вектори відносної та переносної швидкостей точок А і В, а також швидкість обертання точки С, яка належить як диску, так і важелю, бо у цій точці важель прикріплюється шарнірно до диска.

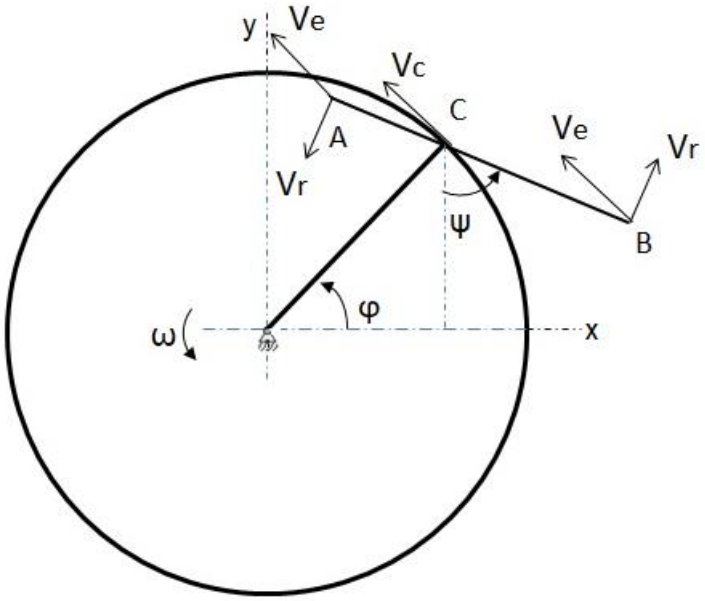


Рисунок 2.1 – Кінематична схема механізму

У відносному русі вантаж 1 обертається навколо горизонтальної осі, що проходить через точку С, тому його відносна швидкість дорівнює $v_1^r = l_1\dot{\psi}$.

У переносному русі вантаж 1 обертається разом з диском навколо горизонтальної осі, що проходить через точку О. Тому його переносна швидкість дорівнює $v_1^e = v_C = R\dot{\varphi}.$

Тоді формула для знаходження квадрата абсолютної швидкості точки А має вигляд:

$$v_1^2 = v_1^{r2} + v_1^{e2} + 2v_1^r v_1^e \cos\alpha, \text{ де } \alpha = \frac{\pi}{2} - \varphi + \psi.$$

А кінетична енергія T_1 вантажу 1 згідно з формулою (2.4) визначається таким чином:

$$T_1 = \frac{m_1}{2}(R^2\dot{\varphi}^2 + l_1^2\dot{\psi}^2 + 2Rl_1\dot{\varphi}\dot{\psi}\sin(\varphi - \psi)). \quad (2.5)$$

Кінетична енергія T_2 вантажу 2, який зображений на рисунку точкою В, визначається за формулою:

$$T_2 = \frac{m_2 v_2^2}{2}. \quad (2.6)$$

У відносному русі вантаж 2 обертається навколо горизонтальної осі, що проходить через точку С, тому відносна його швидкість $v_2^r = l_2\dot{\psi}.$

У переносному русі вантаж 2 обертається разом з диском навколо горизонтальної осі, що проходить через точку О. Тому його переносна швидкість дорівнює $v_2^e = v_C = R\dot{\varphi}.$

Тоді формула для знаходження квадрата абсолютної швидкості точки B має вигляд:

$$v_2^2 = v_2^{r2} + v_2^{e2} + 2v_2^r v_2^e \cos\beta, \quad \text{де} \quad \beta = \frac{\pi}{2} + \varphi - \psi.$$

І кінетична енергія T_2 вантажу згідно з формулою (2.6) визначиться наступною формулою:

$$T_2 = \frac{m_2}{2}(R^2\dot{\varphi}^2 + l_2^2\dot{\psi}^2 - 2Rl_2\dot{\varphi}\dot{\psi}\sin(\varphi - \psi)) \tag{2.7}$$

Підставимо формули (2.5) і (2.7) в (2.3) і отримаємо кінетичну енергію системи, яка буде дорівнювати:

$$\begin{aligned} T_1 &= \frac{m_1}{2}(R^2\dot{\varphi}^2 + l_1^2\dot{\psi}^2 + 2Rl_1\dot{\varphi}\dot{\psi}\sin(\varphi - \psi)) + \\ &+ \frac{m_2}{2}(R^2\dot{\varphi}^2 + l_2^2\dot{\psi}^2 - 2Rl_2\dot{\varphi}\dot{\psi}\sin(\varphi - \psi)). \end{aligned} \tag{2.8}$$

А тепер обчислимо похідні від кінетичної енергії T по узагальненій координаті ψ і по узагальненій швидкості $\dot{\psi}$, необхідних для формули (2.1):

$$\frac{\partial T}{\partial \psi} = -(m_1 l_1 - m_2 l_2)R\dot{\varphi}\dot{\psi}\cos(\varphi - \psi);$$

$$\frac{\partial T}{\partial \dot{\psi}} = (m_1 l_1^2 + m_2 l_2^2)\dot{\psi} + R\dot{\varphi}\sin(\varphi - \psi)(m_1 l_1 - m_2 l_2).$$

Повна похідна від останнього виразу за часом має вигляд:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{\psi}}\right) = (m_1 l_1^2 + m_2 l_2^2)\ddot{\psi} + R\dot{\phi}(m_1 l_1 - m_2 l_2)\cos(\phi - \psi)(\dot{\phi} - \dot{\psi}). \quad (2.9)$$

Знайдемо узагальнену силу. Для цього знайдемо потенційну енергію системи. Вона визначається як сума потенційних енергій тіл 1 і 2:

$$П = П_1 + П_2. \quad (2.10)$$

Потенційна енергія тіла 1 дорівнює:

$$П_1 = m_1 g h_1 = m_1 g l_1 (1 - \cos\psi), \quad (2.11)$$

де g – прискорення вільного падіння;

h_1 – висота, на яку опускається тіло 1, якщо важель повертається на кут ψ, який відраховується від вертикалі.

Потенційна енергія тіла 2 дорівнює:

$$П_2 = m_2 g h_2 = m_2 g l_2 (1 - \cos\psi), \quad (2.12)$$

де h_2 – висота, на яку піднімається тіло 2, якщо важель повертається на кут ψ.

Підставляючи усі знайдені величини (2.11) і (2.12) в рівність (2.9), отримаємо $П = g(1 - \cos\psi)(m_1 l_1 - m_2 l_2)$.

Тоді $\frac{\partial П}{\partial \psi} = g(m_1 l_1 - m_2 l_2)\sin\psi,$

а узагальнена сила дорівнює

$$Q_\psi = -g(m_1 l_1 - m_2 l_2)\sin\psi. \quad (2.13)$$

Прирівнявши ліві і праві частини рівняння Лагранжа II роду, тобто підставивши формули (2.19) та (2.13) в (2.1), отримуємо рівняння:

$$(m_1 l_1^2 + m_2 l_2^2)\ddot{\psi} - R\dot{\varphi}^2(m_1 l_1 - m_2 l_2)\cos(\varphi - \psi) + g(m_1 l_1 - m_2 l_2)\sin\psi = 0. \qquad (2.14)$$

Отримане рівняння (2.14) є диференціальним рівнянням руху важеля. Це рівняння не має аналітичного рішення, тому рішення поставленої задачі зводиться до чисельного інтегрування отриманого диференціального рівняння з наступними початковими умовами:

при t = 0: $\psi = \psi_0$, $\dot{\psi} = \dot{\psi}_0$.

Для чисельного інтегрування отриманого диференціального рівняння застосуємо метод Рунге-Кутта IV порядку точності.

Чисельне рішення рівняння (2.14) буде використовуватись у Windows-додатку для того, щоб розрахувати положення усіх тіл, що становлять механізм, в кожен розрахунковий момент часу.

2.2 Чисельне рішення диференціального рівняння руху механізму

Отримане рівняння (2.14) є диференціальним рівнянням руху важеля. Це диференціальне рівняння другого порядку, і аналітичне його розв'язання загалом дуже складне. Тому скористаємось чисельним розв'язком цього рівняння.

Для чисельного інтегрування отриманого диференціального рівняння (2.14) застосуємо метод Рунге-Кутта IV порядку точності [4]. Цей метод дозволяє ефективно моделювати динамічні процеси, а це надає можливість

аналізувати поведінку навіть складних систем. Зведемо рівняння (2.14) до виду $\ddot{\psi} = f(\psi, t)$ і отримаємо:

$$\ddot{\psi} = \frac{R\omega^2(m_1 l_1 - m_2 l_2)\cos(\psi - \omega t) - (m_1 l_1 - m_2 l_2) g \sin\psi}{(m_1 l_1 + m_2 l_2)}. \qquad (2.15)$$

Метод Рунге-Кутта використовується для побудови однокрокових методів. Хоча у нас диференціальне рівняння другого порядку, ми можемо до нього застосувати метод Рунге-Кутта для звичайних диференціальних рівнянь першого порядку. Для цього необхідно рівняння другого порядку привести до системи рівнянь першого порядку:

$$\begin{cases} y' = w; \\ w' = f(x, y, w). \end{cases}$$

А далі вже до кожного з рівнянь необхідно застосувати метод Рунге-Кутта для звичайних диференціальних рівнянь першого порядку. Найбільш точним методом Рунге-Кутта є метод четвертого порядку точності, якім ми і скористаємося. Розрахункова схема цього методу для рівняння $y'' = f(x, y)$ має наступний вигляд:

$$
\begin{cases}
y'_{i+1} = y'_i + \dfrac{1}{6}\left(k_1 + 2\cdot k_2 + 2\cdot k_3 + k_4\right); \\
y_{i+1} = y_i + h\left(y'_i + \dfrac{1}{6}\left(k_1 + k_2 + k_3\right)\right); \\
k_1 = h\cdot f\left(x_i, y_i, y'_i\right); \\
k_2 = h\cdot f\left(x_i + \dfrac{h}{2}, y_i + \dfrac{h}{2}y'_i + \dfrac{h}{8}k_1, y'_i + \dfrac{k_1}{2}\right); \\
k_3 = h\cdot f\left(x_i + \dfrac{h}{2}, y_i + \dfrac{h}{2}y'_i + \dfrac{h}{8}k_2, y'_i + \dfrac{k_2}{2}\right); \\
k_4 = h\cdot f\left(x_i + h, y_i + hy'_i + \dfrac{h}{2}k_3, y'_i + k_3\right).
\end{cases}
\qquad (2.16)
$$

В якості параметра *x* підставляється параметр *t*, а в якості функції y - функція $\psi(t)$.

При чисельному розрахунку методом Рунге-Кутта даний проміжок часу розбивається на відрізки завдовжки Δt (цей параметр у схемі (2.16) позначений літерою *h)*. На цьому відрізку необхідно знайти рішення диференціального рівняння в кожній точці інтегральної кривої, що проходить через цю точку при виконанні початкових умов, причому існування і єдиність такої кривої забезпечується теоремою Пікара. А по значеннях функції $\psi(t)$ та її похідної на початку кожного відрізку часу по формулах схеми (2.16) розраховуються значення $\psi(t+h)$ та її похідної у кінці відрізку часу. Таким чином, ми отримаємо набір обчислених значень функції та її похідної в ряді точок обраного тимчасового інтервалу, які використовуємо для візуалізації механізму на цьому часовому інтервалі.

3 3D-ГРАФІКА І БІБЛІОТЕКА DIRECTX

3.1 3D-графіка

Тривимірна графіка, або 3D-графіка – це технологія, яка дозволяє створювати та відображати об'єкти у тривимірному просторі. Двовимірні зображення є плоскими картинками, а тривимірна графіка відрізняється від двовимірної тим, що дозволяє моделювати та візуалізувати об'єкти та сцени, надаючи їм має глибину, просторовість (об'єм) і реалістичність.

Основою 3D-графіки є математичне моделювання, яке використовується для створення тривимірних об'єктів.

3D-об'єктом називають об'ємне тіло, у якого є довжина, ширина і глибина. Цей об'єкт характеризується формою і текстурою [5].

Форма — це геометрія об'єкта, що звичайно у тривимірному просторі описується серією взаємозалежних точок (вершин) й багатокутників (граней). Грані є замкненими двомірними фігурами із трьома або більш сторонами.

Дивлячись на тривимірну графічну картинку на моніторі комп'ютера, бачимо зображення, яке створене із різних форм. Це прямі лінії, квадрати, прямокутники, паралелограми, ромби та кола. Для простоти в якості складових базових багатокутників можна вибрати трикутники, які треба зкомпонувати у складні тривимірні сітки, щоб скласти достовірну картинку з кривими лініями, як у навколишньому світі.

Разом вони утворюватимуть структуру, яку називають каркасом. Каркас дуже нагадує ескіз об'єкта. І на наступному кроці цей каркас повинен одержати поверхню з його текстурою.

Текстура поверхні — це простий опис властивостей поверхні: колір, прозорість і так далі.

Насправді 3D-об'єкти існують тільки в пам'яті комп'ютера. А для його відображення на екрані монітора необхідно вирішити завдання відображення 3D-тіла на плоскій, двовимірній поверхні екрану. А для цього доводиться виконувати деякі серйозні математичні розрахунки. Це завдання рендерингу. Рендеринг – це перетворення тривимірної моделі предмета на «плоске» зображення. З математичної точки зору рендеринг перетворює тривимірну математичну просторову модель (векторну структуру даних) на плоску картинку (плоску матрицю пікселів). Цей крок вимагає складних обчислень. Найпростіший вид рендерингу - це побудувати контури моделей на екрані комп'ютера за допомогою проекції. Існує кілька типів технології рендерингу, кожна з яких має свої плюси та мінуси: z-буфер, сканлайн, трасування променів, глобальне освітлення. використовує переважно z-буфер.

Одним із засобів комп'ютерного моделювання тривимірних об'єктів є бібліотека DirectX, яка використовує переважно z-буфер.

Як було зазначено вище, 3D-моделювання дозволяє визначити форму об'єкту, його розміри, текстури та інші його властивості. Всі об'єкти повинні бути розміщені на сцені за допомогою геометричних перетворень відповідно до вимог майбутнього зображення.

При зображені тривимірного об'єкту з'ясовується, які частини об'єкта будуть видні із заданої точки огляду. Крім того, для того, щоб об'єкт виглядав об'ємним, в зображенні треба позначити перспективу. 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, і чи повинні виникати які-небудь тіні та тому подібне.

Тому для отримання 3D-зображення об'єкта необхідно виконати наступні кроки [5]:

1. Створення сцени.

3D-зображення і тривимірна сцена проектуються на двовимірну площину, яку називають вікном (або областю) перегляду і яка є базовим контейнером для всіх 3D об'єктів.

2. Камера.

Щоб побачити створені об'єкти, необхідно вибрати положення в просторі, з якого будемо спостерігати за об'єктами. Ця точка спостереження називається камерою, і вона встановлює, як сцена чи об'єкт будуть відображатися. А це означає, що треба для кожної сцени вибрати точку перегляду, з якої такі камери визначають, де по відношенню до сцени позиціонується об'єкт для його спостереження користувачем. Ці камери об'єднуються з тими, що мають інші властивості, наприклад, визначають перспективу і вміють змінювати розмір зображення. І таким чином виходить кінцевий результат рендеринга 3D-сцен на 2D-екрані.

3. Джерела світла.

3D-рендеринг вимагає наявність джерел світла. Процес їх розміщення і їх налаштування у створеній сцені здійснюється таким чином, щоб освітлення надавало сцені відчуття об'ємності та реальності. Джерело світла може висвітлювати всі об'єкти, або один об'єкт чи його частину. Вони здатні створювати тіні, коли промені падають на об'єкти. Графічні 3D-редактори використовують такі види джерел світла: omni light (всеспрямоване світло), spot light (промені, які розходяться), directional light (паралельні промені) або навіть джерела об'ємного світіння (Sphere light).

4. Матеріал.

Матеріалом тут може бути колір або текстура у вигляді картинки. накладеної на предмет. Саме матеріал разом із освітленням і визначає зовнішній вигляд тривимірного об'єкта.

3.2 Бібліотека DirectX

DirectX – сукупність технологій від корпорації Microsoft, що призначена для роботи з графічними та мультимедійними програмами на платформі Windows. Це інструментарій для створення високопродуктивних графічних програм та комп'ютерних ігор з повнокольоровою графікою, відео, тривимірною анімацією та об'ємним звуком. Direct3D - це набір програмних інтерфейсів API для малювання примітивів з конвеєром відтворення або для виконання паралельних операцій із шейдером обчислень. DirectX має кілька компонентів, які призначені для різних цілей [5, 6]:

- DirectDraw — це компонент для роботи з двовимірною графікою. Він дозволяє відображати та обробляти зображення, текстури та спрайти, має швидкий доступ до відеопам'яті.
- Direct2D — це бібліотека для створення та рендерингу двомірної векторної графіки. Компонент підтримує різні ефекти та трансформації забезпечує високу якість відображення,.
- Direct3D — це компонент для роботи із тривимірною графікою. Він надає багато API для моделювання об'єктів, створення та управління 3D-сценами, текстури, налаштування освітлення тощо.
- DirectXMath – це бібліотека для математичних операцій та перетворень у графіці. Вона може виконувати обчислення з векторами, матрицями та іншими геометричними об'єктами.
- DirectSound та DirectMusic — це компоненти для роботи зі звуком. За їх допомогою можна створювати звукові ефекти та багатоканальні аудіопотоки, керувати гучністю і відтворювати музику в реальному часі.

3.3 Використання Mesh-об'єктів

Усі фігури, що промальовуються за допомогою Direct3D на сцені, являють собою сукупність точок – вершин. Кожна вершина містить такі параметри, як її положення в просторі, колір вершини, текстура і інші [5].

У DirectX є об'єкт, який може інкапсулювати збережені та завантажувані вершини, а також індексувати дані. Такий об'єкт називається Mesh-об'єктом. Ці об'єкти призначені для формування складних моделей і можуть використовуватись для збереження будь-якого типу графічних даних.

Для створення реалістичної моделі об'єкта зазвичай використовують такі геометричні примітиви, як прямокутник, куб, куля, конус та деякі гладкі, так звані сплайнові, поверхні. У DirectX для створення таких готових 3D-примітивів існує ціла низка Mesh-об'єктів.

Зображена мною система представлена у вигляді окремих об'єктів, які зібрані переважно з Mesh-об'єктів циліндр і сфера.

Приведемо опис деяких Mesh-об'єктів.

Для створення циліндру існує відповідний метод Cylinder(). Він центрується на початку координат, його вісь поєднана з віссю *z*. Синтаксис цього метода наступний:

mesh=Mesh.Cylinder(Device device, float radiusl, float radius2, float length, int slices, int stacks)

Параметри методу:

- radiusl - радіус циліндра на від’ємному кінці осі Z. Значення цього параметра має бути більший або рівне 0.0f;
- radius2 - радіус циліндра на додатньому кінці осі Z. Значення цього параметра має бути більший або рівне 0.0f;
- length - довжина циліндра уздовж осі Z;

- slices - число секторів (slices) уздовж головної осі (чим більше це значення, тим більше вершин);
- stacks - число стеків уздовж головної осі (чим більше це значення, тим більше вершин).

Метод Sphere() призначений для створення сфери, що центрується на початку координат, а її вісь є вісь Z. Синтаксис цього метода наступний:

mesh = Mesh.Sphere(Device device, float radius, int slices, int stacks);

Параметри методу:

- radius - радіус сфери. Це значення має бути більше або дорівнювати 0;
- slices - число секторів (slices) уздовж головної вісі (чим більше це значення, тим більше вершин) ;
- stacks - число стеків вздовж головної осі (більше значення додасть більше вершин чим більше це значення, тим більше вершин).

Метод Torus() малює кільцеву форму з центром у точці (0, 0, 0), вісь якої є вісь Z. Синтаксис цього метода наступний:

mesh = Mesh.Torus(Device device, float innerRadius, float outerRadius, int sides, int rings);

Параметри методу:

- innerRadius - внутрішній радіус тора, тобто радіус поперечного перерізу (малий радіус). Це значення має бути більше або дорівнювати нулю;
- outerRadius - зовнішній радіус тора. Це значення має бути більше або дорівнювати трьом;
- rings - число кілець у поперечному перерізі тора. Це значення має бути більше або дорівнювати трьом;

- sides - кількість сторін у поперечному перерізі; воно має бути більше або дорівнювати 3.

Метод Polygon() призначений для створення багатокутника, що центрується на початку координат. Синтаксис цього метода наступний:

mesh = Mesh.Polygon(Device device, float length, int sides);

Параметри методу:

- length - довжина кожної сторони полігона;
- sides - число сторін полігона.

Метод Box() призначений для створення паралелепіпеду, що містить вершину та індекси, необхідні для рендеринга куба із заданою висотою, шириною й глибиною. Створений блок центрується на початку координат. Синтаксис цього метода наступний:

mesh = Mesh.Box(Device device, float width, float height, float depth);

Параметри методу:

- width - визначає розмір куба за шириною уздовж осі X;
- height - визначає розмір куба за висотою уздовж осі Y;
- depth - визначає розмір куба за глибиною уздовж осі Z.

Усі вказані об'єкти в початковий момент розташовуються на сцені з центром на початку координат. Після цього центри тяжіння об'єктів можуть бути перенесені у відповідні точки простору сцени за допомогою методів Translation(), RotationX(), RotationY(), RotationZ().

Кожна сцена для малювання має точку перегляду, з якої користувач може за нею спостерігати. У 3D-системах зазвичай використовуються камери, які визначають, де по відношенню до сцени позиціонується об'єкт для його спостереження користувачем. Іноді використовуються декілька

камер, які визначають перспективу. ЇЇ робота визначається в тому, що ті об'єкти, які розташовані далі, стають менше. Кінцевий результат цих камер дає 3D-зображення на 2D-екрані [2].

Камера управляється за допомогою матриць перетворень. Ці дії використовуються для визначення того, як сцена повинна проектуватися на екран. Один із способів створення матриці проекції полягає у використанні метода Microsoft.DirectX.Matrix.PerspectiveFovLH() структури Matrix. Цей метод створює матрицю проекції лівої системи координат, яка моделює точку зору користувача, коли він дивиться на екран монітора. Перетворення проекції необхідне для опису сцени, що представлена у вигляді усіченої піраміди, січні площини якої визначають перспективу.

Синтаксис методу PerspectiveFovLH() має вигляд:

```
public static Microsoft.DirectX.Matrix.PerspectiveFovLH(float
 fieldOfViewY,  float aspectRatio,  float znearPlane,  float zfarPlane)
```

Параметри методу:

- fieldOfViewY - точка зору камери;
- aspectRatio - форматне співвідношення сторін;
- znearPlane - ближній план (по Z-координаті);
- zfarPlane - далекий план (по Z-координаті).

Параметр поля зору визначається кутом при вершині піраміди. Форматне співвідношення сторін будь-якого з перерізів піраміди аналогічно форматам телебачення. Наприклад, широкоформатне телебачення має співвідношення 1.85. Це показує відношення ширини зображення до висоти. Direct3D промальовував тільки ті об'єкти, які потрапляють всередину усіченої піраміди.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ WINDOWS-ДОДАТКУ ТА ІНСТРУКЦІЯ ПО ЙОГО ВИКОРИСТАННЮ

4.1 Опис Windows–додатку

Для комп'ютерної реалізації поставленого завдання мною розроблений Windows–додаток, який візуалізує рух механізму на екрані монітора в режимі віртуального часу із застосуванням 3D-графіки. Додаток написаний на мові C# в інтегрованому середовищі розробки Microsoft Visual Studio 19 [7-10]. До додатку підключені дві графічні бібліотеки DirectX і DirectX.Direct3D за допомогою директиви using:

```
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
```

Для 3D-зображення усіх складових механізму в даному проекті використані готові Mesh-об'єкти з інтерфейсу Microsoft DirectX.

Після запуску додатка на екрані з'являється форма під назвою «Візуалізація роботи механізму». На формі присутнє меню у вигляді компоненту menuStrip, яке складається з трьох пунктів: «Механізм», «Рух» та «Вихід».

Пункти меню дозволяють керувати візуалізацією руху механізму: зобразити механізм на екрані монітора, запустити додаток на виконання із зображенням візуалізації руху, призупинити виконання додатку чи очистити форму. Промальовування механізму, що рухається в режимі віртуального часу, відбувається по центру форми.

Праворуч вгорі розташована панель для введення користувачем основних початкових геометричних та кінематичних параметрів механічної системи. Для завдання параметрів використані компоненти NumericUpDown,

які дозволяють вводити параметри в певному діапазоні. У цих компонентах вже обрані деякі початкові значення за замовчуванням.

Внизу розташовані 2 компонента – hScrollBar2 і hScrollBar1, ліворуч – vScrollBar1, призначені для повороту огляду перегляду навколо горизонтальної та вертикальної вісей, а також для змінення масштабу зображення.

Нижче показаний вигляд цієї сторінки, на якій зображений механізм в деякому початковому положенні:

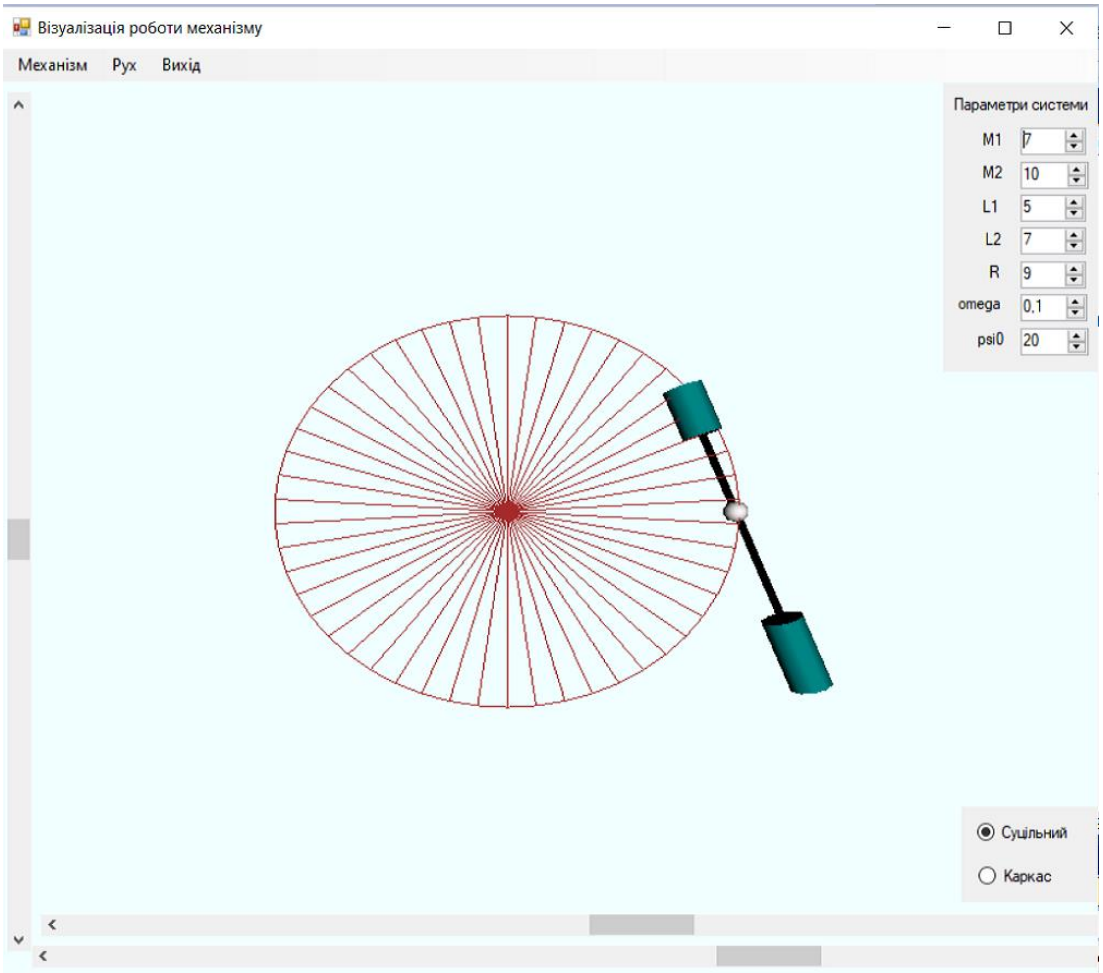


Рисунок 4.1 – Основна форма додатку

4.2 Опис програмного коду додатку

Спочатку опишемо докладніше пункти меню з точки зору програмного коду.

Пункт меню "Механізм" складається з 2 пунктів: «Намалювати» та «Стерти».

Пункт меню Механізм → Намалювати дозволяє зображувати механізм на формі. Якщо потім змінювати окремі геометричні параметри механізму, то ці зміни будуть одночасно відображатись і на малюнку – будуть видозмінюватися початкові положення його складових.. Наприклад, збільшення або зменшення довжина важелів у випадку зміни l_1 або l_2. Маси m_1 і m_2 на формі зображені циліндрами, причому більшій масі відповідає більша довжина циліндра. Але довжина не зростає пропорційно до величини маси вантажу, а зображується в масштабі по відношенню до деякої довжини, обраної в якості характерного розміру.

Промальовування механізму здійснює функція Draw(). Вона зображує механізм, який складається з диску і двох вантажів, сполучених між собою за допомогою важеля, прикріпленого до диску у точці С. У цю функцію передаються всі параметри складових механізму. Зображення усіх складових механізму складається з окремих Mesh-об’єктів. Для промальовування кожного вантажу використовується Mesh-об'єкт циліндр. Обидва циліндри мають однакові діаметри, але різні довжини (в залежності від їх мас). Важелі теж зображені за допомогою циліндрів. Диск зображений за допомогою сфери. Підшипник у місці приєднання важеля до диска теж зображений за допомогою сфери маленького радіусу. Диск насаджений у центрі на вісь, закріплену якимось чином у просторі. Але на зображеному механізмі ми цієї осі бачимо, так як в цій точці сходяться всі промені каркаса диска, які завжди нам видно, і завдяки наявності яких ми можемо простежити за обертанням цього диска.

Метод ChangeData() зчитує значення всіх геометричних і кінематичних параметрів з відповідних компонент вводу и виконує їх перерахунок у розрахункові значення для промальовування її на формі у відповідному масштабі. Метод Creation() виконує підготовчу роботу по створенню відповідних Mesh-об'єктів.

Перед промальовуванням всього механізму спочатку, як було описано в теоретичній частині роботи, налаштовується камера. Для цього викликаються послідовно методи Clear(), BeginScene() та SetupProekcii() для об'єкта класу Device. Вони містять ряд підготовчих налаштувань для створення зображення – освітлення та рендерингу.

В подієвій функції Form1_Load() створюються для кожного тіла відповідні Mesh-об'єкти, вибираються матеріали та кольори.

У методі Draw() здійснюється промальовування всіх складових механізму у вигляді відповідних Mesh-об'єктів в даний розрахунковий момент часу. В початковий момент на сцені всі об'єкти розміщуються на площині XY з центром у початку координат цієї площини. Але кожне тіло повинно розміщуватись у відповідному місці у просторі, тому для переміщення тіл на своє місце до них застосовуються афінні перетворення. Для цього ми кожен з об'єктів спочатку переносимо з центру у відповідне розраховане на площині положення, а потім повертаємо його відповідно до розрахованого за формулою (2.14) кута повороту об'єкта навколо вертикальної осі Z. Потім до об'єкта застосовуються повороти навколо горизонтальної та вертикальної вісей (X та Y) на ті кути, які вибрав користувач на горизонтальній та вертикальній лінійках прокручування форми. Прокручування має діапазон значень від 0 до 180 градусів зі значенням у початковий момент часу 90^0. Далі до об'єкта застосовується масштабування (збільшення всього механізму або зменшення) відповідно до того коефіцієнта, який вибрав користувач на нижній горизонтальній лінійці прокручування. Всі ці дії виконуються за допомогою методів Translation(),

RotationX(), RotationY(), RotationZ() класу Matrix, які застосовуються в описаному вище порядку

Під час промальовування всіх тіл враховується значення, отримане від радіокнопки, що задає зовнішній вигляд циліндрів - каркасний або суцільний. Залежно від цього значення властивість RenderState.FillMode змінної класу Device приймає значення FillMode.WireFrame або FillMode.Solid. При цьому диск завжди промальовується в каркасному вигляді, а вигляд інших тіл у механізмі визначає користувач.

При клацанні на пункт меню Механізм → Стерти очищуються всі буфери, і таким чином зображення механізму очищується. При цьому відновлюються всі початкові налаштування лінійок прокручування.

Рух механізму в програмі пов'язаний з компонентом Timer. Він запускається за допомогою команди «Запустити» в меню «Рух». Через заданий в програмі інтервал часу викликається подія OnTick таймера, в обробнику якого, а саме у методі timer1_Tick(), відбувається низка розрахунків, яка приводить к розрахунку положень складових у наступний момент часу. Ідея реалізації процесу візуалізації руху механічної системи побудована таким чином: зображується механізм в початковому положенні, потім у кінці кожного тимчасового інтервалу перераховуються положення усіх складових механізму у наступний момент часу, потім екран очищується, і механізм зображується в новому положенні. Таким чином, на екрані користувач бачить безперервну картину руху механізму

А от розрахунком положення усіх складових механізму у наступний момент часу займається ціла низка функцій.

Чисельний розрахунок диференціального рівняння руху механізму (2.15) за методом Рунге-Кутта IV порядку точності виконує метод rungekuttIV(), який має наступний прототип:

```
rungekuttIV(double h, double y0, double y10, ref double y1, ref double y11)
```

Цей метод запрограмований за схемою (2.16). В якості параметрів він приймає розрахунковий інтервал за часом Δt, початкові значення куту ψ відхилення важеля від вертикальної осі та його похідної на початок розрахункового інтервалу за часом t_i. Метод повертає значення куту ψ відхилення важеля від вертикальної осі та його похідної на кінець розрахункового тимчасового інтервалу за часом $t_i + \Delta t$. За цим значеннями перераховуються усі координати положення та параметри складових тіл механізму і за допомогою методу Draw() нове положення механізму відображається на формі замість колишнього. Таким чином з часом виходить картина змін положення механізму на формі.

Цей метод використовує всередині себе функцію func(double psi), яка повертає значення правої частини рівняння (2.15) для даного значення вхідного параметра - кута ψ відхилення важеля від вертикальної осі.

За допомогою компонента vScrollBar1 можна повертати площину перегляду навколо горизонтальної осі X. Для цього в програмі передбачений метод vScrollBar1_Scroll().

За допомогою компонента hScrollBar2 можна повертати площину перегляду навколо вертикальної осі. Для цього в програмі передбачений метод hScrollBar2_Scroll().

За допомогою компонента hScrollBar1 можна змінювати масштаб зображення як у бік його збільшення, так і у бік його зменшення. Для цього в програмі передбачений метод hScrollBar1_Scroll().

Тіло останніх трьох методів складається лише з одного оператора:

```
this.Invalidate();
```

Використання цього методу робить недійсною конкретну область елемента управління та викликає його перемальовування. Значення Value

компонентів vScrollBar і hScrollBar використовуються у функції Draw() при визначенні положення Mesh-об'єктів в даний розрахунковий момент часу.

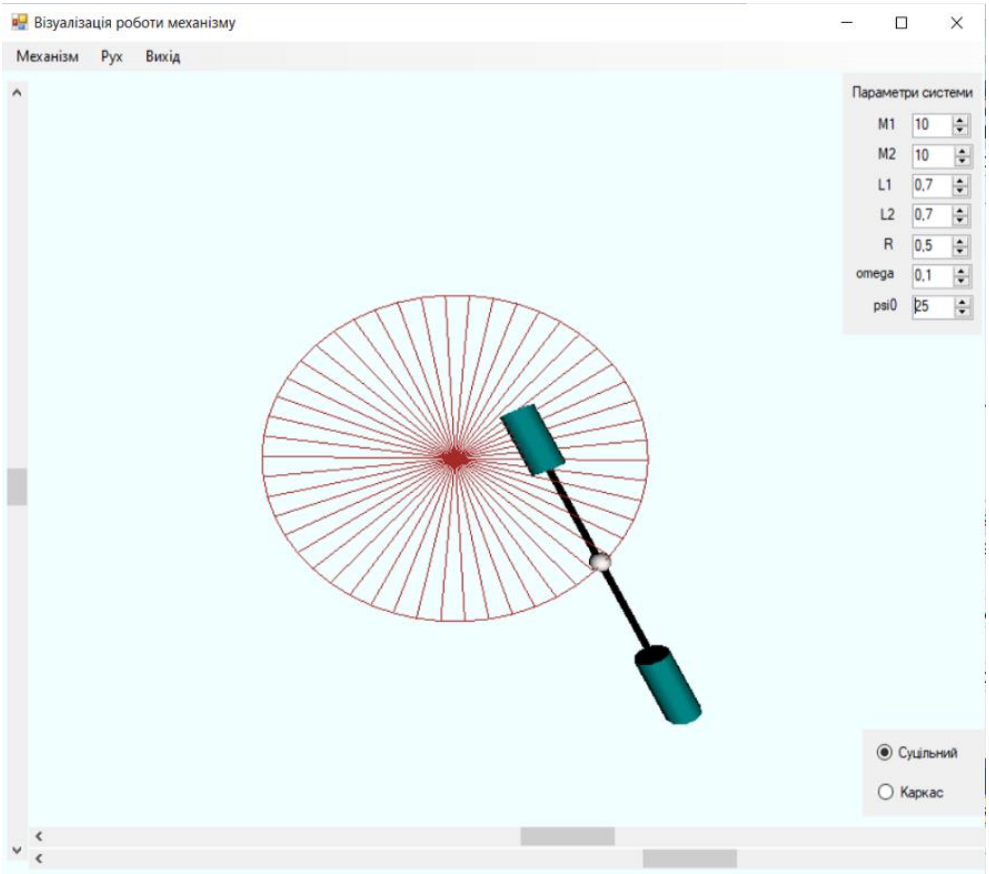
При русі механізму ніякі параметри змінювати не можна, так як вони стають недоступними.

В програмі існує ще низка методів, які є подієвими методами компонентів numericUpDown, у яких користувач змінює відповідне значення. В усіх цих методах відбувається зчитування відповідного значення, перерахування пов’язаних з ним параметрів і перемальовування механізму згідно з новим значенням даного параметра.

5 ДЕМОНСТРАЦІЯ РОБОТИ ДОДАТКУ

Робота програми була протестована на декількох варіантах розрахунку відносної рівноваги для різних значень параметрів механізмів.

1. Випадок однакових мас $m_1 = m_2 = 10$ кг і однакових довжин важелів $l_1 = 0{,}7$ м, $l_2 = 0{,}7$ м. Радіус диску R = 0,5 м, кутова швидкість обертання диску $\omega = 0{,}1$ с$^{-1}$ і початковий кут відхилення важеля $\psi = 25^0$.



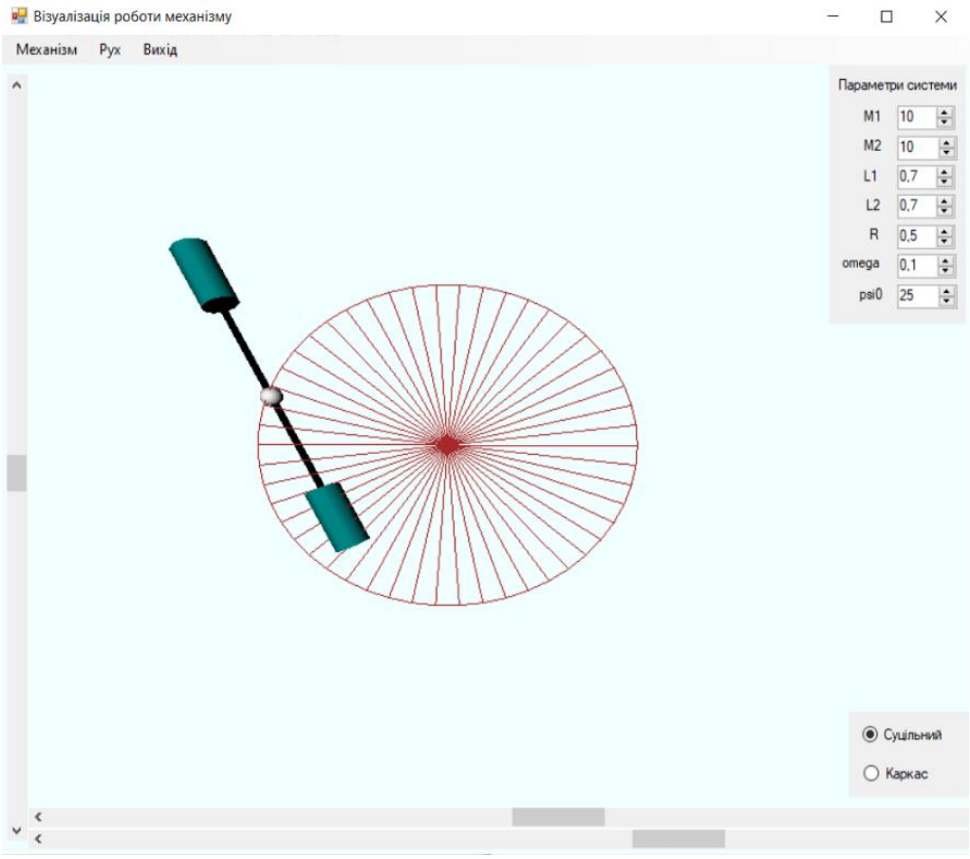


Рисунок 5.1 – Положення механізму в деякий момент часу для випадку 1

На рисунку зображені скриншоти двох положень механізму у різні моменти часу. Під час роботи програми та з цих скриншотів можна бачити, що важелі з вантажами лише обертаються разом із диском навколо осі, що проходить через центр диска перпендикулярно до площини рисунка, але не прокручуються навколо шарніру, за допомогою якого вони прикріплені до диску, тобто вони знаходяться у положенні відносної рівноваги, що підтверджується теоретично: $m_1 l_1 = m_2 l_2$.

2. Знову задамо випадок однакових мас і однакових довжин важелів, як і у попередньому випадку, але змінимо початковий кут відхилення важеля ψ – покладемо $\psi = 90^0$.

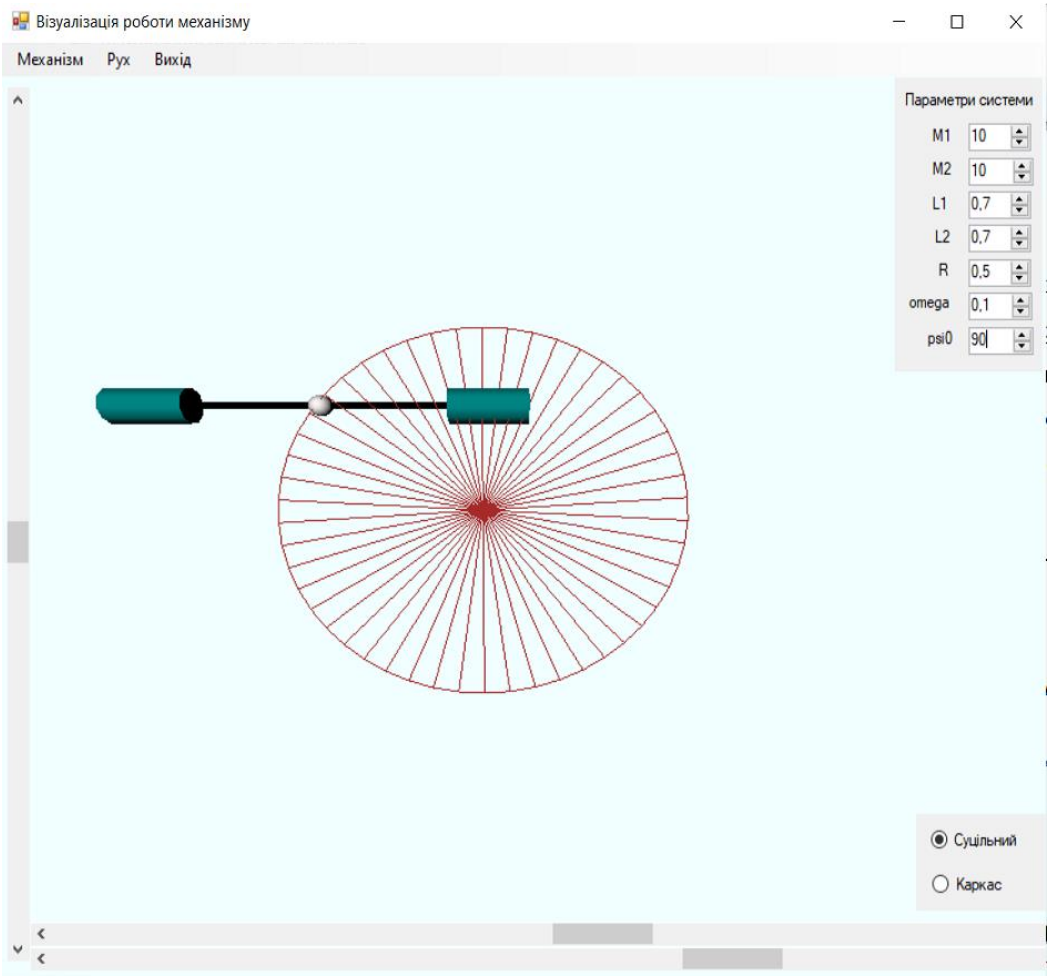


Рисунок 5.2 – Положення механізму в деякий момент часу для випадку 2

Горизонтальне положення важеля з вантажами і в цьому випадку залишається у весь час руху горизонтальним. Це теж випадок відносної рівноваги важелів.

3. На наступному скриншоті вибраний початковий кут відхилення важеля $\psi = 0^0$ при тих же параметрах всієї системи. І це теж випадок відносної рівноваги важелів, тобто у весь час руху важель з вантажами залишається у вертикальному положенні.

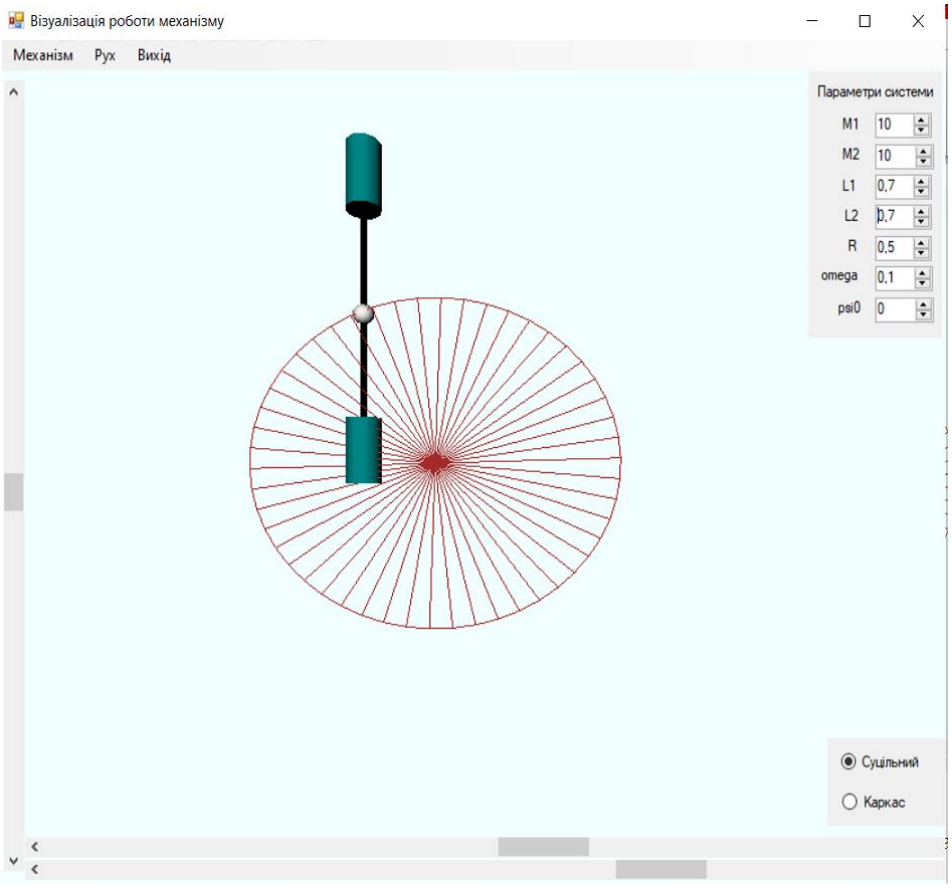


Рисунок 5.3 – Положення механізму в деякий момент часу для випадку 3

4. Спробуємо ще раз отримати випадок відносної рівноваги важелів, але з різними масами: маси m_1 = 5 кг, m_2 = 10 кг. Розрахуємо довжини важелів для виконання рівності $m_1 l_1 = m_2 l_2$: l_1 = 0,8 м, l_2 = 0,4 м. Виберемо R = 0,6 м, кутову швидкість обертання диску ω= 0,05 с-1 і початковий кут відхилення важеля ψ=30^0.

Отримали очікувані результати: зберігається початкове розташування важеля з вантажами під кутом 30^0 від вертикалі. Це знову ж таки випадок відносної рівноваги важелів, і це підтверджуються скриншотами на рисунку 5.4:

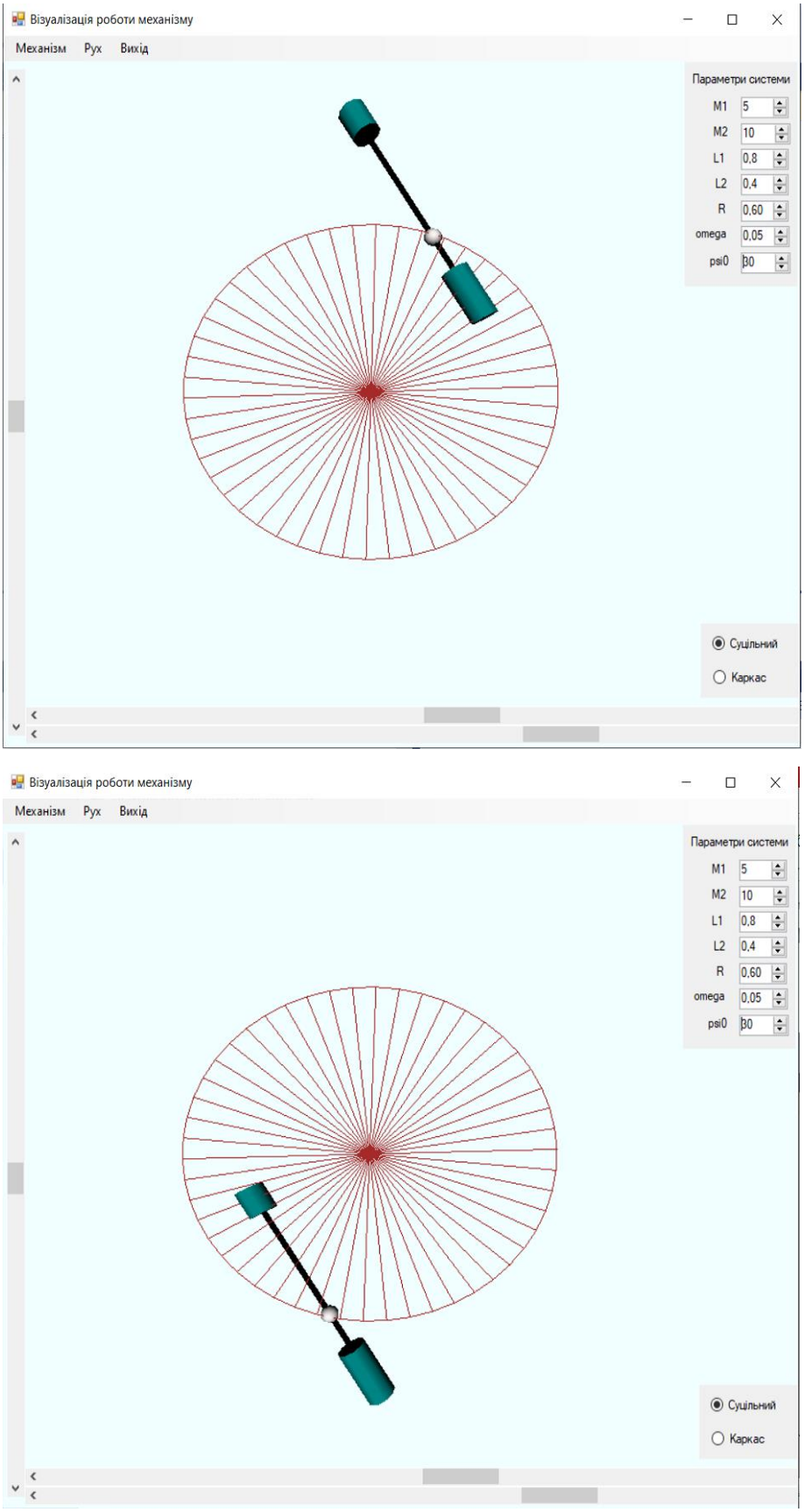
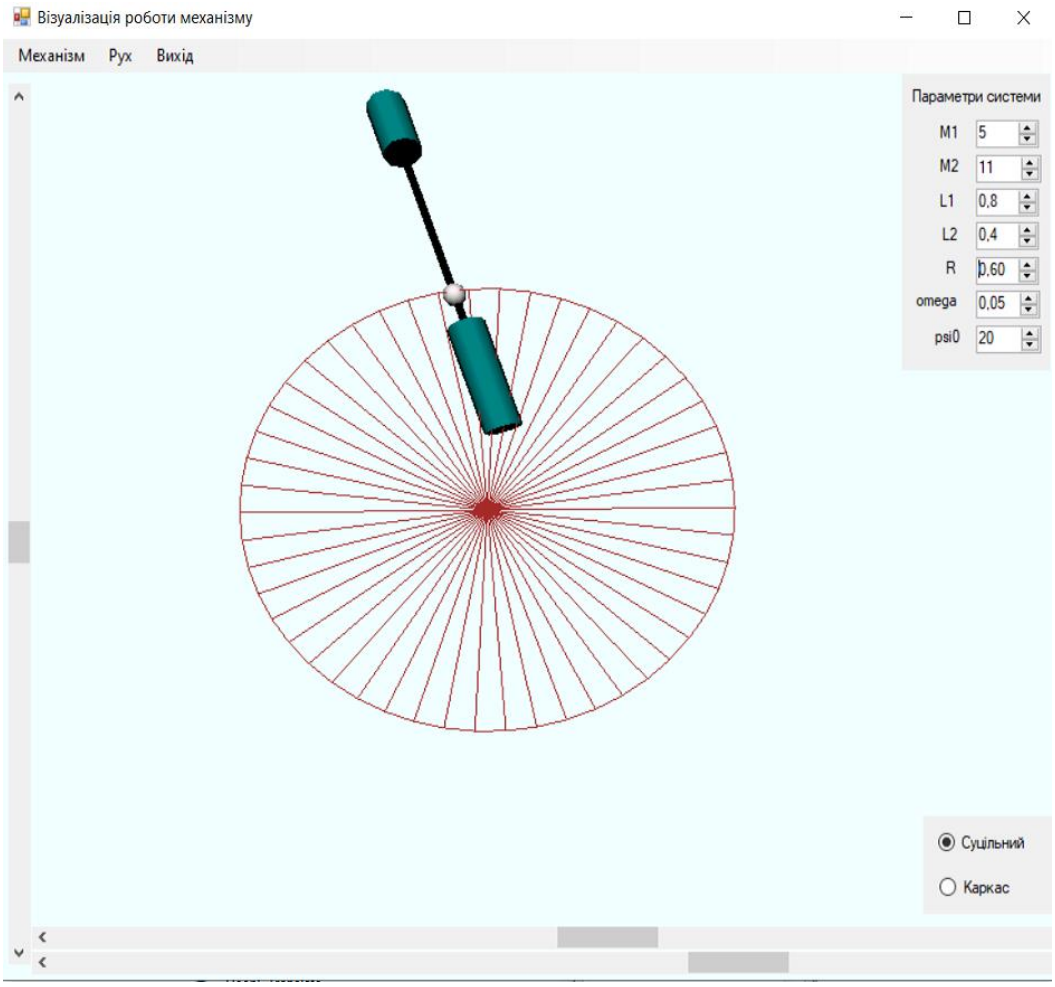


Рисунок 5.4 – Положення механізму в деякий момент часу для випадку 4

5. Трошки збільшимо масу вантажу m_2 в порівнянні з тим, яку він мав у положенні відносної рівноваги, і покладемо її рівною 11 кг. У цьому випадку отримаємо вже коливання важелів з невеличкою амплітудою, що демонструють скриншоти на рисунку 5.5.



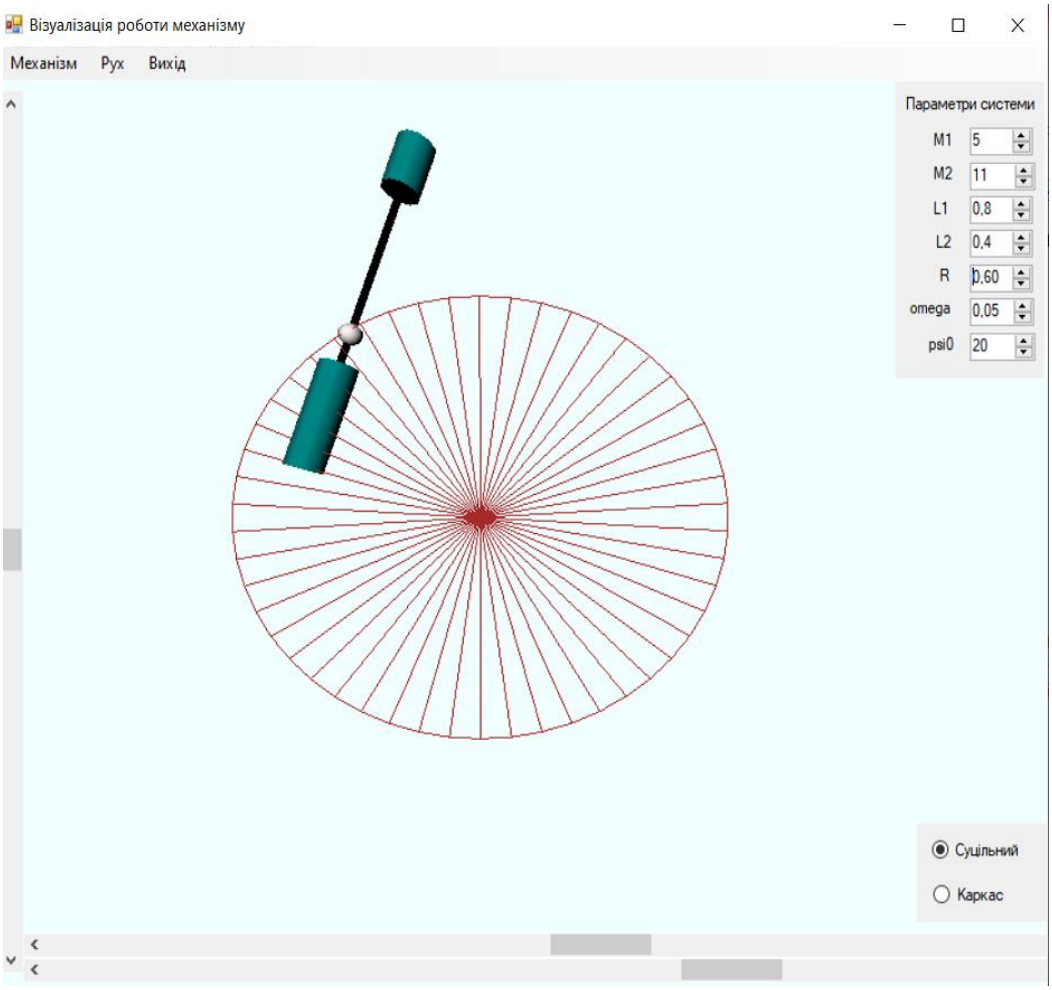


Рисунок 5.5 – Положення механізму в деякий момент часу для випадку 5

6. А тепер вантажі поміняємо місцями – зверху розмістимо той, що має важчу масу і маленьку довжину важеля, а знизу меншу масу і більшу довжину важеля, тобто: маси m_1 = 11 кг, m_2 = 5 кг, довжини l_1 = 0,4 м, l_2 = 0,8 м. Отримаємо теж випадок коливань навколо шарніру, але з набагато більшою амплітудою:

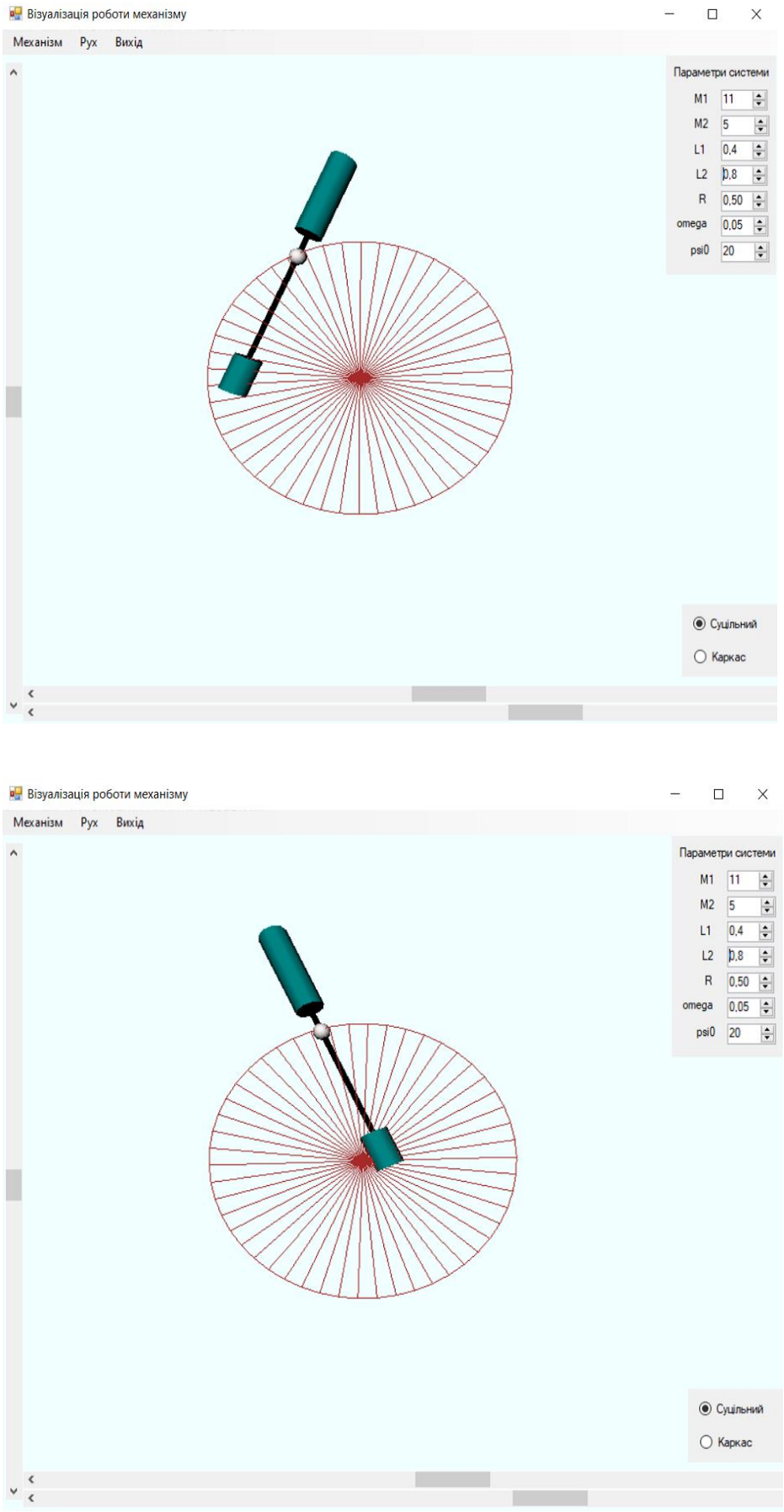
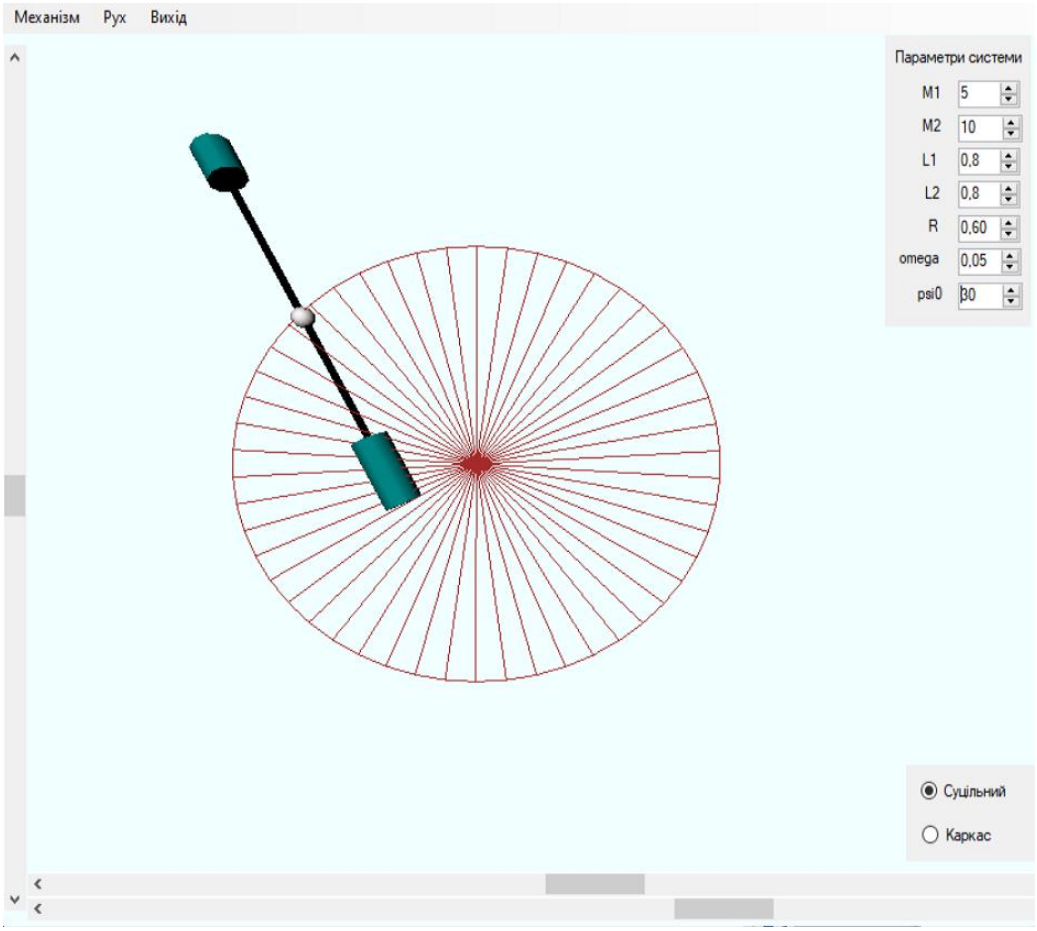


Рисунок 5.6 – Положення механізму в деякий момент часу для випадку 6

7. А тепер поекспериментуємо з довжинами важелів. Виберемо наступні параметри системи: маси m_1 = 5 кг, m_2 = 10 кг, довжини l_1 = 0,8 м, l_2 = 0,8 м, R = 0,6 м, кутова швидкість обертання диску ω= 0,05 с-1 і початковий кут відхилення важеля $\psi=30^0$.

У цьому варіанті маса нижнього вантажу перевищує за масу верхнього вантажу, довжини важелів однакові. Тому при візуалізації ми бачимо картину невеликих коливань стрижня з вантажами:



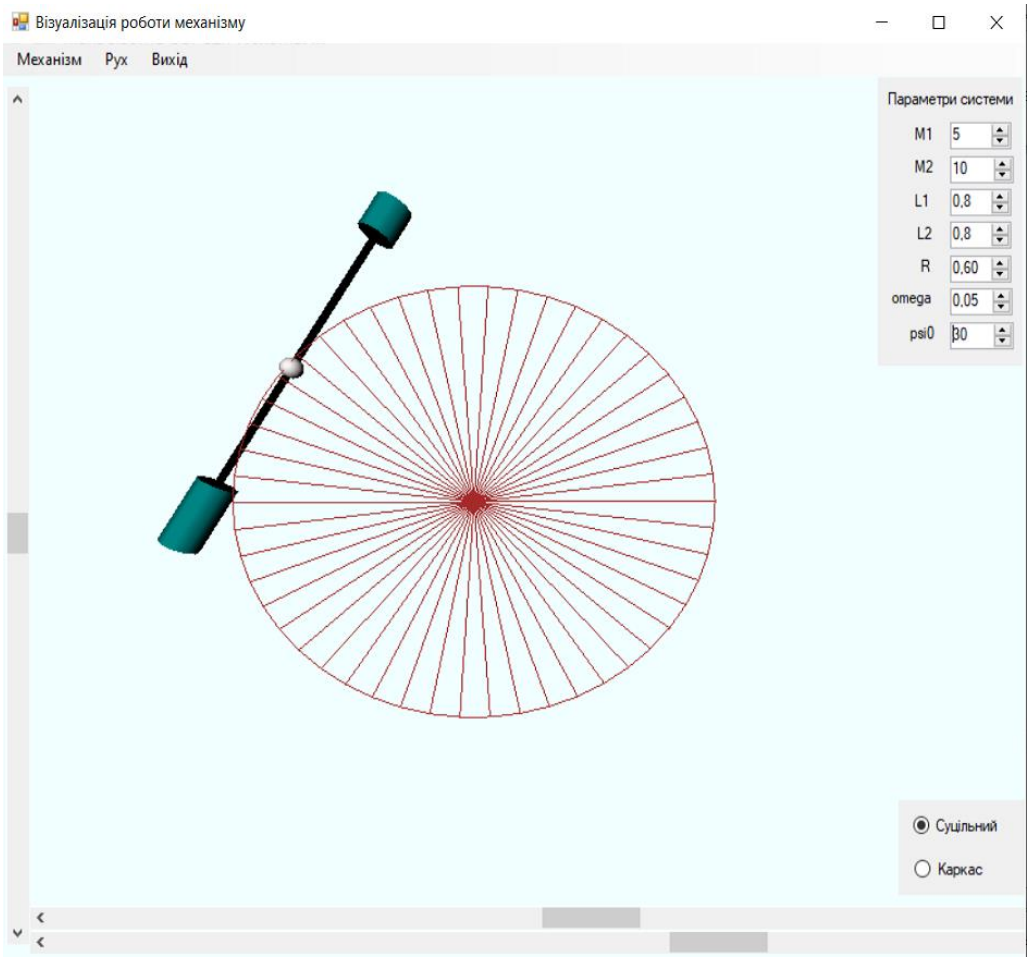


Рисунок 5.7 – Положення механізму в деякий момент часу для випадку 7

8. Поміняємо місцями верхній та нижній вантажі та виберемо наступні параметри системи: маси m_1 = 10 кг, m_2 = 5 кг, довжини l_1 = 1,0 м, l_2 = 0,7 м, R = 0,5 м, кутова швидкість обертання диску ω= 0,05 с-1 і початковий кут відхилення важеля $\psi=20^0$.

Отримаємо теж випадок коливань навколо шарніру, але знову ж таки з набагато більшою амплітудою:

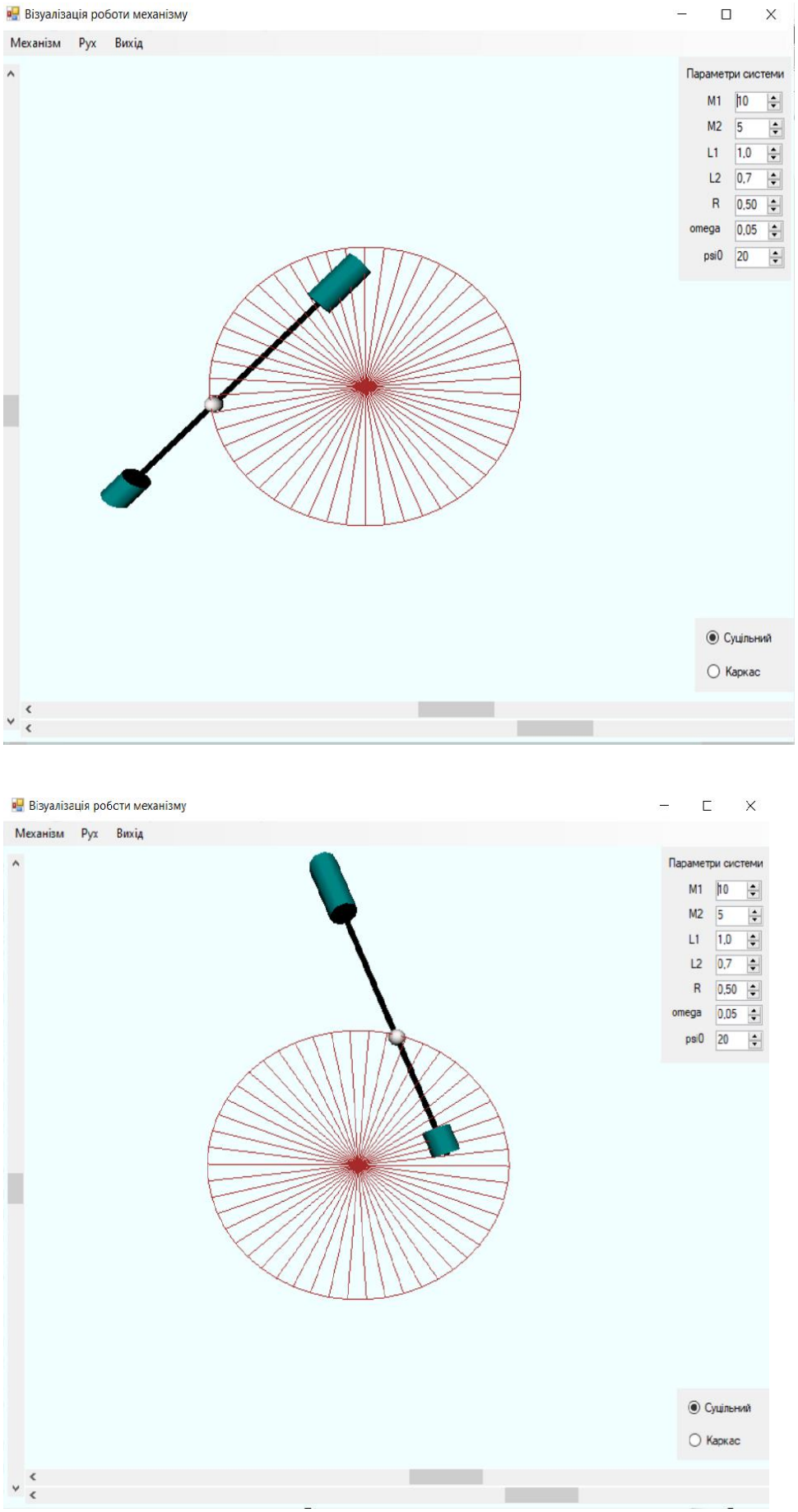


Рисунок 5.8 – Положення механізму в деякий момент часу для випадку 8

А тепер проглянемо візуалізацію цього випадку у каркасному вигляді:

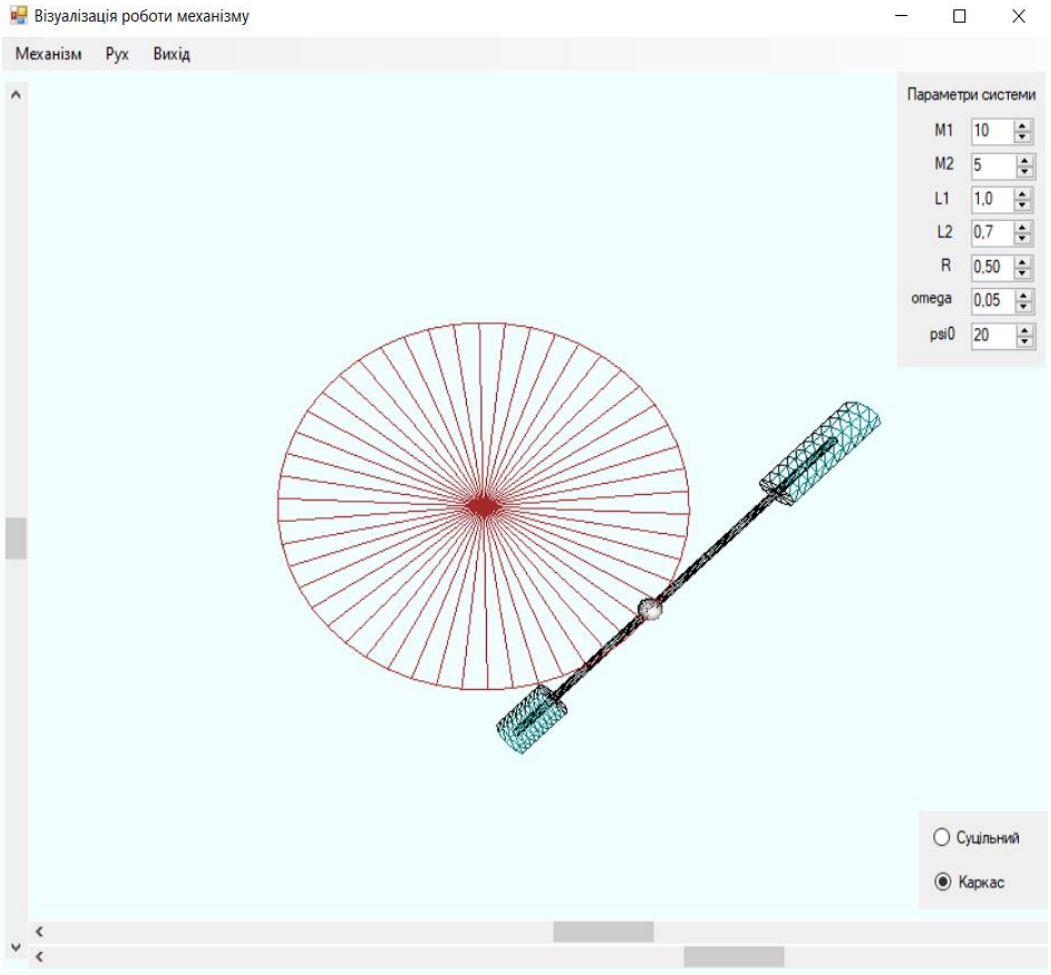


Рисунок 5.9 – Механізм у каркасному вигляді

А ось такий вигляд має механізм, якщо повернути площину перегляду навколо горизонтальної осі за допомогою вертикального елемента прокручування vScrollBar1.

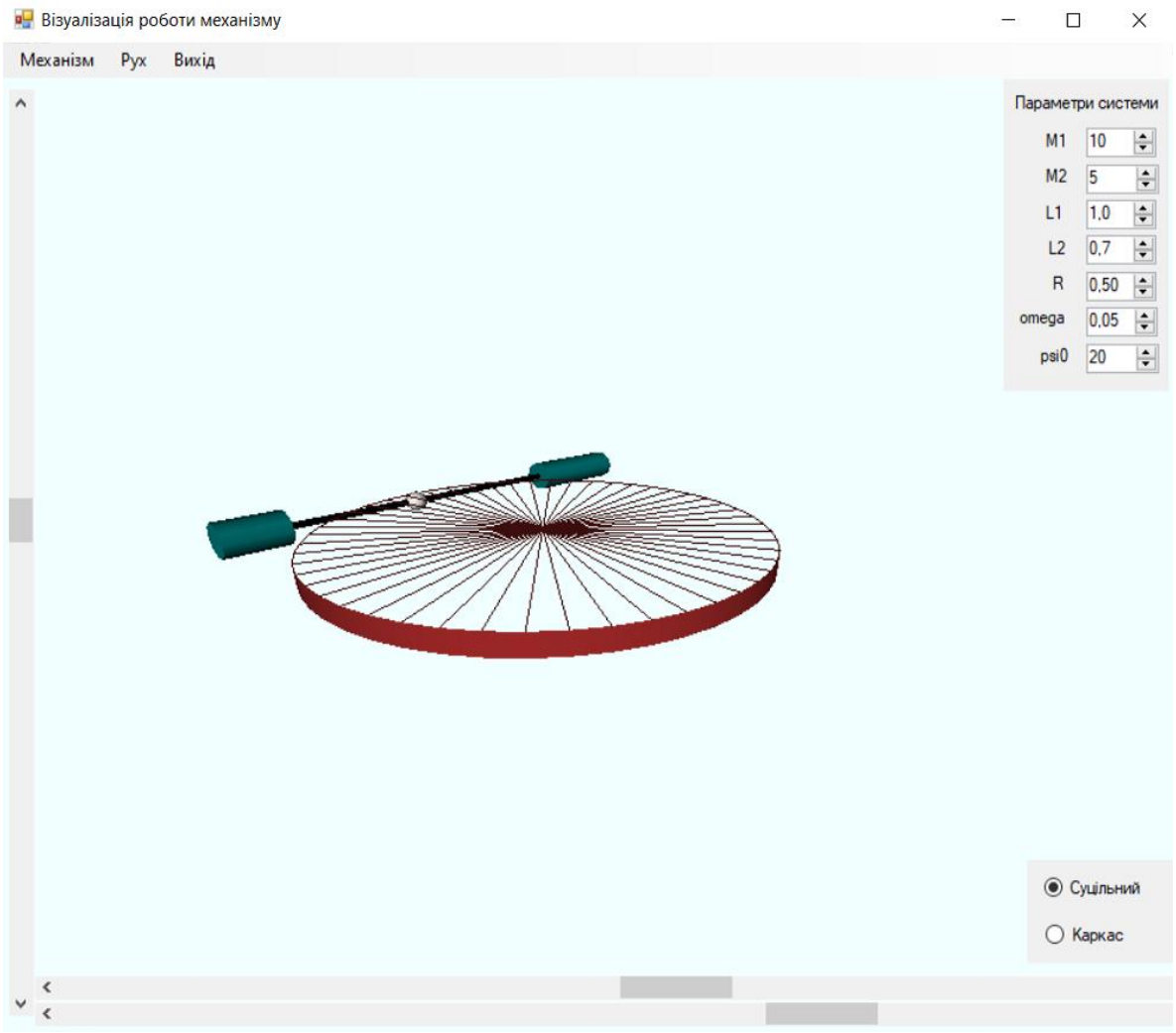


Рисунок 5.10 – Механізм, повернутий навколо горизонтальної осі

Наступні малюнки демонструють механізм, якщо площину перегляду повернути навколо вертикальної осі за допомогою її горизонтального елемента прокручування hScrollBar2.

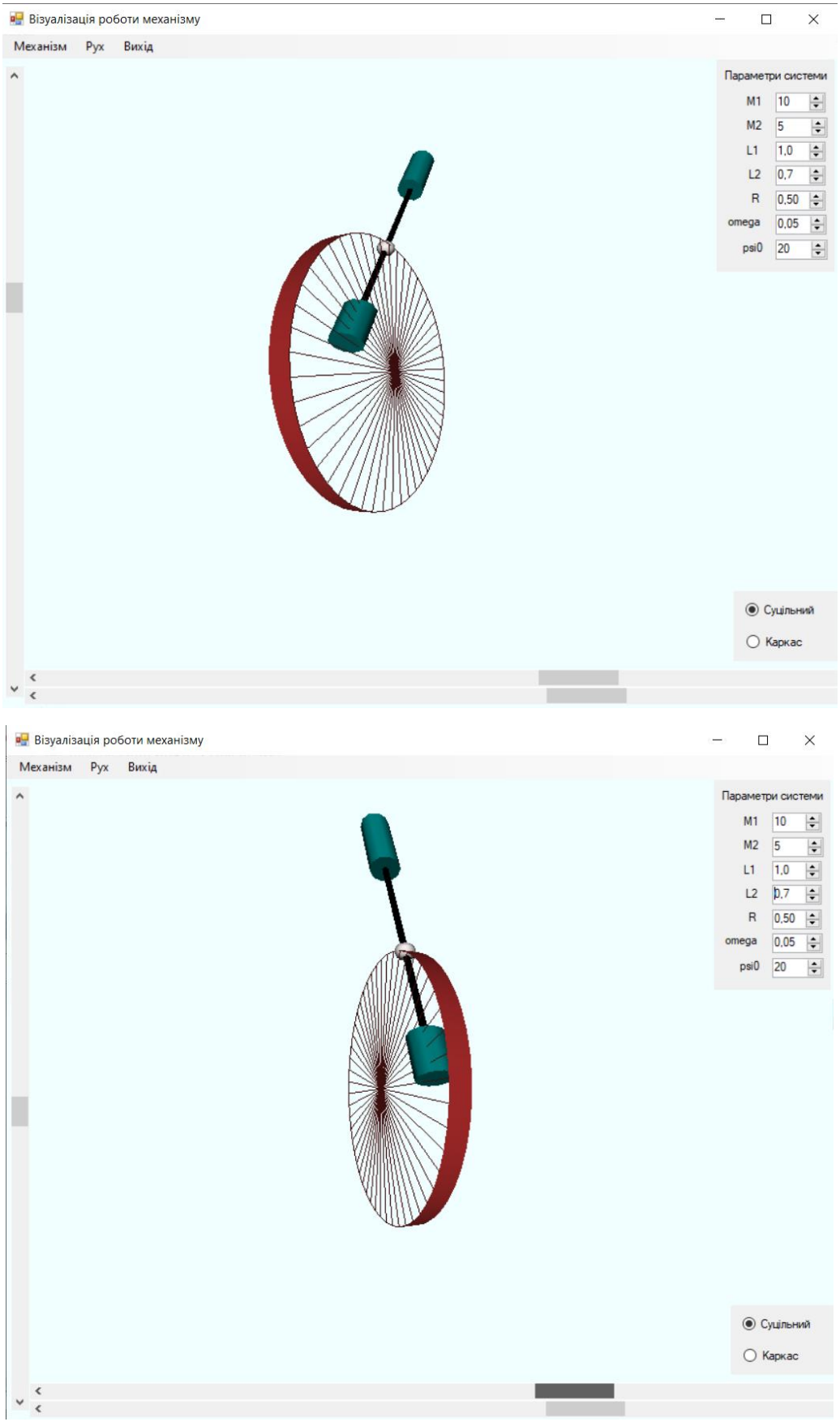


Рисунок 5.11 – Механізм, повернутий навколо вертикальної осі

А на наступному рисунку разом із зміною площини перегляду був зменшений масштаб за допомогою самої нижньої смуги прокрутки hScrollBar1.

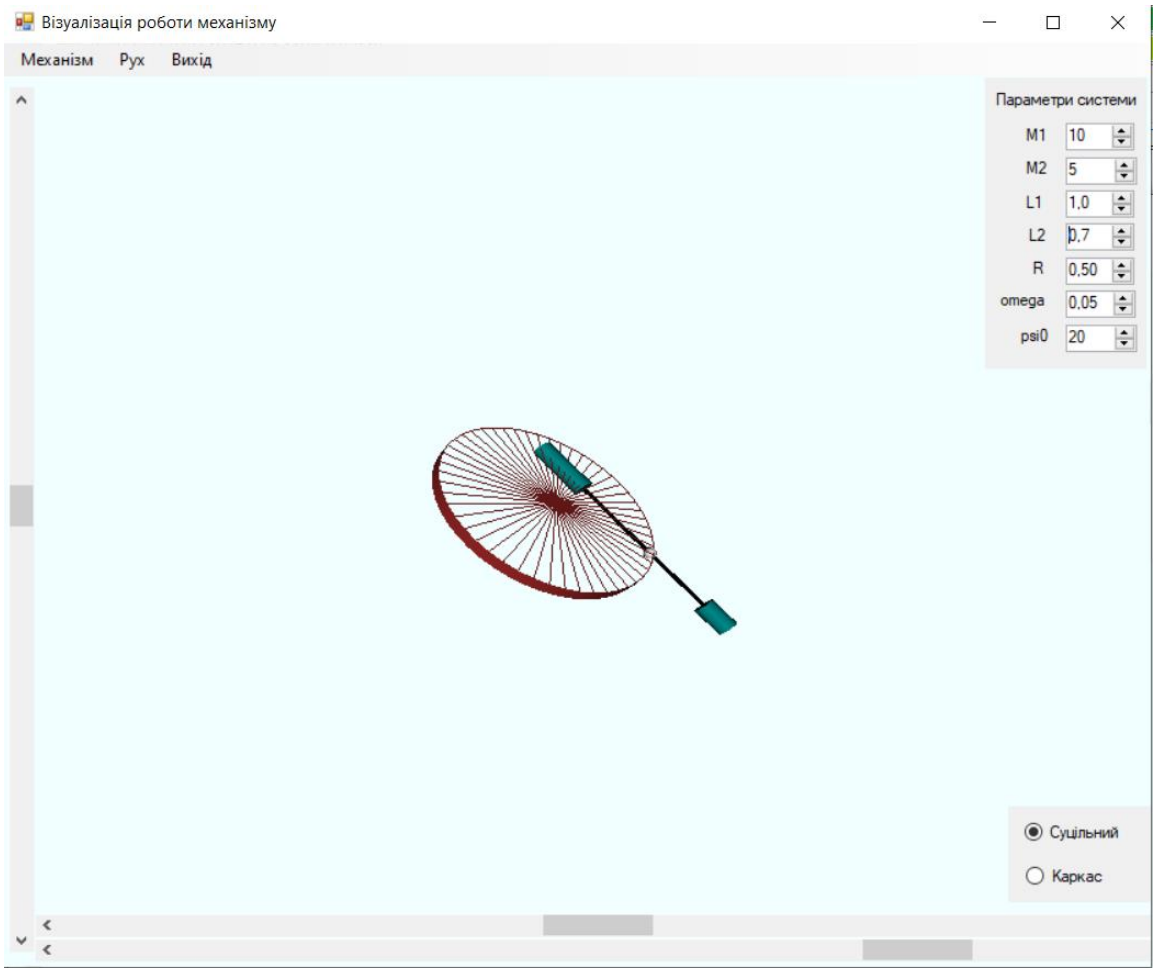


Рисунок 5.12 – Положення механізму в деякий момент часу у зменшеному масштабі

ВИСНОВКИ

Моя робота носить практичний характер. Мною розроблений Windows–додаток, що здійснює візуалізацію руху механізму з однією ступеню свободи. Цей додаток розроблений на мові C# в інтегрованому середовищі Microsoft Visual Studio із використанням 3D-графіки.

В роботі отримано диференціальне рівняння руху механізму за допомогою рівняння Лагранжа II роду. Для чисельного вирішення цього рівняння застосований метод Рунге-Кутта IV порядку точності.

Додаток надає можливість користувачеві самому задавати геометричні параметри механізму, його початкове положення та кутову швидкість переносного руху механізму. Користувач має можливість повертати площину перегляду руху механізму навколо вертикальної та горизонтальної площин. Закінчення перегляду візуалізації регулюється також користувачем.

Думаю, що моя програма може бути корисною як для розрахунків, так і для демонстрації роботи механізмів під час вивчення програмування графіки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бухгольц М.М. "Основний курс теоретичної механіки". Т. 1,2. - М.: Наука, 1972.
2. Кільчевський М.О. "Курс теоретичної механіки". Т. 1,2. - М.: Наука, 1977 .
3. Яблонський О.О. "Збірка завдань для курсових робіт по теоретичній механіці". - М.: Вища школа, 1985.
4. Бахвалов Н.С., Жидков Н.П., Кобельков Г.М. «Численные методы» - М., Наука, 1987. – 600с.
5. Tom Miller. Managed Directx 9. Graphics and Game Programming – Sams, 2003, 504p.
6. David F. Rogers, J. Alan Adams. Mathematical Elements for Computer Graphics. McGraw Hill Book Company, 1990, 604p.
7. Бублик В.В. Об’єктно-орієнтоване програмування – К.: ІТкнига, 2015, 624 с.
8. Andrew Troelsen. Pro C# 7. With .NET and .NET Core. Berkeley: Apress, 2017, 1372p.
9. Schildt H. C# 4.0. The Complete Reference. New York:McGraw Hill Education, 2017, 984 p.
10. Sharp J. Microsoft Visual C# Step by Step. London:Pearson Education, 2018, 878p.

ДОДАТОК А
ЛІСТИНГ КОДУ ПРОГРАМИ

```
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 Microsoft.DirectX.Direct3D;
using Microsoft.DirectX;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        Device d3d;
        Mesh wheel, rod, sphere, Gruz1, Gruz2;
        Material wheelMaterial, rodMaterial, sphereMaterial,
GruzMaterial;

        public static double  R, M_1, M_2, L_1, L_2;
        public static double omega = 0.1, time = 0.0, dt = 0.1;
        double fi_0, psi_0, dpsi_0 = 0, psi_1, dpsi_1, fi00 = 0;
        float r, l1, l2, koef = 1.5f, koefR = 5.0f, K=10; float
koef1=0.9f;
        public static float radius, Mrot, M, Jrot, Jwh, fi0,
fiSht0, mass, I, F;
        float dm1, dm2;
        float R1, R2;
        double mu;
        bool fl = false;
        bool dv = false;
        bool karkas = false; int dd = 1;
        public Form1()
        {
            InitializeComponent();
            d3d = null;
            wheel = null;
            rod = null; sphere = null;
            Gruz1 = null; Gruz2 = null;
```

```
             SetStyle(ControlStyles.Opaque, true);
SetStyle(ControlStyles.UserPaint, true);
            SetStyle(ControlStyles.AllPaintingInWmPaint, true);
            this.SetStyle(ControlStyles.AllPaintingInWmPaint |
ControlStyles.Opaque, true);//???

        }
        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                // режим отображения трехмерной графики
                PresentParameters d3dpp = new
PresentParameters();
                d3dpp.BackBufferCount = 1;
                d3dpp.SwapEffect = SwapEffect.Discard;
                d3dpp.Windowed = true; // Выводим графику в окно
                d3dpp.MultiSample = MultiSampleType.None; //
                d3dpp.EnableAutoDepthStencil = true; //
                d3dpp.AutoDepthStencilFormat = DepthFormat.D16;
                d3d = new Device(0, // D3D_ADAPTER_DEFAULT
                DeviceType.Hardware, // Тип устройства -
                this, // Окно для вывода графики
                CreateFlags.SoftwareVertexProcessing,
d3dpp);
            }
            catch (Exception exc)
            {
                MessageBox.Show(this, exc.Message, "Ошибка
инициализации");
                Close();
            }
            ChangeData();
            //КОЛЕСО
            wheel = Mesh.Cylinder(d3d, r * koefR, r * koefR,
0.5f, 50, 50);//50 50
            wheelMaterial = new Material();
            wheelMaterial.Diffuse = Color.Brown;
            wheelMaterial.Specular = Color.White;
            // Стержень
            rod = Mesh.Cylinder(d3d, 0.1f, 0.1f, l1 + l2, 10,
10);
            rodMaterial = new Material();
            rodMaterial.Diffuse = Color.Black;// Brown;
            rodMaterial.Specular = Color.White;
```

```
            // Шарнир
            sphere = Mesh.Sphere(d3d, 0.3f, 10, 10);
            sphereMaterial = new Material();
            sphereMaterial.Diffuse =
Color.Snow;//.Gray;//.SlateGray;//.Red;// Black;
            sphereMaterial.Specular = Color.White;
            ChangeData();
            // Вантажі
            mu = M_1 / M_2;
            if (mu >= 1) { R1 = 3; R2 = R1 / (float)mu; }
            else { R2 = 3; R1 = R2 * (float)mu; }
            Gruz1 = Mesh.Cylinder(d3d, 0.5f, 0.5f, R1, 10, 10);
            Gruz2 = Mesh.Cylinder(d3d, 0.5f, 0.5f, R2, 10, 10);
            GruzMaterial = new Material();
            GruzMaterial.Diffuse = Color.DarkCyan;//
            GruzMaterial.Specular = Color.White;
        }
        public void OnIdle(object sender, EventArgs e)
        {
            Invalidate();
        }

        private void ChangeData()
        {
            M_1 = (double)numericUpDown1.Value;
            M_2 = (double)numericUpDown2.Value;
            L_1 = (double)numericUpDown3.Value;
            L_2 = (double)numericUpDown4.Value;
            R = (double)numericUpDown5.Value;
            r = (float)R/ koefR*K;
            l1 = (float)L_1  / koef*K;
            l2 = (float)L_2/ koef*K;
            float dm1 = (float)(numericUpDown1.Maximum -
numericUpDown1.Minimum) / 10;
            float dm2 = (float)(numericUpDown2.Maximum -
numericUpDown2.Minimum) / 10;
            // R1 =  (float)M_1 / dm1 / 5; R2 = (float)M_2 / dm2
/ 5;
            mu = M_1 / M_2;  //!!!!!!!
            if (mu >= 1) { R1 = 3f; R2 = R1 / (float)mu; }
            else { R2 = 3f; R1 = R2 * (float)mu; }
            omega = Convert.ToDouble(numericUpDown6.Value);
            psi_0 = Convert.ToDouble(numericUpDown7.Value) *
Math.PI / 180;
            fi_0 = 0;
```

```
            this.Invalidate();
        }
            private void SetupProekcii()
            {
            d3d.RenderState.Lighting = true;
            d3d.Transform.Projection =
Matrix.PerspectiveFovLH((float)Math.PI / 2, (this.Width) /
this.Height, 1.0f, 50.0f);
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);
}

        public void Creation()
        {
            wheel.Dispose();
            rod.Dispose();
            sphere.Dispose();
            Gruz1.Dispose();
            Gruz2.Dispose();
            wheel = Mesh.Cylinder(d3d, r*koefR, r*koefR, 0.5f,
50, 50);
            rod = Mesh.Cylinder(d3d, 0.1f, 0.1f, l1 + l2, 10,
10);
            sphere = Mesh.Sphere(d3d, 0.3f, 10, 10);
            if (mu >= 1) { R1 = 3; R2 = R1 / (float)mu; }
            else { R2 = 3; R1 = R2 * (float)mu; }
            Gruz1 = Mesh.Cylinder(d3d, 0.5f, 0.5f, R1, 10, 10);
            Gruz2 = Mesh.Cylinder(d3d, 0.5f, 0.5f, R2, 10, 10);
        }
        private void Draw(float R, float L1, float L2, float
fi1, float fi2, float R1, float R2)
        {
            R = R * koefR; L1 *= koef1; L2 *= koef1;
             R1 *= koef1; R2 *= koef1;

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

```
            //  Matrix.RotationY((float)hScrollBar2.Value /
10.0f) * Matrix.RotationX((float)vScrollBar1.Value / 10.0f) *
Matrix.Translation(0, 0, (float)hScrollBar1.Value);
            if (fl == true)
            {
                // Колесо
                d3d.RenderState.FillMode = FillMode.WireFrame;
                d3d.Material = wheelMaterial;
                d3d.Transform.World = Matrix.RotationZ(fi1) *
Matrix.RotationY((float)hScrollBar2.Value / 10.0f) *
Matrix.RotationX(-(float)vScrollBar1.Value / 10.0f) *
Matrix.Translation(-dd, 0, (float)hScrollBar1.Value);
                wheel.DrawSubset(0);
                if (karkas)
                    d3d.RenderState.FillMode =
FillMode.WireFrame;
                else
                    d3d.RenderState.FillMode = FillMode.Solid;
                // Стержень
               d3d.Material = rodMaterial;
                // d3d.Transform.World =
Matrix.RotationY((float)Math.PI / 2.0f)  * Matrix.Translation(-
(L1 + L2) / 2.0f, 0, 0) * Matrix.Translation(L2, 0, 0) *
Matrix.Translation(0, R, 0) *
Matrix.RotationY((float)hScrollBar2.Value / 10.0f) *
Matrix.RotationX(-(float)vScrollBar1.Value / 10.0f) *
Matrix.Translation(0, 0, (float)hScrollBar1.Value);
                d3d.Transform.World =
Matrix.RotationY((float)Math.PI / 2.0f)
                   * Matrix.Translation(-(L1 + L2) / 2.0f, 0, 0)
                   * Matrix.Translation(L2, 0, 0)
                   * Matrix.RotationZ(-(float)Math.PI / 2.0f +
fi2)
                   * Matrix.Translation(R *
(float)Math.Cos(fi1), R * (float)Math.Sin(fi1), 0)
                   * Matrix.RotationY((float)hScrollBar2.Value /
10.0f) * Matrix.RotationX(-(float)vScrollBar1.Value / 10.0f)
                   * Matrix.Translation(-dd, 0,
(float)hScrollBar1.Value);
                rod.DrawSubset(0);
                // Шарнир
                d3d.Material = sphereMaterial;
                // d3d.Transform.World =  Matrix.Translation(0,
R, 0) * Matrix.RotationY((float)hScrollBar2.Value / 10.0f) *
```

```
Matrix.RotationX(-(float)vScrollBar1.Value / 10.0f) *
Matrix.Translation(0, 0, (float)hScrollBar1.Value);
                d3d.Transform.World = Matrix.Translation(R *
(float)Math.Cos(fi1), R * (float)Math.Sin(fi1), 0)
                    * Matrix.RotationY((float)hScrollBar2.Value
/ 10.0f)
                    * Matrix.RotationX(-(float)vScrollBar1.Value
/ 10.0f)
                    * Matrix.Translation(-dd, 0,
(float)hScrollBar1.Value);
                sphere.DrawSubset(0);
                // Вантажі
                d3d.Material = GruzMaterial;
                d3d.Transform.World =
Matrix.RotationY((float)Math.PI / 2.0f)
                // * Matrix.Translation(-L1, R, 0)
                * Matrix.RotationZ(-(float)Math.PI / 2.0f + fi2)
                    * Matrix.Translation(R *
(float)Math.Cos(fi1) - L1 * (float)Math.Sin(fi2), R *
(float)Math.Sin(fi1) + L1 * (float)Math.Cos(fi2), 0)

                    * Matrix.RotationY((float)hScrollBar2.Value
/ 10.0f)
                    * Matrix.RotationX(-(float)vScrollBar1.Value
/ 10.0f)
                    * Matrix.Translation(-dd, 0,
(float)hScrollBar1.Value);
                Gruz1.DrawSubset(0);
                d3d.Transform.World =
Matrix.RotationY((float)Math.PI / 2.0f)
                   // * Matrix.Translation(L2, R, 0)
                   * Matrix.RotationZ(-(float)Math.PI / 2.0f +
fi2)
                    * Matrix.Translation(R *
(float)Math.Cos(fi1) + L2 * (float)Math.Sin(fi2), R *
(float)Math.Sin(fi1) - L2 * (float)Math.Cos(fi2), 0)
                    * Matrix.RotationY((float)hScrollBar2.Value
/ 10.0f)
                    * Matrix.RotationX(-(float)vScrollBar1.Value
/ 10.0f)
                    * Matrix.Translation(-dd, 0,
(float)hScrollBar1.Value);
                Gruz2.DrawSubset(0);
            }
              d3d.EndScene();
```

```
            d3d.Present();
        }
        private void Form1_Paint(object sender, PaintEventArgs
e)
        {
                Draw(r, l1, l2, (float)fi_0,
(float)psi_0,R1,R2);
        }

        private void намалюватиToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            time = 0; fi_0 = 0; fl = true;
            ChangeData();
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0,R1,R2);
            стертиToolStripMenuItem.Enabled = true;
            намалюватиToolStripMenuItem.Enabled= false;

            зупинитиToolStripMenuItem.Enabled = false;
            початиToolStripMenuItem.Enabled = true;
        }
        private void стертиToolStripMenuItem_Click(object
sender, EventArgs e)
        {
           fl = false;
             намалюватиToolStripMenuItem.Enabled = true;
             стертиToolStripMenuItem.Enabled = false;
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1,
R2);//
            зупинитиToolStripMenuItem.Enabled = false;
            початиToolStripMenuItem.Enabled = false;
            vScrollBar1.Value = 0;
            hScrollBar2.Value = 0;
            hScrollBar1.Value = 13;
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            rungekuttIV(dt, psi_0, dpsi_0, ref psi_1, ref
dpsi_1);
            psi_0 = psi_1;
            dpsi_0 = dpsi_1;
            time += dt;
            // стертиToolStripMenuItem.Enabled = true;
            fi_0 = fi00 + omega * time;
```

```
            Draw(r, l1, l2, (float)fi_0, (float)psi_0,R1,R2);//
            fl = true;
            this.Invalidate();
        }
        private void початиToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            timer1.Enabled = true;
            зупинитиToolStripMenuItem.Enabled = true;
            початиToolStripMenuItem.Enabled = false;
            стертиToolStripMenuItem.Enabled = false;
        }


        private void зупинитиToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            timer1.Enabled = false;
            зупинитиToolStripMenuItem.Enabled = false;
            початиToolStripMenuItem.Enabled =true;
            стертиToolStripMenuItem.Enabled = true;
        }
        private void vScrollBar1_Scroll(object sender,
ScrollEventArgs e)
        {
            this.Invalidate();
        }
        public static void rungekuttIV(double h, double y0,
double y10, ref double y1, ref double y11)
        {
            double k1 = h * func(y0);
            double k2 = h * func(y0 + h / 2 * y10 + h / 8 * k1);
            double k3 = h * func(y0 + h * y10 + h / 2 * k2);
            y11 = y10 + ((k1 + 4 * k2 + k3) / 6);
            y1 = y0 + h * (y10 + (k1 + 2 * k2) / 6);
        }
        private static double func(double psi)
        {
            return (R * omega * omega * (M_2 * L_2 - M_1 * L_1)
* Math.Cos(psi - omega * time)
                - (M_2 * L_2 - M_1 * L_1) * 9.81 *
Math.Sin(psi))
                / (M_1 * L_1 * L_1 + M_2 * L_2 * L_2);
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
        }
        private void numericUpDown6_ValueChanged(object sender,
EventArgs e)
        {
            omega = Convert.ToDouble(numericUpDown6.Value);
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1, R2);
        }

        private void numericUpDown7_ValueChanged(object sender,
EventArgs e)
        {
            psi_0 = Convert.ToDouble(numericUpDown7.Value) *
Math.PI / 180;
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1, R2);
        }

        private void numericUpDown5_ValueChanged(object sender,
EventArgs e)
        {
            R = (double)numericUpDown5.Value;
            r = (float)R/ koefR*K;
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1, R2);
        }

        private void numericUpDown4_ValueChanged(object sender,
EventArgs e)
        {
            L_2 = (double)numericUpDown4.Value;
            l2 = (float)L_2 / koef*K;
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1, R2);
        }

        private void numericUpDown3_ValueChanged(object sender,
EventArgs e)
        {
            L_1 = (double)numericUpDown3.Value;
            l1 = (float)L_1/ koef*K;
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1, R2);
```

```
        }

        private void numericUpDown2_ValueChanged(object sender,
EventArgs e)
        {
            M_2 = (double)numericUpDown2.Value;
            mu = M_1 / M_2;
            if (mu >= 1) { R1 = 3f; R2 = R1 / (float)mu; }
             else { R2 = 3f; R1 = R2 * (float)mu; }
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1, R2);
        }

        private void numericUpDown1_ValueChanged(object sender,
EventArgs e)
        {
            M_1 = (double)numericUpDown1.Value;
            mu = M_1 / M_2;
            if (mu >= 1) { R1 = 3f; R2 = R1 / (float)mu; }
            else { R2 = 3f; R1 = R2 * (float)mu; }
            Creation();
            Draw(r, l1, l2, (float)fi_0, (float)psi_0, R1, R2);
        }
        private void radioButton1_CheckedChanged(object sender,
EventArgs e)
        {
            karkas = false;
        }

        private void radioButton2_CheckedChanged(object sender,
EventArgs e)
        {
            karkas = true;
        }
        private void вихідToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Close();
        }
        private void hScrollBar1_Scroll(object sender,
ScrollEventArgs e)
        {
            this.Invalidate();
        }
```

```
        private void hScrollBar2_Scroll(object sender,
ScrollEventArgs e)
        {
            this.Invalidate();
        }
    }
}
```