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»

«Наближене обчислення центрів ваги та моментів інерції для неоднорідних тіл складної форми**»**

(тема кваліфікаційної роботи українською мовою)

«Approximate calculation of centers of gravity and moments of inertia for heterogeneous bodies of complex shape**»**

(тема кваліфікаційної роботи англійською мовою)

Виконав(ла): здобувач(ка) денної/заочної форми навчання
спеціальності 126 – Інформаційні системи та технології
(код, назва спеціальності)
Освітня програма Інформаційні системи та технології
(назва)
Зіменко Вадим Анатолійович
(прізвище, ім'я, по-батькові здобувача)

Керівник старший викладач Царенко О.П. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

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

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

Завідувач(ка) кафедри
_____ Алла РАЧИНСЬКА
(підпис)

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

Оцінка_____/___ ___/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ Володимир ВИЧУЖАНІН
(підпис)

Одеса 2024

АНОТАЦІЯ

Цей проект присвячений розробці програми для дослідження можливостей обробки зображень фігур та обчислення їхніх характеристик, таких як статичні моменти та моменти інерції. Основною метою є визначення центрів мас та моментів інерції для плоских фігур довільної форми, які можуть бути задані як аналітично, так і за допомогою графічних файлів. Програма дозволяє інтерактивно працювати з користувачем, завантажувати та обробляти зображення, додавати декартову систему координат на зображення та здійснювати обчислення у реальному часі. Реалізація проекту підтверджує ефективність застосування сучасних обчислювальних методів та інструментів програмування для вирішення задач механіки та геометрії, і може бути використана як навчальний матеріал у курсах об'єктно-орієнтованого програмування та чисельних методів.

This project focuses on developing a program to explore the capabilities of image processing for figures and calculating their characteristics, such as static moments and moments of inertia. The primary goal is to determine the centers of mass and moments of inertia for plane figures of arbitrary shapes, which can be defined analytically or through graphic files. The program allows interactive user engagement, enabling image uploading and processing, adding Cartesian coordinate systems to images, and performing real-time calculations. The project's implementation confirms the effectiveness of modern computational methods and programming tools for solving problems in mechanics and geometry, and it can be used as educational material in courses on object-oriented programming and numerical methods.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

АТТ - Абсолютно тверде тіло

ПФ – Плоска фігура

СМТ – Система матеріальних точок

ВСТУП

Математична постановка задачі про рух абсолютно твердого тіла (АТТ) під дією прикладених до нього зовнішніх сил та моментів по відношенню до заданої інерційної системи відліку потребує завдання моментів інерції цього тіла, обчислених по відношенню до відповідних координатних осей.

Оскільки задане тверде тіло може бути, по-перше, складної просторової форми, та, по-друге, може бути неоднорідним за розподілом маси тіла за його об’ємом, стає окремим попереднім завдання про обчислення його статичних моментів (центр мас) та моментів інерції (радіусів інерції).

Загальна задача визначення вказаних характеристик для 3D абсолютно твердого тіла може бути дуже складною як з математичної точки зору, так і з точки зору певних можливостей в її програмній реалізації.

Тому в рамках даної дипломної роботи завдання було обмежене розглядом моментів інерції та інших параметрів АТТ виключно для плоских суцільних фігур довільної форми із довільним законом розподілення маси фігури по її точках по всій площині фігури.

Передбачається, що задана плоска фігура (ПФ) будується як певний екземпляр відповідного класу, якій в якості фактичного параметра свого конструктора отримує або зображення фігури, або фігура задається аналітичним шляхом (для фігур канонічної геометричної форми).

Для з’ясування самої можливості такого підходу до розв’язування зазначеної проблеми треба створити абстрактний клас для плоских фігур, які потім можна задавати аналітичним способом, та для яких нам вже відомі формули обчислень статичних моментів та моментів інерції по відношенню до заданих систем відліку.

Це дозволить, принаймні для однорідних ПФ канонічної геометричної форми, перевіряти результати наближеного обчислення статичних моментів та моментів інерції за спеціально розробленим чисельним алгоритмом.

Для реалізації обчислень вказаних характеристик ПФ слід розробити та налагодити алгоритм наближеного обчислення подвійних інтегралів заданого функціонального виду.

Передбачається, що ця чисельна процедура повинна бути ітераційною, та отримувати шуканий результат із заданою абсолютною похибкою.

Задля перевірки математичної коректності та збіжності зазначеної чисельної процедури слід реалізувати низку тестових обчислень із використанням відомих подвійних інтегралів із різними підінтегральними функціями та різними границями інтегрування.

Всі розроблені алгоритми та типи даних повинні бути оформленими у вигляді класів та їх методів. Кожен з класів повинен бути описаним задля можливого подальшого використання в інших дослідницьких завданнях.

Крім того, успіх в програмній реалізації обчислень статичних моментів та моментів інерції для геометричних ПФ канонічної форми, підкаже – як саме можна здійснити ті ж обчислення для фігур, форма яких отримана із зображень заданого формату (мається на увазі – із графічних файлів).

В якості тестування такого підходу можна починати саме з реальних графічних зображень ПФ, які можна задавати також аналітично (круг, тор, прямокутник, еліпс, рівнобічний трикутник, прямокутний трикутник тощо).

Для тестування підходу потрібно створити програму для дослідження можливостей обробки зображень фігур та обчислення їхніх характеристик, таких як статичні моменти та моменти інерції. Зокрема, вона повинна перевіряти можливість визначення центрів мас та моментів інерції для плоских фігур довільної форми, які можуть бути задані як аналітично, так і за допомогою графічних файлів. Це має розширити область застосування розроблених методик та алгоритмів на більш широкий спектр задач, включаючи аналіз форми, отриманих з реальних зображень.

Окрім цього, програма повинна бути здатна інтерактивно працювати з користувачем, дозволяючи завантажувати та обробляти зображення, додавати декартову систему координат на зображення, а також здійснювати обчислення у реальному часі. Це дозволить користувачам ефективно визначати необхідні характеристики фігур, не вдаючись до складних математичних обчислень вручну.

Такий підхід також передбачає створення гнучкої архітектури програмного забезпечення, яка дозволить легко додавати нові функції та адаптувати існуючі методи для різних типів плоских фігур. Це стане важливим кроком у напрямку розробки універсальних інструментів для аналізу геометричних форм у різних галузях науки та техніки.

Крім того, реалізація такого проекту дозволить на практиці перевірити ефективність застосування сучасних обчислювальних методів та інструментів програмування для вирішення задач механіки та геометрії. Успішна розробка програми стане основою для подальших досліджень та може бути використана як навчальний матеріал у курсах об'єктно-орієнтованого програмування та чисельних методів. Для програмної реалізації завдання на дипломну роботу слід застосувати середовище розробки MS Visual Studio та інші програмні ресурси, мова програмування MS C#.

На основі матеріалів даної дипломної роботи можна в подальшому розробити лабораторну роботу з навчальної дисципліни «Об’єктно-орієнтоване програмування» за темою «Класи. Успадкування».

РОЗДІЛ 1. ГЕОМЕТРІЯ МАС

1.1 Характеристики розподілу мас

Як відомо, система матеріальних часток (СМТ), в якій відстань між двома її будь-якими точками лишається сталою, називається *незмінною системою матеріальних точок*. Якщо частки незмінної системи суцільним чином заповнюють область простору, яку вона займає, то остання називається *абсолютно твердим тілом* АТТ.

Абсолютно тверде тіло представляє собою ідеальний образ (модель), який тим ближче підходить до реального твердого тіла, чим менше воно спроможне деформуватися під дією зовнішніх та внутрішніх сил.

В динаміці АТТ велику роль відіграють величини, які характеризують просторовий розподіл мас в тілі. Ці величини називаються *моментами*.

Момент представляє собою суму добутків мас всіх точок системи на однорідну функцію координат цих точок, тобто має вигляд $\sum_i m_i x_i^\alpha y_i^\beta z_i^\gamma$, причому ціле число $n = \alpha + \beta + \gamma$ називається *степенем моменту*.

Якщо густина тіла є неперервною функцією координат $\rho = \rho(x, y, z)$, то момент буде вже виражатися у вигляді об'ємного інтеграла $\iiint \rho(x, y, z) x^\alpha y^\beta z^\gamma dxdydz$, який взятий за всім об'ємом цього тіла.

Для однорідних тіл (коли густина ρ від координат не залежить) момент представляє собою добуток густини та об'ємного інтегралу від функції координат $\rho \cdot \iiint x^\alpha y^\beta z^\gamma dxdydz$, та буде вже чисто геометричним моментом n -го степеня.

В механіці зустрічаються моменти *першого* та *другого степеня*, моменти вищих степенів використовуються в теорії міцності та задачах опору матеріалів.

1.2 Моменти першого степеня

Моменти першого степеня, які зазвичай називаються *статичними моментами*, виражаються величинами $\sum m_i \vec{r}_i$, де $\vec{r}_i$ – є радіус-вектором матеріальної частки з масою m_i відносно нерухомого центра O, що служить початком вектора $\vec{r}_i$. Відомо, що

$$\sum m_i \vec{r}_i = (\sum m_i x_i) \cdot \vec{\imath} + (\sum m_i y_i) \cdot \vec{\jmath} + (\sum m_i z_i) \cdot \vec{k} = M \cdot \vec{r}_C, \qquad (1)$$

де M є загальною масою всієї системи, а $\vec{r}_C$ –радіус-вектор центра мас.

Коли центр O співпадає з центром мас C, то статичний момент (1) обертається в нуль.

Величини $\sum m_i x_i, \sum m_i y_i, \sum m_i z_i$ представляють собою статичні моменти системи відносно координатних площин $Oyz, \quad Ozx, \quad Oxy$, причому

$$\sum m_i x_i = M \cdot x_C, \quad \sum m_i y_i = M \cdot y_C, \quad \sum m_i z_i = M \cdot z_C, \qquad (2)$$

де $x_C, \quad y_C, \quad z_C$ є просторовими координатами центра мас C даної системи відносно системи нерухомих осей $Oxyz$.

Якщо початок координат O співпадає з центром мас C, то всі статичні моменти (2) відносно координатних площин $Oyz, \quad Ozx, \quad Oxy$ для такої механічної системи будуть також дорівнювати нулю.

Якщо тіло є суцільним, то формули (2) набувають іншого функціонального вигляду (для координат центра мас C):

$$\begin{aligned} x_C &= \frac{1}{M} \iiint_V \rho(x,y,z) x \, dV, \quad y_C = \frac{1}{M} \iiint_V \rho(x,y,z) y \, dV, \\ z_C &= \frac{1}{M} \iiint_V \rho(x,y,z) z \, dV, \qquad M = \iiint_V \rho(x,y,z) \, dV. \end{aligned} \qquad (2)$$

1.3 Моменти другого степеня

Моменти другого степеня мають наступні зображення:

$$J_{(yz)} = \sum m_i x_i^2, \quad J_{yz} = \sum m_i y_i z_i, \quad J_{xx} = \sum m_i (y_i^2 + z_i^2),$$
$$J_{(zx)} = \sum m_i y_i^2, \quad J_{zx} = \sum m_i z_i x_i, \quad J_{yy} = \sum m_i (z_i^2 + x_i^2),$$
$$J_{(xy)} = \sum m_i z_i^2, \quad J_{xy} = \sum m_i x_i y_i, \quad J_{zz} = \sum m_i (x_i^2 + y_i^2),$$

$$J_O = \sum m_i (x_i^2 + y_i^2 + z_i^2) = \sum m_i r_i^2.$$

Перші три з них, що представляють суми добутків мас точок на квадрати їх відстаней до координатних площин Oyz, Ozx, Oxy, відповідно називаються *моментами інерції відносно цих площин*.

Другі три – *відцентровими моментами інерції*, треті – як суми добутків мас m_i на квадрати їх відстаней до координатних осей Ox, Oy, Oz – *осьовими моментами інерції*.

Сума добутків мас m_i точок на квадрати їх відстаней до центра координат O називається *полярним моментом інерції* відносно цього центра O.

З наведених вище означень можна зробити висновки, що:

- розмірність моменту другого степеню дорівнює кг $\cdot$ $м^2$;
- сума трьох площинних моментів інерції дорівнює полярному моменту інерції, тобто $J_{(yz)} + J_{(zx)} + J_{(xy)} = J_O$;
- сума трьох осьових моментів інерції дорівнює подвоєному полярному моменту інерції, тобто $J_{xx} + J_{yy} + J_{zz} = 2J_O$;
- сума двох осьових моментів інерції завжди більша за третій осьовий момент:

  $J_{xx} + J_{yy} > J_{zz}$, $J_{yy} + J_{zz} > J_{xx}$, $J_{zz} + J_{xx} > J_{yy}$.

1.4 Момент інерції відносно осі

В динаміці АТТ з моментів другого степеня переважно зустрічаються осьові та відцентрові моменти інерції. Якщо ми маємо суцільне АТТ, то маса елементарної частки тіла буде дорівнювати ρdV, де $dV = dxdydz$ – є елементарний об'єм частки, а $\rho = \rho(x, y, z)$ – густина елементарної частки із координатами (x, y, z) в системі $Oxyz$.

Тому осьовий момент інерції відносно осі Ox виразиться об'ємним інтегралом, який береться по всьому об'єму тіла

$$J_{xx} = \int \rho h^2 dV = \iiint_V \rho(x, y, z)(y^2 + z^2)dxdydz.$$

Якщо тіло однорідне, то його густина є константою $\rho = \rho_o$ і може бути винесеною за знак інтеграла, і тоді ми отримаємо $J_{xx} = \rho_o \cdot \int h^2 dV$, де $\int h^2 dV$ – *геометричний момент інерції* даного тіла, $h^2 = y^2 + z^2$.

Момент інерції АТТ відносно якої-небудь осі Ox можна виразити у вигляді $J_{xx} = \sum m_i h_i^2 = M \cdot i_{xx}^2$, де M – маса всієї системи (тіла), а $i_{xx} = \sqrt{\frac{J_{xx}}{M}}$ – є лінійною величиною, яка називається *радіусом інерції* АТТ відносно осі Ox. Радіус інерції i_{xx} є довжиною, яка дорівнює відстані від осі Ox до тієї точки G простору, в якій треба зосередити масу M всієї системи (тіла), щоб отримати той самий момент інерції J_{xx} для обраної віртуальної матеріальної точки G.

Інша задача: знайти таку масу M', яку треба зосередити в даній точці D, що знаходиться на заданій відстані d від осі Ox, щоб отримати для віртуальної матеріальної точки D той самий момент інерції J_{xx}.

Для визначення M' маємо рівність $M' \cdot d^2 = M \cdot i_{xx}^2$ звідси $M' = M \cdot \frac{i_{xx}^2}{d^2}$. Це є алгоритм зведення маси системи до однієї точки.

1.5 Моменти інерції АТТ відносно множини паралельних осей

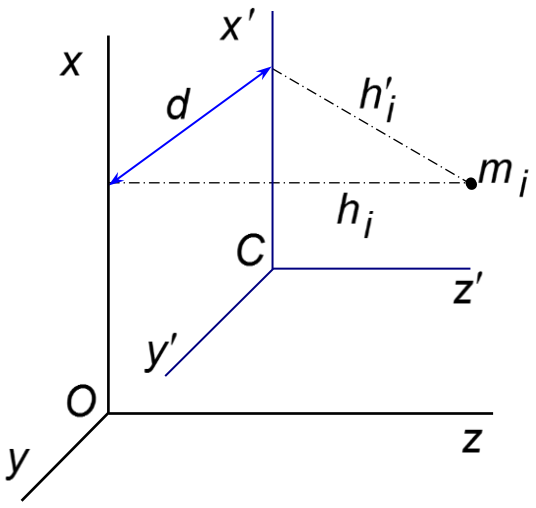
Розглянемо, як змінюється осевий момент інерції при переході від однієї осі x до іншої ξ, тобто розглянемо моменти інерції системи (тіла) відносно

множини паралельних осей та моменти інерції відносно пучка осей, що виходить із заданої точки. Знаючи, яким чином змінюється момент інерції при переході від однієї осі до іншої, ми можемо за даним моментом інерції відносно якої-небудь осі x знайти відповідний момент інерції відносно будь-якої іншої осі ξ. Для цього, обираючи довільну точку O на осі ξ, знаходимо спочатку момент інерції відносно осі x', що паралельна x та проходить через точку O, а потім від осі x' переходимо до осі ξ.

Встановимо спочатку залежність між моментами інерції відносно множини паралельних осей. Проведемо координатні осі $Oxyz$ та розглянемо момент інерції АТТ відносно осі Ox:

$$J_{xx} = \sum m_i h_i^2 = \sum m_i (y_i^2 + z_i^2). \qquad (3)$$

Візьмемо вісь Cx', яка є паралельною до осі Ox та проходить через центр мас системи $C(x_C, y_C, z_C)$, а також побудуємо осі Cy' та Cz', які є паралельними до осей Oy та Oz.



Тоді $y_i = y_i' + y_C$ та $z_i = z_i' + z_C$, де x_i', y_i', z_i' будуть координатами деякої точки тіла із масою m_i відносно системи осей $Cx'y'z'$.

Підставляючи ці рівності до виразів (3), будемо мати

$$J_{xx} = \sum m_i (y_i'^2 + z_i'^2) + \sum m_i (y_c^2 + z_c^2) + 2 \cdot \sum m_i (y_i' y_c + z_i' z_c).$$

Перший доданок правої частини рівності є моментом інерції $J_{x'x'}$ системи (тіла) відносно осі Cx', а другий доданок можна представити у вигляді $(y_c^2 + z_c^2) \cdot \sum m_i = d^2 \cdot M$, де M є загальною масою системи (тіла), а $d = \sqrt{y_c^2 + z_c^2}$ – відстань між осями Ox та Cx'.

Третій доданок дорівнює нулю, оскільки містить статичні моменти $\sum m_i y_i'$ та $\sum m_i z_i'$ відносно площин, що проходять через центр мас C.

Отже, $$J_{xx} = J_{x'x'} + M \cdot d^2. \quad (4)$$

Рівність (4) є теоремою Гюйгенса: «*момент інерції системи (тіла) відносно будь-якої осі дорівнює моменту інерції відносно паралельної осі, що проходить через центр мас системи (тіла), та складений з добутком маси всієї системи на квадрат відстані між цими осями*».

Для радіусів інерції системи (тіла) із теореми Гюйгенса маємо формулу:

$$i_{xx}^2 = i_{x'x'}^2 + d^2. \quad (5)$$

Із рівностей (4) та (5) виходить, що найменшим з моментів інерції АТТ відносно множини паралельних осей буде момент інерції відносно тої осі, що проходить через центр мас C даного тіла (системи).

Користуючись теоремою Гюйгенса можна отримати рівність

$$J_2 = J_1 + M \cdot (d_2^2 - d_1^2), \quad (6)$$

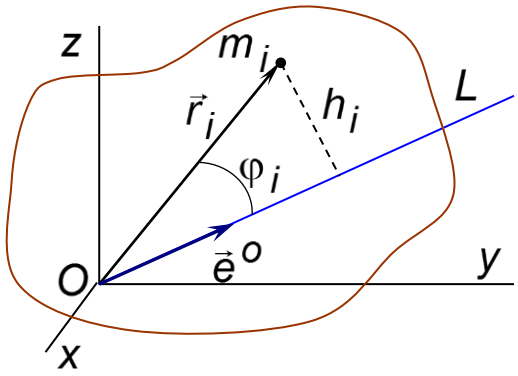
яка зв’язує значення моментів інерції АТТ відносно будь-яких паралельних осей, де d_1 та d_2 є відстані від цих осей до тієї осі з даної множини, що саме проходить через центр мас C даної системи (тіла).

Теорема Гюйгенса має силу не тільки для осевих моментів інерції.

ЇЇ можна поширити на всі моменти другого степеня, тобто аналогічним чином можна довести, що

$\sum m_i x_i^2 = \sum m_i x_i'^2 + M x_c^2$ і т.д., $\sum m_i y_i z_i = \sum m_i y_i' z_i' + M y_c z_c$ і т.д., $\sum m_i (x_i^2 + y_i^2 + z_i^2) = \sum m_i (x_i'^2 + y_i'^2 + z_i'^2) + M(x_c^2 + y_c^2 + z_c^2)$.

1.6 Моменти інерції відносно осей пучка, що виходить із однієї точки



Знайдемо момент інерції системи відносно вісі L, яка проходить через задану точку O та напрямок якої по відношенню до осей $Oxyz$ визначається одиничним вектором $\vec{e}^{\,o} = (\alpha, \beta, \gamma)$.

Згідно визначення осевого моменту інерції маємо $J_L = \sum m_i h_i^2$, де h_i – відстань від точки з масою m_i до обраної осі L.

Нехай точці $M_i(x_i, y_i, z_i)$ з масою m_i відповідає радіус-вектор $\vec{r}_i = x_i\vec{\imath} + y_i\vec{\jmath} + z_i\vec{k}$, тоді $h_i = r_i \cdot sin(\varphi_i)$, де φ_i – є кут між напрямками векторів $\vec{r}_i$ та $\vec{e}^{\,o}$, а $\{\alpha, \beta, \gamma\}$ є направляючими косинусами осі L, причому $\alpha^2 + \beta^2 + \gamma^2 = 1$.

Тоді, для осьового моменту інерції будемо мати:

$$J_L = \sum m_i h_i^2 = \sum m_i r_i^2 \, sin^2(\phi_i) = \sum m_i r_i^2 [1 - cos^2(\phi_i)] =$$
$$= \sum m_i [(x_i^2 + y_i^2 + z_i^2)(\alpha^2 + \beta^2 + \gamma^2) - (\alpha x_i + \beta y_i + \gamma z_i)^2] =$$
$$= \sum m_i [\alpha^2 (y_i^2 + z_i^2) + \beta^2 (z_i^2 + x_i^2) + \gamma^2 (x_i^2 + y_i^2) -$$
$$-2\beta\gamma y_i z_i - 2\gamma\alpha z_i x_i - 2\alpha\beta x_i y_i]. \qquad (7)$$

Застосуємо позначення:

$$J_{xx} = \sum m_i (y_i^2 + z_i^2), \quad J_{yy} = \sum m_i (z_i^2 + x_i^2), \quad J_{zz} = \sum m_i (x_i^2 + y_i^2),$$
$$J_{yz} = \sum m_i y_i z_i, \qquad J_{zx} = \sum m_i z_i x_i, \qquad J_{xy} = \sum m_i x_i y_i, \qquad (8)$$

та перепишемо (7) в компактній формі

$$J_L = J_{xx}\alpha^2 + J_{yy}\beta^2 + J_{zz}\gamma^2 - 2J_{yz}\beta\gamma - 2J_{zx}\gamma\alpha - 2J_{xy}\alpha\beta. \qquad (9)$$

1.7 Тензор інерції

Із рівності (9) виходить, що для визначення моменту інерції тіла відносно будь-якої осі з пучка, що проходить через задану точку O, достатньо знати

шість величин (2.1.8) та напрямок цієї осі, який визначається напрямними косинусами $\{\alpha, \beta, \gamma\}$.

Ці шість величин залежать від положення точки O та від напрямку координатних осей по відношенню до АТТ, оскільки при зміні розташування точки O та зміні напрямку координатних осей змінюватимуться координати x_i, y_i, z_i точок тіла m_i. Перелічені в (8) величини можна розташувати у вигляді симетричної матриці:

$$\boldsymbol{J} \equiv \begin{Bmatrix} J_{xx} & -J_{xy} & -J_{xz} \\ -J_{yx} & J_{yy} & -J_{yz} \\ -J_{zx} & -J_{zy} & J_{zz} \end{Bmatrix}, \tag{10}$$

яка носить назву *тензора інерції*, а її елементи називаються *компонентами тензора інерції*, причому діагональні компоненти є осьовими моментами інерції, а інші – добутки інерції, які взяті зі знаком «мінус».

Тензор $\boldsymbol{J}$ представляє собою символічний оператор, який, діючи на деякий вектор $\vec{a}$, дає інший вектор $\vec{b}$ з проекціями, які будуть лінійними функціями проекцій вектора $\vec{a}$. При цьому матрицею такого лінійного перетворення буде саме тензор інерції $\boldsymbol{J}$.

Вектор $\vec{b}$ називається лінійною вектор-функцією $\vec{a}$, а сама операція – операцією множення вектора $\vec{a}$ на тензор $\boldsymbol{J}$ та позначається $\vec{b} = \vec{a} \cdot \boldsymbol{J}$.

Рівність $\vec{b} = \vec{a} \cdot \boldsymbol{J}$ можна записати в проекціях:

$$\begin{aligned} b_x &= +J_{xx}a_x - J_{xy}a_y - J_{xz}a_z, \\ b_y &= -J_{yx}a_x + J_{yy}a_y - J_{yz}a_z, \\ b_z &= -J_{zx}a_x - J_{zy}a_y + J_{zz}a_z. \end{aligned}$$

Отже, якщо в якості вектора $\vec{a}$ взяти радіус-вектор деякої точки $M(a_x, a_y, a_z)$ тривимірного простору, що нами розглядається, то вектор $\vec{b}$ буде вказувати на відповідну точку $M'(b_x, b_y, b_z)$ того ж самого простору.

Таке перетворення простору називається *афінним*, а сам оператор $\boldsymbol{J}$ називається *афінором*.

Тензор інерції $\boldsymbol{J}$ представляє собою *симетричний афінор*.

Скористуємось тензором інерції для перетворення виразу (9).

Для цього візьмемо одиничний вектор $\vec{e}^{\,o} = \alpha\vec{\imath} + \beta\vec{\jmath} + \gamma\vec{k}$ напрямку осі L та помножимо його на тензор $\boldsymbol{J}$:

$$\vec{e}^{\,o} \cdot \boldsymbol{J} = (J_{xx}\alpha - J_{xy}\beta - J_{xz}\gamma) \cdot \vec{\imath} + (-J_{yx}\alpha + J_{yy}\beta - J_{yz}\gamma) \cdot \vec{\jmath} + (-J_{zx}\alpha - J_{zy}\beta + J_{zz}\gamma) \cdot \vec{k}\ .$$

Помножуючи вектор $\vec{e}^{\,o} \cdot \boldsymbol{J}$ скалярно на вектор $\vec{e}^{\,o}$ (виконуємо проектування вектора $\vec{e}^{\,o} \cdot \boldsymbol{J}$ на напрямок осі L), матимемо

$$(\vec{e}^{\,o} \cdot \boldsymbol{J}) \cdot \vec{e}^{\,o} = J_{xx}\alpha^2 + J_{yy}\beta^2 + J_{zz}\gamma^2 - 2J_{yz}\beta\gamma - 2J_{zx}\gamma\alpha - 2J_{xy}\alpha\beta = 2\Psi(\alpha,\beta,\gamma) = J_L\ \ ,$$

де $2\Psi(\alpha,\beta,\gamma)$ – квадратична форму аргументів α,β,γ.

Таким чином, знаючи компоненти тензора інерції $\boldsymbol{J}$, які обчислені для даної системи координатних осей $Oxyz$, можна знайти момент інерції АТТ для будь-якої іншої осі, що проходить через спільну точку O.

1.8 Еліпсоїд інерції.

Розглянемо геометричну інтерпретацію розподілу моментів інерції АТТ відносно пучка осей з центром в точці O шляхом побудування еліпсоїду інерції для цієї точки.

Візьмемо на осі L довільний відрізок OA із довжиною R, тоді координати точки $A(x,y,z)$ кінця цього відрізка будуть дорівнювати $x = \alpha R,\quad y = \beta R,\quad z = \gamma R.$

Підставляючи отримані з цих рівностей значення для напрямних косинусів α,β,γ до виразу (9) для J_L, отримаємо

$$J_L R^2 = J_{xx}x^2 + J_{yy}y^2 + J_{zz}z^2 - 2J_{yz}yz - 2J_{zx}zx - 2J_{xy}xy,$$

або $$J_L R^2 = 2\Psi(x, y, z).$$

Будемо тепер обирати довжину R так, щоб було

$$J_L R^2 = k^2 \quad \text{або} \quad R = \frac{k}{\sqrt{J_L}}, \quad \text{де } k \text{ є сталою величиною (константою),}$$

тоді матимемо

$$J_{xx}x^2 + J_{yy}y^2 + J_{zz}z^2 - 2J_{yz}yz - 2J_{zx}zx - 2J_{xy}xy = k^2, \qquad (11)$$

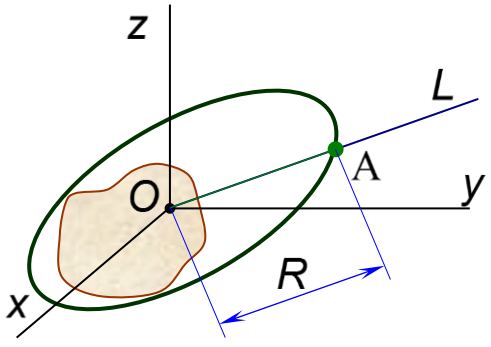
тобто геометричним місцем точок $A(x, y, z)$ буде поверхня 2-го порядку, яка визначається рівнянням (11).

Зрозуміло, що ця поверхня не має нескінченно віддалених точок, оскільки величина J_L завжди додатна та не дорівнює нулю.

Отже, поверхня із рівнянням (11) є еліпсоїдом, який носить назву *еліпсоїда інерції* АТТ відносно центра O.

Центр еліпсоїду знаходиться саме в точці O.

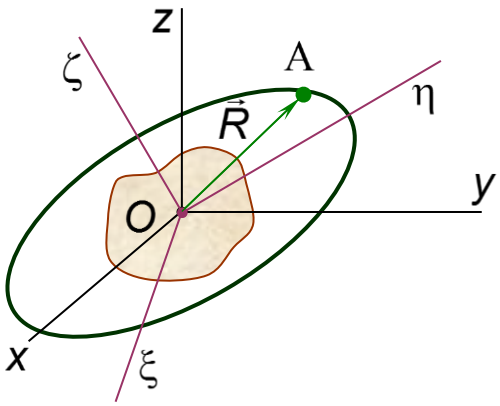
Стала k^2 може бути обраною довільно, та вона визначає масштаб побудування, тобто змінюючи k^2, ми будемо отримувати подібні еліпсоїди із спільним центром O.



Таким чином, кожній точці O даного тіла буде відповідати (з точністю до обраного масштабу k^2) цілком визначений еліпсоїд інерції.

Якщо точку O обирати в центрі мас C твердого тіла, то еліпсоїд інерції, побудований для цієї точки, буде називатися *центральним*. Головні осі еліпсоїда інерції називаються *головними осями інерції* тіла для точки O.

Якщо замість моменту інерції $J_L = M \cdot i_L^2$, використовувати відповідний радіус інерції i_L , то матимемо $R = \frac{k}{i_L \sqrt{M}}$, тобто довжина R радіус-вектора $\overrightarrow{OA}$ точки A еліпсоїда інерції на деякому напрямку OL буде зворотно пропорційною до радіусу інерції i_L твердого тіла відносно осі OL того ж напрямку.



1.9 Головні осі інерції.

Побудування головних осей інерції АТТ в даній точці O зводиться до визначення головних осей еліпсоїда інерції, побудованого для цієї точки.

Скористуємося рівнянням (2.1.11) еліпсоїда в системі координат $Oxyz$.

Для знаходження головних осей еліпсоїда інерції (11) будемо виходити з того, що для цих осей напрямок радіус-вектора $\overrightarrow{OA} = \vec{R}(x, y, z)$ співпадає з напрямком нормалі до поверхні цього еліпсоїда.

Ці напрямки повинні задовольняти умові колінеарності двох векторів:

$$\overrightarrow{grad}\{\Psi(x, y, z)\} = \lambda \cdot \vec{R}, \qquad (12)$$

де λ – коефіцієнт пропорційності.

Умова колінеарності (12) в проекціях на осі $Oxyz$ буде мати вигляд:

$$\frac{\partial \Psi}{\partial x} = \lambda x, \quad \frac{\partial \Psi}{\partial y} = \lambda y, \quad \frac{\partial \Psi}{\partial z} = \lambda z, \tag{13}$$

звідки, беручи до уваги (11) та переносячи всі члени (13) до лівої частини, матимемо систему рівнянь:

$$\left.\begin{aligned} (J_{xx} - \lambda)x - J_{xy}y - J_{xz}z = 0, \\ -J_{yx}x + (J_{yy} - \lambda)y - J_{yz}z = 0, \\ -J_{zx}z - J_{zy}y + (J_{zz} - \lambda)z = 0. \end{aligned}\right\} \tag{14}$$

Умовою спільності системи з трьох однорідних лінійних рівнянь (14) буде

$$\begin{vmatrix} J_{xx} - \lambda & -J_{xy} & -J_{xz} \\ -J_{yx} & J_{yy} - \lambda & -J_{yz} \\ -J_{zx} & -J_{zy} & J_{zz} - \lambda \end{vmatrix} = 0. \tag{15}$$

Рівняння (15) є характеристичним та у випадку симетричності тензора інерції $\boldsymbol{J}$ та дійсності його компонентів матимемо три дійсні корені $\lambda_1,\ \lambda_2,\ \lambda_3$.

Кожному з цих коренів відповідає дійсний напрямок, який задовольняє умові (12) та визначає напрямок головної осі еліпсоїда інерції. Можна показати, що ці три напрямки ξ, η, ζ є взаємно перпендикулярними.

Візьмемо тепер в якості осей координат головні осі інерції $O\xi\eta\zeta$ та розглянемо, як перетворюються при переході до цієї системи осей рівняння еліпсоїда інерції (11) та компоненти тензора інерції (10).

Для осі $O\xi$ (тобто коли вектор $\vec{R}$ спрямований вздовж осі $O\xi$) буде $\eta = 0$ і $\zeta = 0$, та з (13) ми отримаємо

$$\frac{\partial \Psi}{\partial \xi} = \lambda_1 \xi, \quad \frac{\partial \Psi}{\partial \eta} = 0, \quad \frac{\partial \Psi}{\partial \zeta} = 0. \tag{16}$$

Подібним же чином для осей $O\eta$ та $O\zeta$ із (13) будемо мати

$$\frac{\partial \Psi}{\partial \xi} = 0, \quad \frac{\partial \Psi}{\partial \eta} = \lambda_2 \eta, \quad \frac{\partial \Psi}{\partial \zeta} = 0, \quad \text{та} \quad \frac{\partial \Psi}{\partial \xi} = 0, \quad \frac{\partial \Psi}{\partial \eta} = 0, \quad \frac{\partial \Psi}{\partial \zeta} = \lambda_3 \zeta. \tag{17}$$

Із рівності (16) маємо для осі $O\xi$ ($\eta = 0$ і $\zeta = 0$):

$$J_{\xi\xi}\cdot\xi=\lambda_1\xi,\quad -J_{\eta\xi}\cdot\xi=0,\quad -J_{\zeta\xi}\cdot\xi=0,$$

тобто
$$\lambda_1=J_{\xi\xi},\quad J_{\eta\xi}=0,\quad J_{\zeta\xi}=0. \tag{18}$$

Аналогічно для осей $O\eta$ та $O\zeta$ матимемо

$$-J_{\xi\eta}\cdot\eta=0,\quad J_{\eta\eta}\cdot\eta=\lambda_2\eta,\quad -J_{\zeta\eta}\cdot\eta=0,$$

тобто
$$J_{\xi\eta}=0,\quad \lambda_2=J_{\eta\eta},\quad J_{\zeta\eta}=0. \tag{19}$$

$$-J_{\xi\zeta}\cdot\zeta=0,\quad -J_{\eta\zeta}\cdot\zeta=0,\quad J_{\zeta\zeta}\cdot\zeta=\lambda_3\zeta,$$

тобто
$$J_{\xi\zeta}=0,\quad J_{\eta\zeta}=0,\quad \lambda_3=J_{\zeta\zeta}. \tag{20}$$

Формули (18)–(20) показують, що:

- корені $\lambda_1,\lambda_2,\lambda_3$ рівняння (15) відповідно дорівнюють $J_{\xi\xi},J_{\eta\eta},J_{\zeta\zeta}$, тобто дорівнюють моментам інерції АТТ відносно головних осей інерції $O\xi\eta\zeta$;
- якщо за координатну вісь взяти головну вісь інерції, то добутки інерції, які містять координату, яка відповідає цій осі, дорівнюватимуть нулю, тобто, якщо вісь $O\xi$ є головною віссю інерції, то $J_{\xi\eta}=J_{\xi\zeta}=0$.

Моменти інерції $J_{\xi\xi},J_{\eta\eta},J_{\zeta\zeta}$ називаються *головними моментами інерції тіла* для центра O.

Якщо за вісі координат взяті головні вісі інерції $O\xi\eta\zeta$, то тензор інерції буде діагональним:

$$\boldsymbol{J}\equiv\begin{Bmatrix}J_{xx} & -J_{xy} & -J_{xz}\\ -J_{yx} & J_{yy} & -J_{yz}\\ -J_{zx} & -J_{zy} & J_{zz}\end{Bmatrix}\rightarrow\begin{Bmatrix}J_{\xi\xi} & 0 & 0\\ 0 & J_{\eta\eta} & 0\\ 0 & 0 & J_{\zeta\zeta}\end{Bmatrix},$$

а рівняння еліпсоїда інерції, віднесене до головних осей, та квадратична форма отримають вид

$$2\Psi(\xi,\eta,\zeta)=J_{\xi\xi}\xi^2+J_{\eta\eta}\eta^2+J_{\zeta\zeta}\zeta^2=k^2. \tag{21}$$

Якщо рівняння (15) має два рівних кореня $\lambda_1 = \lambda_2$, то в цьому випадку два головні моменти інерції будуть рівними між собою: $J_{\xi\xi} = J_{\eta\eta}$, та умові (13) будуть задовольняти всі можливі напрямки в площині $O\xi\eta$, що виходять з точки O.

Отже, всі осі в площині $O\xi\eta$ будуть головними осями інерції, а еліпсоїд інерції буде еліпсоїдом обертання з екваторіальною площиною $O\xi\eta$.

Якщо ж всі три корені будуть рівними $\lambda_1 = \lambda_2 = \lambda_3$, то $J_{\xi\xi} = J_{\eta\eta} = J_{\zeta\zeta}$ і еліпсоїд інерції перетворюється в сферу, а всі осі, що проходять через точку O, будуть головними.

1.10 Властивості головних осей інерції.

Якщо будь-яка вісь, наприклад координатна вісь Ox, є головною для точки O, то добутки інерції, які містять координату x, дорівнюватимуть нулю:

$$J_{xy} \equiv \sum m_i x_i y_i = 0, \quad J_{xz} \equiv \sum m_i x_i z_i = 0. \tag{22}$$

Справедливим буде і зворотне твердження, що якщо мають місце рівності (22), то вісь Ox буде головною віссю інерції, а рівняння еліпсоїда інерції буде

$$J_{xx}x^2 + J_{yy}y^2 + J_{zz}z^2 - 2J_{yz}yz = k^2.$$

Доведемо, що якщо однорідне **АТТ** має вісь геометричної симетрії, то ця вісь буде головною віссю інерції для всіх точок даної вісі.

Нехай Ox буде віссю симетрії однорідного абсолютно твердого тіла.

Тоді кожній частинці m_i тіла $M(x, y, z)$ буде відповідати така ж частинка $M'(x, -y, -z)$, а тому $\sum m_i x_i y_i = 0, \quad \sum m_i x_i z_i = 0$.

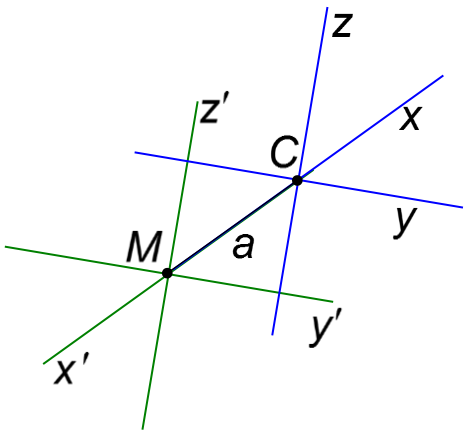
Отже, вісь Ox буде головною віссю інерції для точки O, причому це має місце для будь-якої точки O на цій осі симетрії.

Так само доводиться твердження, що якщо однорідне **АТТ** має площину симетрії, то для всіх точок цієї площини одна з головних осей інерції буде перпендикулярною до цієї площини.

Так, якщо приймемо площину симетрії за площину Oxy, то всякій частинці $M(x,y,z)$ буде відповідати така ж частинка $M'(x,y,-z)$, отже буде $\sum m_i z_i x_i = 0, \quad \sum m_i z_i y_i = 0$, а тому вісь Oz, яка буде перпендикулярною до площини Oxy, буде головною віссю інерції для точки O. Причому положення точки O на площині симетрії Oxy буде абсолютно довільним.

Головні осі інерції для центра мас АТТ називаються головними центральними осями інерції.

Точки, що лежать на головних центральних осях інерції тіла, володіють тією властивістю, що для всіх цих точок головні вісі інерції паралельні головним центральним осям інерції, а отже, і між собою.



Доведемо це.

Нехай C є центром мас твердого тіла, а x, y, z – його головні центральні осі інерції.

Візьмемо на осі Cx точку M, причому $CM = a$, та проведемо через M осі My' та Mz' паралельно осям Cy та Cz.

Так як вісь Cx є головною для точки C, то

$$\sum m_i x_i y_i = 0, \quad \sum m_i x_i z_i = 0.$$

Переходячи до системи координат x', y', z' з початком у точці M, маємо

$$x_i = x_i' + a, \quad y_i = y_i', \quad z_i = z_i'.$$

Підставляючи ці величини до попередніх рівностей, отримаємо

$$\sum m_i (x_i' + a) \cdot y_i' = 0, \quad \sum m_i (x_i' + a) \cdot z_i' = 0,$$

або $\sum m_i x_i' y_i' = -a \sum m_i y_i', \quad \sum m_i x_i' z_i' = -a \sum m_i z_i',$

але оскільки $\sum m_i y_i' = \sum m_i y_i = 0, \quad \sum m_i z_i' = \sum m_i z_i = 0,$ (як статичні моменти тіла відносно площин, що проходять через центр мас C), то $\sum m_i x_i' y_i' = 0, \quad \sum m_i x_i' z_i' = 0.$

Отже, вісь Cx буде головною віссю інерції для точки M.

Так як осі Cy та Cz будуть теж головними, то $\sum m_i z_i y_i = 0$, а звідси виходить, що $\sum m_i z_i' y_i' = 0$.

Можна зробити висновок про те, що осі My' та Mz' будуть також головними осями інерції для точки M, що й треба було довести. Необхідно відмітити, що інші точки тіла вказаною властивістю у загальному випадку не володіють.

РОЗДІЛ 2. НАБЛИЖЕНЕ ОБЧИСЛЕННЯ ПОДВІЙНИХ ІНТЕГРАЛІВ

2.1 Постановка задачі

Нехай необхідно обчислити інтеграл наступного виду:

$$I = \int_{A_1}^{B_1} \int_{A_2}^{B_2} f(\xi, \eta) d\xi d\eta, \quad (1)$$

де функція $f(\xi, \eta)$ є неперервною в заданій області

$$\{ A_1 \leq \xi \leq B_1, \quad A_2 \leq \eta \leq B_2 \}, \quad (2)$$

ξ і η – її Декартові змінні.

2.2 Попередні міркування

Функція $f(\xi, \eta)$ може суттєво змінювати свою «поведінку» на різних ділянках заданої області інтегрування (2).

Тому при використанні класичних квадратурних формул із сталими ваговими коефіцієнтами задля досягнення підсумкової шуканої точності при обчисленні інтегралу (1) на кожній наступній ітерації ми будемо вимушені значно збільшувати загальну кількість вузлів інтегрування у всій заданій області (2).

Такий спосіб не тільки суттєво збільшує процесорний час, який витрачається безпосередньо на арифметичні операції, але й «накопичує» значні помилки округлення.

Оптимізувати обчислювальний процес можна наступним чином (одразу зазначимо, що запропонований алгоритм не є єдиним можливим та універсальним).

Вся область інтегрування (2), яка являє собою прямокутник, (рис. 2.1), попередньо розбивається на однакові прямокутні підобласті, площі яких не перевищують однієї квадратної одиниці (на малюнку розбиття прямокутника на підобласті зображене штриховими лініями).

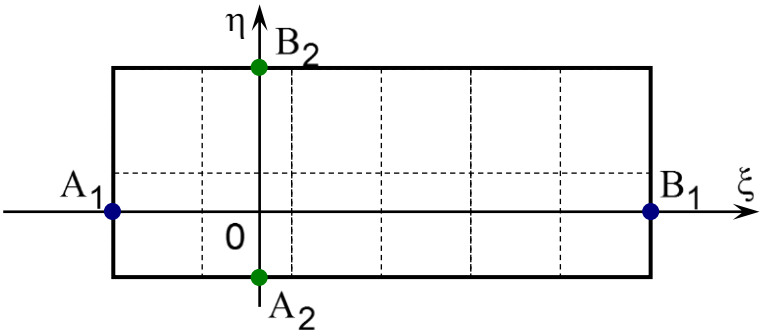


Рисунок 2.1 – Область інтегрування

Інтегрування виконується в кожній з таких підобластей, після чого всі часткові інтеграли підсумовуються, і ми отримуємо шукане значення інтегралу (1):

$$I = \int_{A_1}^{B_1}\int_{A_2}^{B_2} f(\xi,\eta)d\xi d\eta = \sum_{i=1}^{N_1}\sum_{j=1}^{N_2} I_{ij} = \sum_{i=1}^{N_1}\sum_{j=1}^{N_2}\left(\int_{a_i}^{b_i}\int_{a_j}^{b_j} f(\xi,\eta)d\xi d\eta\right), \quad (3)$$

Це дозволить виконувати «згущення» вузлів інтегрування тільки для тих підобластей, де це дійсно необхідно через істотні зміни в цій підобласті значень підінтегральної функції $f(\xi,\eta)$.

Виграш за часом обчислень та зменшення загальної кількості арифметичних операцій є цілком очевидними та зрозумілими.

Реалізувати вказане розбиття можна в такий спосіб.

Обчислимо розміри області інтегрування (2) в цілих числах, виконуючи перетворення довжини інтервалу до цілого з надлишком через метод $ceiling()$:

$$N_1 = ceiling(B_1 - A_1), \quad N_2 = ceiling(B_2 - A_2), \quad (4)$$

наприклад, якщо $A_1 = -1, \quad B_1 = \pi, \quad A_2 = 0.4, \quad B_2 = e,$ то маємо $N_1 = 5, \quad N_2 = 3.$

Тоді розміри однакових прямокутних підобластей можна обчислити за формулами:

$$h_1 = \frac{B_1 - A_1}{N_1}, \quad h_2 = \frac{B_2 - A_2}{N_2}, \quad \text{де} \quad h_1 \le 1, \quad h_2 \le 1, \quad (5)$$

при цьому площа кожної підобласті не перевищуватиме квадратну одиницю.

Для математичного опису підобластей ми скористаємося такими виразами:

$$\{\ a_i \le \xi \le b_i,\quad a_j \le \eta \le b_j\ \},\ i = \overline{1, N_1},\quad j = \overline{1, N_2}, \qquad (6)$$

де $a_i = A_1 + (i-1)\cdot h_1,\quad b_i = a_i + h_1,\quad a_j = A_2 + (j-1)\cdot h_2,\quad b_j = a_j + h_2.$ (7)

Тепер наше завдання полягає в обчисленні всіх подвійних інтегралів

$$I_{ij} = \int_{a_i}^{b_i}\int_{a_j}^{b_j} f(\xi,\eta)d\xi d\eta, \qquad (8)$$

де $f(\xi,\eta)$ – функція, яка неперервна в підобласті $\{\ a_i \le \xi \le b_i,\quad a_j \le \eta \le b_j\ \}$.

Розглянемо ітераційне уточнення наближеного значення інтегралу (8).

По-перше, виконаємо в (8) заміну лінійного типу для змінних ξ і η, так що новою областю інтегрування стане квадрат зі сторонами, які в точності дорівнюють 1:

$$\begin{cases}\xi = (b_i - a_i)\cdot x + a_i,\\ \eta = (b_j - a_j)\cdot y + a_j,\end{cases} \Rightarrow \begin{cases}d\xi = (b_i - a_i)\cdot dx,\\ d\eta = (b_j - a_j)\cdot dy,\end{cases} \Rightarrow$$

$$\begin{cases}\xi \in [a_i; b_i],\quad x \in [\ 0;\ 1\ ],\\ \eta \in [a_j; b_j],\quad y \in [\ 0;\ 1\ ].\end{cases} \qquad (9)$$

Тоді $$\int_{a_i}^{b_i}\int_{a_j}^{b_j} f(\xi,\eta)d\xi d\eta = (b_i - a_i)(b_j - a_j)\int_0^1\int_0^1 \varphi(x,y)dxdy, \qquad (10)$$

де $$\varphi(x,y) = f\big(\ (b_i - a_i)\cdot x + a_i,\ (b_j - a_j)\cdot y + a_j\ \big). \qquad (11)$$

По-друге, самі обчислювальні ітерації здійснюються за наступним алгоритмом.

Область інтегрування (9), яка представляє тепер собою квадрат з одиничними сторонами, послідовно розбивається на k^2 однакових «квадратних клітинок». (рис. 2.2)

При цьому довжина сторони такої «квадратної клітинки» дорівнюватиме $\Delta = \frac{1}{k}$.

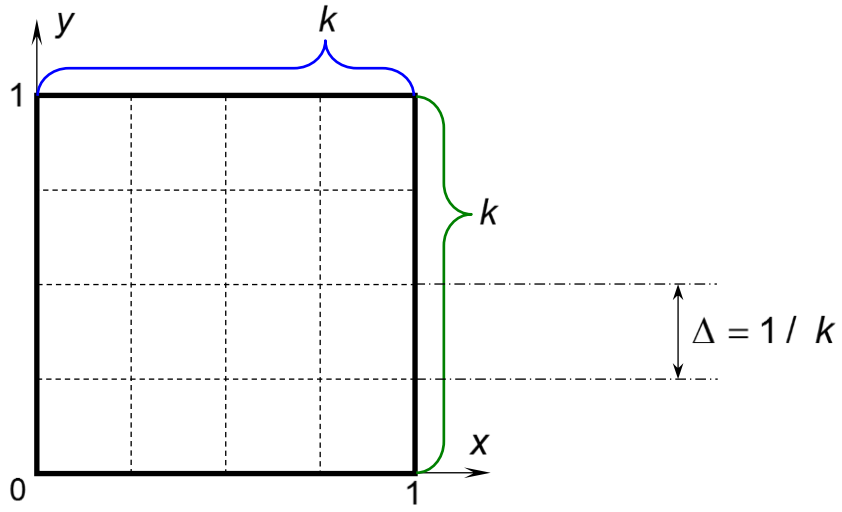


Рисунок 2.2 – Квадрат з квадратних клітинок

На кожній такій «квадратній клітинці» застосовується квадратурна формула (12) (рис. 2.3).

$$\tilde{I} = \frac{h^2}{81}\left[64 \cdot \phi(x_2, y_2) + 40 \cdot \left(\phi(x_2, y_1) + \phi(x_3, y_2) + \phi(x_2, y_3) + \phi(x_1, y_2)\right) + \right.$$
$$\left. +25 \cdot \left(\phi(x_1, y_1) + \phi(x_3, y_1) + \phi(x_3, y_3) + \phi(x_1, y_3)\right)\right], \quad (12)$$

де $h = \frac{\Delta}{2}$.

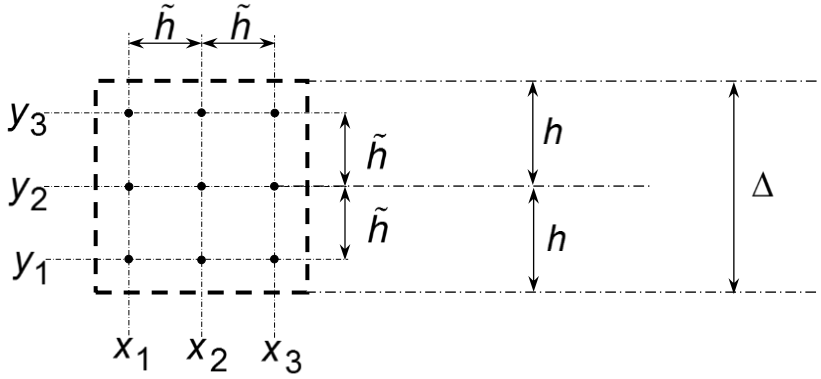


Рисунок 2.3 – Обчислювальна схема

Точка із координатами (x_2, y_2) є геометричним центром даної «клітинки».

Якщо ввести індексацію «клітинок» із використанням індексу m за координатою x та індексу n за координатою y, то для координат (x_2, y_2) центру «квадратної клітинки» з індексами (m, n) матимемо наступні формули:

$$\begin{aligned} x_2 &= (2m-1)\cdot h\ \cdot (b_i - a_i), \quad m = \overline{1,k}\ ; \\ y_2 &= (2n-1)\cdot h\ \cdot \left(b_j - a_j\right), \quad n = \overline{1,k}\ ; \end{aligned} \tag{13}$$

Координати всіх інших вузлів квадратурної формули (12) будуть обчислюватися через координати (x_2, y_2) центру поточної «клітинки» з індексами (m, n):

$$x_1 = x_2 - \tilde{h}, \quad x_3 = x_2 + \tilde{h}, \quad y_1 = y_2 - \tilde{h}, \quad y_3 = y_2 + \tilde{h}, \quad \text{де} \quad \tilde{h} = \sqrt{\frac{3}{5}}\cdot h. \tag{14}$$

Відповідно до зображення (11) аргументи підінтегральної функції повинні обчислюватися в такий спосіб:

$$\varphi\left(x_\alpha, y_\beta\right) = f\left(\ x_\alpha + a_i,\ \ y_\beta + a_j\ \right), \quad \text{де } \alpha, \beta = 1{,}2{,}3. \tag{15}$$

Якщо ж ввести наступні позначення (формули обчислень, які були реалізовані):

$$h_x = h\cdot(b_i - a_i), \quad h_y = h\cdot\left(b_j - a_j\right), \quad \tilde{h}_x = \sqrt{\frac{3}{5}}\cdot h_x, \quad \tilde{h}_y = \sqrt{\frac{3}{5}}\cdot h_y, \tag{16}$$

то для координат вузлів квадратурної формули (12) будемо мати:

$$\begin{aligned} x_2 &= (2m-1)\cdot h_x + a_i, \quad x_1 = x_2 - \tilde{h}_x, \quad x_3 = x_2 + \tilde{h}_x, \\ y_2 &= (2n-1)\cdot h_y + a_j, \quad y_1 = y_2 - \tilde{h}_y, \quad y_3 = y_2 + \tilde{h}_y, \end{aligned} \tag{17}$$

а тоді із врахуванням (17) матимемо:

$$\phi\left(x_\alpha, y_\beta\right) = f\left(x_\alpha,\ y_\beta\right), \text{ де } \alpha, \beta = 1{,}2{,}3. \tag{18}$$

Оскільки квадратурна формула (12) має місце лише для окремої «квадратної клітинки», загальна інтегральна сума може бути записаною в наступному виді:

$$\int_{a_i}^{b_i}\int_{a_j}^{b_j} f(\xi,\eta)d\xi d\eta = (b_i-a_i)\big(b_j-a_j\big)\int_0^1\int_0^1 \phi(x,y)dxdy = \sum_{m=1}^{k}\sum_{n=1}^{k}\tilde{I}_{mn}, \quad (19)$$

де $\tilde{I}_{mn}$ – наближене значення інтегралу для «квадратної клітинки» з індексами (m,n).

2.3 Тестування

Наступні подвійні інтеграли були обрані в якості тестових, оскільки вони мають відомі значення, які були отримані аналітичним шляхом.

1) $\int_0^{\pi}\int_0^{\pi}|cos(x+y)|dxdy = 2\pi;$ ([1, № 3971]);

2) $\int_0^1\int_0^1 xydxdy = \frac{1}{4};$ ([1, № 3901]);

3) $\int_{-1}^1\int_0^2\sqrt{|y-x^2|}dxdy = \frac{5}{3}+\frac{\pi}{2};$ ([1, № 3973]);

4) $\int_0^1\int_0^1(x+y)dxdy = 1;$ ([1, № 3906]);

5) $\int_0^1\int_0^1(x^2+y^2)dxdy = \frac{2}{3};$ ([1, № 4204a]);

6) $\int_0^1\int_0^1(x+y)^2dxdy = \frac{7}{6};$ ([1, № 4204b]);

7) $\int_{-c}^{+c}\int_{-d}^{+d}dxdy = 2c\cdot 2d = 4cd$ – площа прямокутника із сторонами $2c$ і $2d$

2.4 Програмна реалізація. Клас Quadrature2D

Клас Quadrature2D призначений для обчислення значень подвійних інтегралів виду (1).

```
namespace SOLID_DLL
{
  public class Quadrature2D
  {
    readonly double sqr6 = Math.Sqrt(0.60); double error;
    Func<double, double, double> Fxy;

    public Quadrature2D(Func<double, double, double> fxy) { Fxy = fxy; }

    public double Q2D(double A1, double B1, double A2, double B2, double eps)...

    double QScheem(double ai, double bi, double aj, double bj)...
  }
}
```

Рисунок 2.4 – Клас Quadrature2D

Основним членом цього класу є метод, який має сигнатуру відповідного узагальненого делегата Func<double,double,double>, та який містить імплементацію формули обчислень підінтегральної функції (1).

Безпосередні обчислення інтегралів здійснюються для існуючого екземпляра даного класу шляхом виклику його відкритого метода Q2D().(рис. 2.5)

```
public Quadrature2D(Func<double, double, double> fxy) { Fxy = fxy; }

public double Q2D(double A1, double B1, double A2, double B2, double eps)
{
  double I = 0.0, ai, bi, aj, bj;

  error = eps / 10.0;                          // Реалізація формул (3-4)
  int N1 = (int)Math.Ceiling(B1 - A1); double h1 = (B1 - A1) / N1;
  int N2 = (int)Math.Ceiling(B2 - A2); double h2 = (B2 - A2) / N2;

  for (int i = 1; i <= N1; i++)                // Реалізація формул (5-7)
  {
    ai = A1 + (i - 1) * h1; bi = ai + h1;
    for (int j = 1; j <= N2; j++)
    {
      aj = A2 + (j - 1) * h2; bj = aj + h2; I += QScheem(ai, bi, aj, bj);
    }
  }
  return I;                                    // Реалізація формули (8)
}
```

Рисунок 2.5 – Метод Q2D

Фактичними параметрами цього методу є границі інтегрування (1) та задана абсолютна похибка наближених обчислень.

Безпосередньо квадратурна схема та ітераційний алгоритм уточнення значення подвійного інтегралу реалізовані в закритому методі QScheem().(рис. 2.6)

```
double QScheem(double ai, double bi, double aj, double bj)
{
  double Io, Imn = double.MaxValue;
  double x1, x2, x3, y1, y2, y3, h, hx, hy, hhx, hhy;
  int k = 2, kk = 0;

  do
  {
    Io = Imn;    // запамятовуємо попереднє значення інтегралу
    Imn = 0.0;   // зануляємо поточне значення інтегралу
    kk++;        // лічильник ітерацій
    k *= 2;      // розбиття області інтегрування на k квадратних клітинок

    // крок квадратурної формули = половині сторони квадратика
    h = 0.5 / k;
    hx = (bi - ai) * h; hhx = hx * sqr6;
    hy = (bj - aj) * h; hhy = hy * sqr6;

    for (int m = 1; m <= k; m++)
    {
      x2 = (2 * m - 1) * hx + ai; x1 = x2 - hhx; x3 = x2 + hhx;
      for (int n = 1; n <= k; n++)
      {
        y2 = (2 * n - 1) * hy + aj; y1 = y2 - hhy; y3 = y2 + hhy;

        Imn += Fxy(x2, y2) * 64.0 +
               (Fxy(x2, y1) + Fxy(x3, y2) + Fxy(x2, y3) + Fxy(x1, y2)) * 40.0 +
               (Fxy(x1, y1) + Fxy(x3, y1) + Fxy(x3, y3) + Fxy(x1, y3)) * 25.0;
      }
    }
    Imn *= hx * hy / 81.0;
    if (Math.Abs(Imn - Io) < error) return Imn;
  } while (true);
}
```

Рисунок 2.6 – Метод QScheem

Задля підтвердження коректності наближеного обчислення подвійних інтегралів був створений відповідний проект Test_1 консольного додатку.(рис. 2.7)

В цьому додатку перевірялися обчислення тестових інтегралів, наведених в розділі *тестування*.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SQ = SOLID_DLL.Quadrature2D;

namespace Test_1
{
  class Main_T_01
  {
    static double A_1, B_1, A_2, B_2, Value, eps, True;

    static void Main(string[] args)...

    static double f_2c2d(double x, double y) => 1.0;
    static double f_3971(double x, double y) => Math.Abs(Math.Cos(x + y));
    static double f_3901(double x, double y) => x * y;
    static double f_3973(double x, double y) => Math.Sqrt(Math.Abs(y - x * x));
    static double f_3906(double x, double y) => x + y;
    static double f_4204a(double x, double y) => x * x + y * y;
    static double f_4204b(double x, double y) => (x + y) * (x + y);
  }
}
```

Рисунок 2.7 – Консольний додаток Test_1

Компіляція і запуск на виконання даної програми дає наступне вікно консольного додатку(рис. 2.8)

```
C:\Windows\system32\cmd.exe
  i_3971 =      6,28318527  true =      6,28318531
  i_3901 =      0,25000000  true =      0,25000000
  i_3973 =      3,23746298  true =      3,23746299
  i_3906 =      1,00000000  true =      1,00000000
 i_4204a =      0,66666667  true =      0,66666667
 i_4204b =      1,16666667  true =      1,16666667
  i_2c2d =      4,00000000  true =      4,00000000
Для продолжения нажмите любую клавишу . . .
```

Рисунок 2.8 – Результат роботи програми

Результати обчислень можна вважати більш ніж задовільними.

РОЗДІЛ 3. ФІГУРИ. КЛАСИ

3.1 Клас Point_3D

Клас Point_3D призначений для генерації об'єктів, геометричний зміст яких є точка тривимірного Евклідового простору.

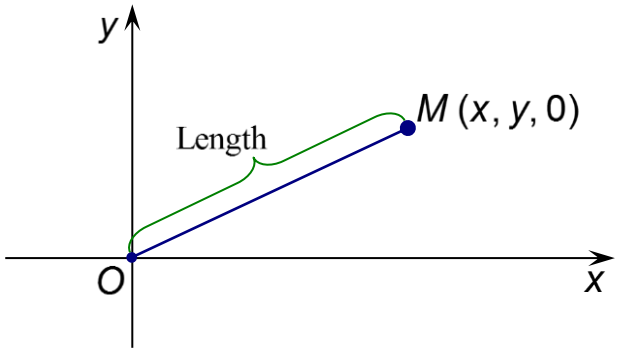
Сенс його членів є зрозумілим з контексту коду.(рис. 3.1)

Якщо екземпляр цього класу є «робочою» змінною, призначеною для проміжних обчислень, або носить суто допоміжний характер, то задавати ім'я точці не треба. У всіх інших випадках, особливо при проведенні тестових обчислень або при налагоджені алгоритмів, слід іменувати екземпляри посередництвом члена Name.

```
namespace SOLID_DLL
{
  public class Point_3D
  {
    public readonly double X, Y, Z; public readonly string Name;
    public double Length
    { get { return Math.Sqrt(X * X + Y * Y + Z * Z); } }
    public Point_3D(double x, double y, double z)
    { X = x; Y = y; Z = z; Name = "Unknown"; }
    public Point_3D(double x, double y, double z, string name)
    { X = x; Y = y; Z = z; Name = name; }
    public string ToPrint() =>
      $"[ X ={X,7:F2} ; Y ={Y,7:F2} ; Z ={Z,7:F2} ]" +
      $"  <{Name}>  Length ={Length,7:F2} \r\n";
  }
}
```

Рисунок 3.1 – Клас Point_3D

Якщо екземпляри цього класу використовуються в задачах із тілами в формі плоских фігур, слід використовувати перші дві просторові координати x та y, а третій координаті z задавати 0.0 в якості її значення.



Властивість Length визначає відстань від даної точки M до початку обраної системи просторових декартових координат Oxy.

3.2 Абстрактний Клас Figure_2D

Абстрактний клас Figure_2D призначений для поєднання в своєму складі базових членів, властивостей та методів, які будуть відноситись до всіх об’єктів даного дослідження, які відповідатимуть загальному геометричному поняттю – *плоска фігура*.

Загальна структура класу у згорнутому вигляді(рис. 3.2)

```
using System;
using Q_2D = SOLID_DLL.Quadrature2D;
namespace SOLID_DLL
{
  public abstract class Figure_2D
  {
    public Func<double, double, double> Density;
    public double Min_X, Min_Y, Max_X, Max_Y, OC, Massa, Square;
    public double J_Oz, J_Cz, J_Ox, J_Cx, J_Oy, J_Cy, J_xy, J_Cxy;
    public Point_3D Center; public string Result;

    public abstract void MinMax();
    public abstract bool Flag(double x, double y);
    public virtual string ToPrint() => "Figure_2D";

    double F_Ma(double x, double y)...
    double F_Sq(double x, double y)...
    double F_Cx(double x, double y)...
    double F_Cy(double x, double y)...
    double F_J_Oz(double x, double y)...
    double F_J_Ox(double x, double y)...
    double F_J_Oy(double x, double y)...
    double F_J_xy(double x, double y)...

    public void
      ToCalc(Func<double, double, double> density, double eps)...
  }
}
```

Рисунок 3.2 – Клас Figure_2D

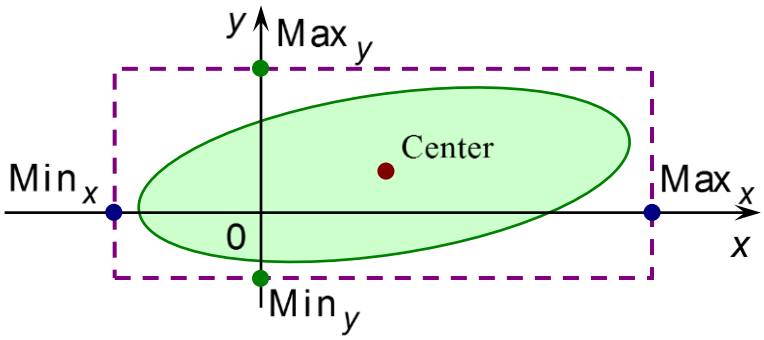
Основним членом цього класу є метод Density(), який має сигнатуру узагальненого делегата Func<double,double,double>, та який має містити закон розподілення густини фігури в залежності від координат точок цієї фігури, тобто $\rho = \rho(x, y)$.

Якщо фігура є однорідною, то значення густини фігури є константою, яку повинен повертати даний метод, тобто $\rho(x, y) = \rho_o = const$.

Детально зміст метода Density() буде розглянуто під час проведення тестових обчислень із відомими геометричними фігурами.

Члени Min_x, Min_y, Max_x, Max_y визначають координати допоміжного прямокутника, який повністю «покриває» ззовні фігуру, що розглядається.

На малюнку такою фігурою є еліпс (в якості наочного зразка).



Сторони цього прямокутника завжди є паралельними до координатних осей Oxy.

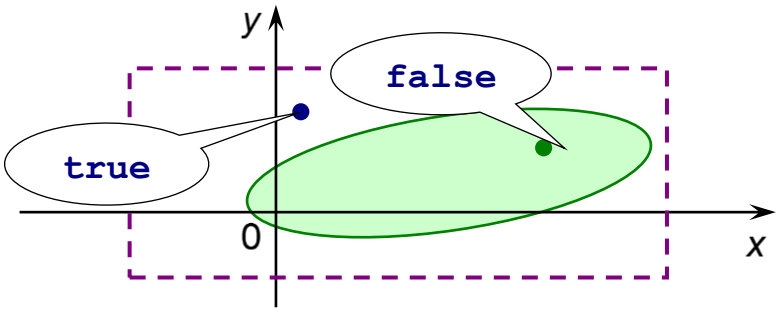
Для обчислення значень членів Min_x, Min_y, Max_x, Max_y слід імплементувати відповідний метод MinMax() в коді певного класу-нащадка.

Члени абстрактного класу Figure_2D, що залишилися, мають наступний зміст:

- OC – відстань від початку координат до центру мас даної фігури;
- Square – площа даної фігури;
- Massa – маса даної фігури;
- Center – точка на площині, яка є центром мас даної фігури;
- J_Oz – осьовий момент інерції відносно осі Oz;
- J_Ox – осьовий момент інерції відносно осі Ox;
- J_Oy – осьовий момент інерції відносно осі Oy;
- J_Cz – осьовий момент інерції відносно осі Cz, яка є паралельною до осі Oz, та проходить через центр мас C даної фігури;
- J_Cx – осьовий момент інерції відносно осі Cx, яка є паралельною до осі Ox, та проходить через центр мас C даної фігури;
- J_Cy – осьовий момент інерції відносно осі Cy, яка є паралельною до осі Oy, та проходить через центр мас C даної фігури;

- J_xy – добуток інерції відносно осі Oz;
- J_Cxy – добуток інерції відносно осі Cz, яка є паралельною до осі Oz, та проходить через центр мас C даної фігури.

Абстрактний метод Flag() призначений для ідентифікації точок допоміжного прямокутника за наступним правилом: якщо будь-яка геометрична точка $P(x, y)$, яка була довільно обрана в площині цього прямокутника, не належить фігурі, що досліджується, то метод повертає булево значення true.



Інакше – якщо точка $P(x, y)$ одночасно із прямокутником належить і площині фігури, що досліджується, то метод Flag() повертає булево значення false.

Абстрактний метод Flag() повинен бути імплементованим у відповідному класі-нащадку, де будуть враховані особливості математичного опису геометричної форми певної фігури, що буде досліджуватись.

Наступні методи абстрактного класу Figure_2D мають повертати значення підінтегральних виразів, які використовуються задля обчислення певних шуканих фізичних характеристик фігури, що досліджується.(рис. 3.3)

```
double F_Ma(double x, double y)
{ if (Flag(x, y)) return 0.0; return Density(x, y); }

double F_Sq(double x, double y)
{ if (Flag(x, y)) return 0.0; return 1.0; }

double F_Cx(double x, double y)
{ if (Flag(x, y)) return 0.0; return Density(x, y) * x; }

double F_Cy(double x, double y)
{ if (Flag(x, y)) return 0.0; return Density(x, y) * y; }

double F_J_Oz(double x, double y)
{ if (Flag(x, y)) return 0.0; return Density(x, y) * (x * x + y * y); }

double F_J_Ox(double x, double y)
{ if (Flag(x, y)) return 0.0; return Density(x, y) * y * y; }

double F_J_Oy(double x, double y)
{ if (Flag(x, y)) return 0.0; return Density(x, y) * x * x; }

double F_J_xy(double x, double y)
{ if (Flag(x, y)) return 0.0; return Density(x, y) * x * y; }
```

Рисунок 3.3 – Методи обчислень класу Figure_2D

Метод F_Ma() є підінтегральною функцією для $m = \iint \rho(x,y)dxdy$, за якою обчислюється загальна маса m фігури, що досліджується.

Ця функція за фізичною суттю є густиною фігури, тобто $\rho = \rho(x,y)$. Якщо фігура є однорідною, то $\rho = \rho_o = const$.

Метод F_Sq() є підінтегральною функцією для $S = \iint dxdy$, за якою обчислюється загальна площа S фігури, що досліджується.

Метод F_Cx() є підінтегральною функцією для обчислення координати x_C центра мас фігури, що досліджується.

При цьому $x_C = \frac{1}{m} \cdot \iint (\rho(x,y) \cdot x)dxdy$.

Метод F_Cy() є підінтегральною функцією для обчислення координати y_C центра мас фігури, що досліджується.

При цьому $y_C = \frac{1}{m} \cdot \iint (\rho(x,y) \cdot y)dxdy$.

Метод F_J_Cz() є підінтегральною функцією, призначеною для обчислення значення моменту інерції $J_{Oz} = \iint\big(\rho(x,y)\cdot(x^2+y^2)\big)dxdy$ даної фігури відносно координатної осі Oz.

Метод F_J_Cx() є підінтегральною функцією, призначеною для обчислення значення моменту інерції $J_{Ox} = \iint(\rho(x,y)\cdot y^2)dxdy$ даної фігури відносно координатної осі Ox.

Метод F_J_Cy() є підінтегральною функцією, призначеною для обчислення значення моменту інерції $J_{Oy} = \iint(\rho(x,y)\cdot x^2)dxdy$ даної фігури відносно координатної осі Oy.

Метод F_J_xy() є підінтегральною функцією, призначеною для обчислення значення добутку інерції $J_{xy} = \iint(\rho(x,y)\cdot x\cdot y)dxdy$ даної фігури відносно координатної осі Oz.

Відкритий метод ToCalc() призначений задля наближеного обчислення із заданою абсолютною похибкою всіх вище перелічених фізичних характеристик фігури.(рис. 3.4)

```
public void
  ToCalc(Func<double, double, double> density, double eps)
{
  Density = density;

  Q_2D massa = new Q_2D(F_Ma);
  Massa = massa.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps);

  Q_2D square = new Q_2D(F_Sq);
  Square = square.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps);

  Q_2D qCx = new Q_2D(F_Cx);
  double Cx = qCx.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps) / Massa;

  Q_2D qCy = new Q_2D(F_Cy);
  double Cy = qCy.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps) / Massa;

  Center = new Point_3D(Cx, Cy, 0.0, "Center");
  OC = Center.Length;

  Q_2D qJOz = new Q_2D(F_J_Oz);
  J_Oz = qJOz.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps);
  J_Cz = J_Oz - Massa * OC * OC;

  Q_2D qJOx = new Q_2D(F_J_Ox);
  J_Ox = qJOx.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps);
  J_Cx = J_Ox - Massa * Cy * Cy;

  Q_2D qJOy = new Q_2D(F_J_Oy);
  J_Oy = qJOy.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps);
  J_Cy = J_Oy - Massa * Cx * Cx;

  Q_2D qJxy = new Q_2D(F_J_xy);
  J_xy = qJxy.Q2D(Min_X, Max_X, Min_Y, Max_Y, eps);
  J_Cxy = J_xy - Massa * Cx * Cy;

  Result =
    $" Min_X ={Min_X,7:F2}  Max_X ={Max_X,7:F2} \r\n" +
    $" Min_Y ={Min_Y,7:F2}  Max_Y ={Max_Y,7:F2} \r\n\r\n" +
    $" Massa ={Massa,10:F5}  Square = {Square,10:F5}\r\n" +
    Center.ToPrint() + "\r\n" +
    $" J_Oz ={J_Oz,12:F5}  J_Cz = {J_Cz,12:F5}\r\n" +
    $" J_Ox ={J_Ox,12:F5}  J_Cx = {J_Cx,12:F5}\r\n" +
    $" J_Oy ={J_Oy,12:F5}  J_Cy = {J_Cy,12:F5}\r\n" +
    $" J_xy ={J_xy,12:F5} J_Cxy = {J_Cxy,12:F5}\r\n";
}
```

Рисунок 3.4 – Метод ToCalc

Для наближеного обчислення подвійних інтегралів застосовується метод, який був окремо розроблений в рамках даної роботи та протестований на відповідних інтегралах із відомими числовими значеннями.

Моменти інерції відносно осей $Cxyz$, яка є паралельною до системи координатних осей $Oxyz$, та проходить через центр мас $C(C_x, C_y, 0)$ даної фігури, обчислювалися за відомою формулою Гюйгенса-Штейнера:

$$\begin{cases} J_{Ox} = J_{Cx'} + m \cdot y_C^2, \\ J_{Oy} = J_{Cy'} + m \cdot x_C^2, \\ J_{Oz} = J_{Cz'} + m \cdot OC^2, \\ J_{xy} = J_{Cx'y'} + m \cdot x_C y_C, \end{cases} \text{ звідки маємо } \begin{cases} J_{Cx'} = J_{Ox} - m \cdot y_C^2, \\ J_{Cy'} = J_{Oy} - m \cdot x_C^2, \\ J_{Cz'} = J_{Oz} - m \cdot OC^2, \\ J_{Cx'y'} = J_{xy} - m \cdot x_C y_C, \end{cases}$$

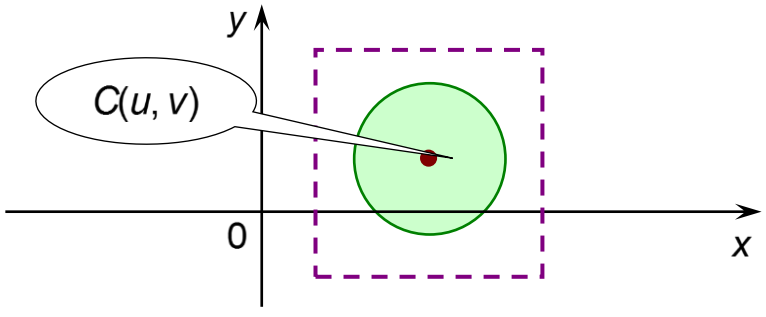
Віртуальний метод ToPrint() призначається задля підготовки довідкової інформації про значення ключових параметрів певних фігур та результатів обчислень.

Результати обчислень зберігаються в члені Result.

Метод ToPrint(), як правило, перевантажується у відповідних класах-нащадках, із урахуванням певних відмінностей між різними фігурами, та способами їх ініціалізації.

3.3 Клас Circle

Цей клас призначений задля математичного завдання фігури в формі звичайного кола, для якого задаються координати (u, v) його центру та його радіус R.



```
namespace SOLID_DLL
{
  public class Circle : Figure_2D
  {
    public readonly double u, v, R, S_Circle;
    public readonly string Name = "Circle: ";

    #region    < конструктори >
    public static
      Circle New(double ux, double vy, double r, string n)
    {
      if (r < 1.0) return null; return new Circle(ux, vy, r, n);
    }
    Circle(double ux, double vy, double r, string name)
    {
      u = ux; v = vy; R = r; S_Circle = Math.PI * R * R;
      MinMax();
      if (name != "") Name += name;
    }
    #endregion < конструктори >

    public override void MinMax()...
    public override bool Flag(double x, double y)...
    public override string ToPrint()...
  }
}
```

Рисунок 3.5 – Клас Circle

Площа такої фігури обчислюється за формулою $S = \pi R^2$.

Рівняння кола, що відповідає заданим параметрам, має наступний функціональний вид: $(x-u)^2 + (y-v)^2 = R^2$.

На основі саме цього рівняння формулюється математична умова для метода Flag()(рис. 3.6):

«*Якщо для точки із координатами* (x, y) *виконується нерівність* $\sqrt{(x-u)^2 + (y-v)^2} > R$, *то ця точка не потрапляє в задане коло*»(рис. 3.6):

```
public override bool Flag(double x, double y)
{
  return R < Math.Sqrt((x - u) * (x - u) + (y - v) * (y - v));
}
```

Рисунок 3.6 – Метод Flag

Координати прямокутника, який повністю «покриває» задане коло, обчислюються у відповідності до наступних формул:

$$Min_x = u - R - 1; \quad Max_x = u + R + 1;$$
$$Min_y = v - R - 1; \quad Max_y = v + R + 1.$$

Отже, в нашому випадку це буде квадрат. Відповідна імплементація абстрактного метода MinMax() буде наступною(рис. 3.7):

```
public override void MinMax()
{
  Min_X = u - R - 1.0; Max_X = u + R + 1.0;
  Min_Y = v - R - 1.0; Max_Y = v + R + 1.0;
}
```

Рисунок 3.7 – Метод MinMax

Перевантажений віртуальний метод ToPrint() матиме код(рис. 3.8):

```
public override string ToPrint()
{
  string txt = Name;
  txt += $" u ={u,7:F2} v ={v,7:F2} R ={R,7:F2} \r\n\r\n";
  return txt + Result;
}
```

Рисунок 3.8 – Метод ToPrint

Для екземплярів даного класу було прийнято штучне обмеження на величину R радіусу кола, а саме – радіус кола не повинен бути меншим за одиницю довжини, яка приймається однаковою для всіх фігур.

Задля підтвердження коректності фізичного та програмного моделювання фігур в формі кола був створений відповідний проект Test_Circle консольного додатку.

В цьому додатку перевірялися обчислення всіх шуканих фізичних характеристик для кола із заданими параметрами:

$$u = 4.0; \quad v = 3.0; \quad R = 2.0; \quad \rho = \rho_o = \frac{3}{\pi}, \text{ або}$$

$$u = -4.0; \quad v = -3.0; \quad R = 2.0; \quad \rho = \rho_o = \frac{3}{\pi}.$$

З курсу теоретичної механіки відомо, що осьовий момент інерції кола по відношенню до осі Cz', яка проходить через його центр C ортогонально площині цього кола, буде дорівнювати: $J_{Cz'} = \frac{1}{2}mR^2$.

Інші два моменти інерції, відносно осей Cx' та Cy' будуть: $J_{Cx'} = J_{Cy'} = \frac{1}{4}mR^2$.

Тому, для заданого кола ми повинні отримати наступні результати:

$S = \pi R^2 = 4\pi \approx 12.56637, \quad m = \rho_o S = 12.00000,$

$x_C = +4.00000, \quad y_C = +3.00000, \quad OC = 5.00000,$ або

$x_C = -4.00000, \quad y_C = -3.00000, \quad OC = 5.00000,$

$J_{Cz'} = \frac{1}{2}mR^2 = 24.00000, J_{Oz} = \frac{1}{2}mR^2 + m \cdot OC^2 = 324.00000,$

$J_{Cx'} = \frac{1}{4}mR^2 = 12.00000, J_{Ox} = \frac{1}{4}mR^2 + m \cdot y_C^2 = 120.00000,$

$J_{Cy'} = \frac{1}{4}mR^2 = 12.00000, J_{Oy} = \frac{1}{4}mR^2 + m \cdot x_C^2 = 204.00000,$

$J_{Cx'y'} = 0.00000, J_{xy} = m \cdot x_C \cdot y_C = 144.00000.$

Нижче наводиться код консольного додатку Test_Circle(рис. 3.9):

```
using System;
using Cr = SOLID_DLL.Circle;
namespace Test_Circle
{
  class Main_T_Circle
  {
    static void Main(string[] args)
    {
      double u = +4.0, v = +3.0, R = 2.0;
      Cr circle = Cr.New(u, v, R, "test");
      circle.ToCalc(Ro, 1.0E-3);
      Console.WriteLine(circle.ToPrint());
    }
    static double Ro(double x, double y) => 3.0 / Math.PI;
  }
}
```

Рисунок 3.9 – Консольний додаток Test_Circle

Виконання даної програми дає наступні результати (для двох варіантів вхідних даних)(рис. 3.10)(рис. 3.11):

$u = 4.0; \quad v = 3.0; \quad R = 2.0; \quad \rho = \rho_o = \frac{3}{\pi}$:

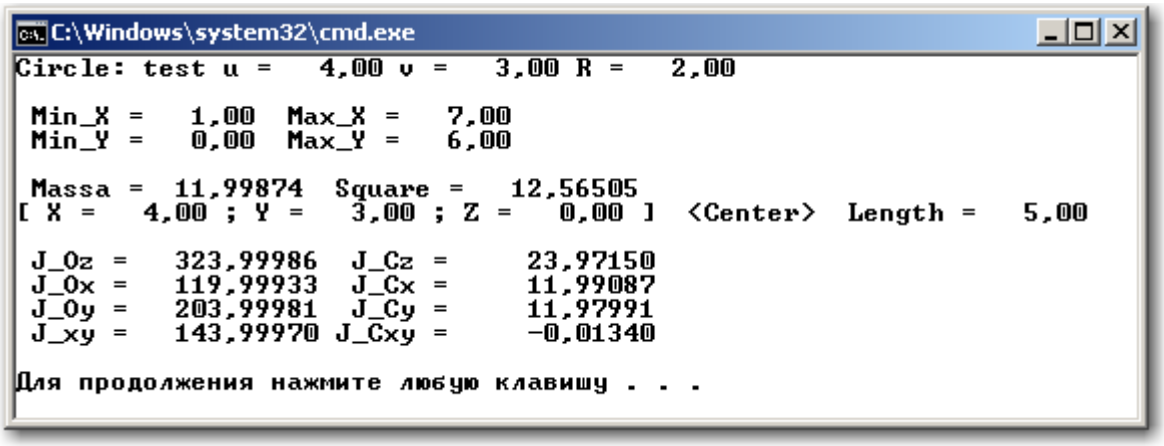


Рисунок 3.10 – Результат першого варіанту

$u = -4.0; \quad v = -3.0; \quad R = 2.0; \quad \rho = \rho_o = \frac{3}{\pi}$:

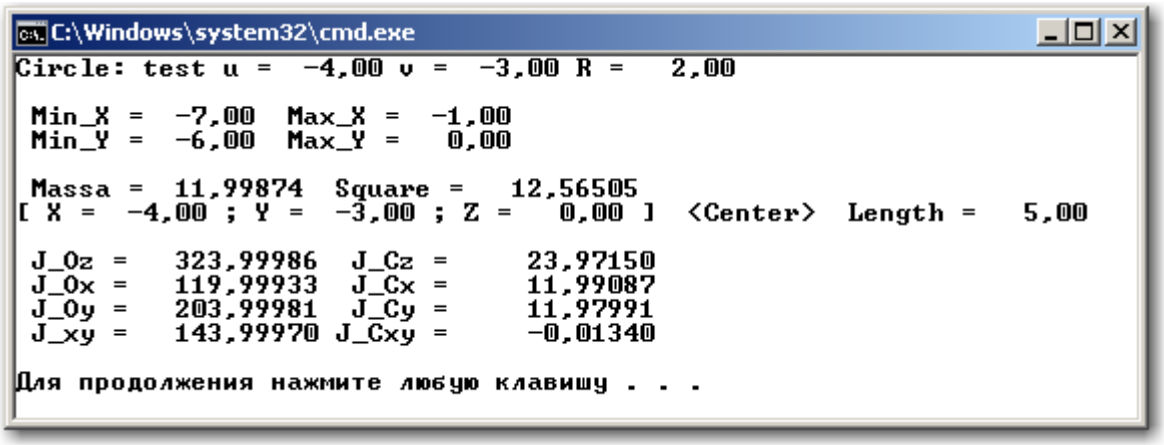


Рисунок 3.11 – Результат другого варіанту

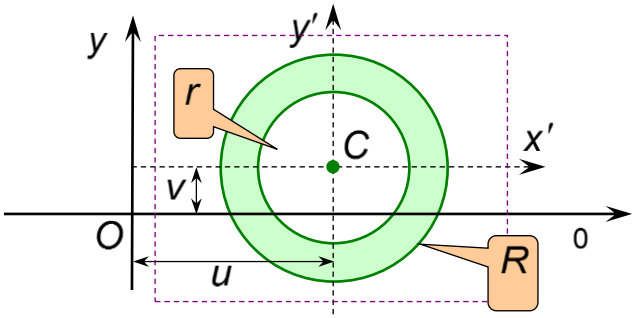
Результати обчислень можна вважати більш ніж задовільними, оскільки відмінність наближених значень параметрів (моментів інерції) по відношенню до їх точних значень складає менше 1%.

Слід зазначити, що зменшення похибки обчислень буде супроводжуватися значним збільшенням загального часу обчислень.

Наведені вище результати отримані для значення абсолютної похибки обчислень $\varepsilon \sim 10^{-3}$.

3.4 Клас Tor

Цей клас призначений задля математичного завдання фігури в формі звичайного тора, для якого задаються координати (u, v) його центру C та його зовнішній R та внутрішній радіуси r(рис. 3.12).




```
namespace SOLID_DLL
{
  public class Tor : Figure_2D
  {
    public readonly double u, v, R, r, S_Tor;
    public readonly string Name = "Tor: ";

    #region    < конструктори >
    public static
      Tor New(double ux, double vy, double r2, double r1, string n)
    {
      if (r2 < 1.0) return null; if (r1 < 0.5) return null;
      if (r2 <= r1) return null;
      return new Tor(ux, vy, r2, r1, n);
    }
    Tor(double ux, double vy, double r2, double r1, string name)
    {
      u = ux; v = vy; R = r2; r = r1;
      S_Tor = Math.PI * (R * R - r * r); MinMax();
      if (name != "") Name += name;
    }
    #endregion < конструктори >

    public override void MinMax()...
    public override bool Flag(double x, double y)...
    public override string ToPrint()...
  }
}
```

Рисунок 3.12 – Клас Tor

Площа такої фігури обчислюється за формулою $S = \pi(R^2 - r^2)$.

На основі рівнянь для зовнішнього кола $(x-u)^2+(y-v)^2=R^2$ та внутрішнього кола $(x-u)^2+(y-v)^2=r^2$, що відповідають заданим параметрам, формулюється математична умова для метода Flag()(рисунок 3.13):

«*Якщо для довільної точки із координатами* (x,y) *виконується нерівність* $\sqrt{(x-u)^2+(y-v)^2}<r$ *або нерівність* $\sqrt{(x-u)^2+(y-v)^2}>R$, *то ця точка не потрапляє в задану фігуру – тор*»(рис. 3.13):

```
public override bool Flag(double x, double y)
{
    double d = Math.Sqrt((x - u) * (x - u) + (y - v) * (y - v));
    return (r > d) || (d > R);
}
```

Рисунок 3.13 – Метод Flag

Координати прямокутника, який повністю «покриває» заданий тор, обчислюються у відповідності до наступних формул:

$$Min_x = u - R - 1; \quad Max_x = u + R + 1;$$
$$Min_y = v - R - 1; \quad Max_y = v + R + 1.$$

Отже, в нашому випадку це буде квадрат. Відповідна імплементація абстрактного метода MinMax() буде наступною(рис. 3.14):

```
public override void MinMax()
{
    Min_X = u - R - 1.0; Max_X = u + R + 1.0;
    Min_Y = v - R - 1.0; Max_Y = v + R + 1.0;
}
```

Рисунок 3.14 – Метод MinMax

Перевантажений віртуальний метод ToPrint() матиме код(рис. 3.15):

```
public override string ToPrint()
{
    string txt = Name;
    txt += $" u ={u,7:F2} v ={v,7:F2} r ={r,7:F2} R ={R,7:F2} \r\n\r\n";
    return txt + Result;
}
```

Рисунок 3.15 – Метод ToPrint

Для екземплярів даного класу було прийнято штучне обмеження на величини радіусів тора, а саме – внутрішній радіус кола не повинен бути меншим за 0.5, та зовнішній радіус повинен бути більшим за одиницю.

Задля підтвердження коректності фізичного та програмного моделювання фігур в формі тора був створений відповідний проект Test_Tor консольного додатку.

В цьому додатку перевірялися обчислення всіх шуканих фізичних характеристик для тора із заданими параметрами:

$$u = 4.0; \quad v = 3.0; \quad r = 1.0; \quad R = 2.0; \quad \rho_o = \frac{2}{\pi};$$

або $$u = -4.0; \quad v = -3.0; \quad r = 1.0; \quad R = 2.0; \quad \rho_o = \frac{2}{\pi}.$$

З курсу теоретичної механіки відомо, що осьовий момент інерції тора по відношенню до осі Cz', яка проходить через його центр C ортогонально площині цього тора, буде дорівнювати: $J_{Cz'} = \frac{1}{2}m(R^2 + r^2)$.

Інші два моменти інерції, відносно осей Cx' та Cy' будуть: $J_{Cx'} = J_{Cy'} = \frac{1}{4}m(R^2 + r^2)$.

Тому, для заданого тора ми повинні отримати наступні результати:

$S = \pi(R^2 - r^2) = 3\pi \approx 9.42478, \quad m = \rho_o S = 6.00000,$

$x_C = +4.00000, \quad y_C = +3.00000, \quad OC = 5.00000,$ або

$x_C = -4.00000, \quad y_C = -3.00000, \quad OC = 5.00000,$

$J_{Cz'} = 15.00000, J_{Oz} = J_{Cz'} + m \cdot OC^2 = 165.00000,$

$J_{Cx'} = 7.50000, J_{Ox} = J_{Cx'} + m \cdot y_C^2 = 103.50000,$

$J_{Cy'} = 7.50000, J_{Oy} = J_{Cy'} + m \cdot x_C^2 = 61.50000,$

$J_{Cx'y'} = 0.00000, J_{xy} = m \cdot x_C \cdot y_C = 72.00000.$

Нижче наводиться код консольного додатку Test_Tor(рис. 3.16):

```
using System;
using Tor = SOLID_DLL.Tor;
namespace Test_Tor
{
  class Main_T_Tor
  {
    static void Main(string[] args)
    {
      double u = -4.0, v = -3.0, r2 = 2.0, r1 = 1.0 ;
      Tor tor = Tor.New(u, v, r2, r1, "test");
      tor.ToCalc(Ro, 1.0E-3);
      Console.WriteLine(tor.ToPrint());
    }
    static double Ro(double x, double y) => 2.0 / Math.PI;
  }
}
```

Рисунок 3.16 – Консольний додаток Test_Tor

Виконання програми дає наступні результати(рис. 3.17)(рис. 3.18) :

```
C:\Windows\system32\cmd.exe
Tor: test u =   4,00 v =   3,00 r =   1,00 R =   2,00

 Min_X =   1,00  Max_X =   7,00
 Min_Y =   0,00  Max_Y =   6,00

 Massa =   5,99933  Square =    9,42342
[ X =   4,00 ; Y =   3,00 ; Z =   0,00 ]  <Center>  Length =   5,00

 J_Oz =   164,99971  J_Cz =     14,98590
 J_Ox =    61,49949  J_Cx =      7,49506
 J_Oy =   103,49950  J_Cy =      7,49012
 J_xy =    71,99977 J_Cxy =     -0,00670

Для продолжения нажмите любую клавишу . . .
```

Рисунок 3.17 – Результат першого варіанту

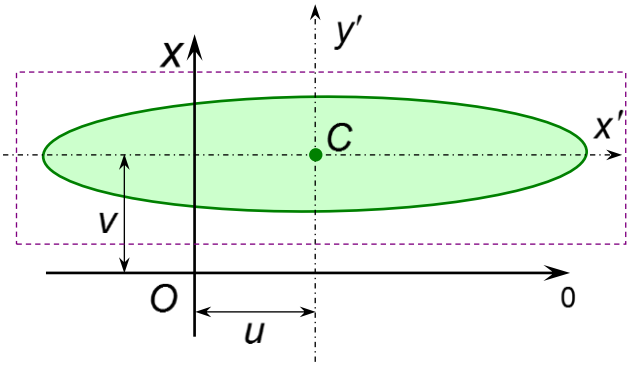
```
C:\Windows\system32\cmd.exe
Tor: test u =  -4,00 v =  -3,00 r =   1,00 R =   2,00

 Min_X =  -7,00  Max_X =  -1,00
 Min_Y =  -6,00  Max_Y =   0,00

 Massa =   5,99933  Square =    9,42342
[ X =  -4,00 ; Y =  -3,00 ; Z =   0,00 ]  <Center>  Length =   5,00

 J_Oz =   164,99971  J_Cz =     14,98590
 J_Ox =    61,49949  J_Cx =      7,49506
 J_Oy =   103,49950  J_Cy =      7,49012
 J_xy =    71,99977 J_Cxy =     -0,00670

Для продолжения нажмите любую клавишу . . .
```

Рисунок 3.18 - Результат другого варіанту

Результати обчислень для фігури тор можна теж вважати більш ніж задовільними, оскільки відмінність наближених значень параметрів (моментів інерції) по відношенню до їх точних значень складає менше 1%.

3.5 Клас Ellipse

Цей клас призначений задля математичного завдання фігури в формі еліпса, для якого задаються координати (u, v) його центру C та довжини його півосей a та b. (рис. 3.19)




```
namespace SOLID_DLL
{
  public class Ellipse : Figure_2D
  {
    public readonly double a, b, u, v, S_Ellipse;
    public readonly string Name = "Ellipse: ";

    #region    < конструктори >
    public static
      Ellipse New(double ax, double by, double ux, double vy, string n)
    {
      if (ax < 1.0) return null; if (by < 1.0) return null;
      return new Ellipse(ax, by, ux, vy, n);
    }
    Ellipse(double ax, double by, double ux, double vy, string name)
    {
      a = ax; b = by; u = ux; v = vy;
      S_Ellipse = Math.PI * a * b; MinMax();
      if (name != "") Name += name;
    }
    #endregion < конструктори >

    public override void MinMax()...
    public override bool Flag(double x, double y)...
    public override string ToPrint()...
  }
}
```

Рисунок 3.19 – Клас Ellipse

Площа такої фігури обчислюється за формулою $S = \pi ab$.

На основі рівняння еліпса $\frac{(x-u)^2}{a^2} + \frac{(y-v)^2}{b^2} = 1$ формулюється математична умова для метода Flag()(рис. 3.20):

«*Якщо для довільної точки із координатами* (x, y) *виконується нерівність* $\frac{(x-u)^2}{a^2} + \frac{(y-v)^2}{b^2} > 1$, *то ця точка не потрапляє в задану фігуру – еліпс*»(рис. 3.20):

```
public override bool Flag(double x, double y)
{
    double xx = (x - u) / a, yy = (y - v) / b;
    return Math.Sqrt(xx * xx + yy * yy) > 1.0;
}
```

Рисунок 3.20 – Метод Flag

Координати прямокутника, який повністю «покриває» заданий еліпс, обчислюються у відповідності до наступних формул:

$$Min_x = u - a - 1; \quad Max_x = u + a + 1;$$
$$Min_y = v - b - 1; \quad Max_y = v + b + 1.$$

Відповідна імплементація абстрактного метода MinMax() буде наступною(рис. 3.21):

```
public override void MinMax()
{
    Min_X = u - a - 1.0; Max_X = u + a + 1.0;
    Min_Y = v - b - 1.0; Max_Y = v + b + 1.0;
}
```

Рисунок 3.21 - Метод MinMax

Перевантажений віртуальний метод ToPrint() матиме код(рис. 3.22):

```
public override string ToPrint()
{
    string txt = Name;
    txt += $" a ={a,7:F2} b ={b,7:F2} u ={u,7:F2} v ={v,7:F2} \r\n\r\n";
    return txt + Result;
}
```

Рисунок 3.22 – Метод ToPrint

Для екземплярів даного класу було прийнято штучне обмеження на довжини півосей, а саме – вони не повинні бути меншими за 1.0.

Задля підтвердження коректності фізичного та програмного моделювання фігур в формі еліпса був створений відповідний проект Test_Ellipse консольного додатку.

В цьому додатку перевірялися обчислення всіх шуканих фізичних характеристик для тора із заданими параметрами:

$$u = 0.0; \quad v = 0.0; \quad a = 2.0; \quad b = 1.0; \quad \rho_o = \frac{4}{\pi};$$

або $$u = 4.0; \quad v = 3.0; \quad a = 2.0; \quad b = 1.0; \quad \rho_o = \frac{4}{\pi}.$$

З курсу теоретичної механіки відомо, що осьовий момент інерції еліпсу по відношенню до осі Cz', яка проходить через його центр C ортогонально площині цього тора, буде дорівнювати: $J_{Cz'} = \frac{1}{4}m(a^2 + b^2)$.

Інші два моменти інерції, відносно осей Cx' та Cy' будуть: $J_{Cx'} = \frac{1}{4}mb^2$ та $J_{Cy'} = \frac{1}{4}ma^2$.

Тому, для заданого тора ми повинні отримати наступні результати:

$S = \pi ab = 2\pi \approx 6.28319, \quad m = \rho_o S = 8.00000,$

$x_C = +4.00000, \quad y_C = +3.00000, \quad OC = 5.00000,$ або

$x_C = -4.00000, \quad y_C = -3.00000, \quad OC = 5.00000,$

$J_{Cz'} = 10.00000, J_{Oz} = J_{Cz'} + m \cdot OC^2 = 210.00000,$

$J_{Cx'} = 2.00000, J_{Ox} = J_{Cx'} + m \cdot y_C^2 = 74.00000,$

$J_{Cy'} = 8.00000, J_{Oy} = J_{Cy'} + m \cdot x_C^2 = 136.00000,$

$J_{Cx'y'} = 0.00000, J_{xy} = m \cdot x_C \cdot y_C = 96.00000.$

Нижче наводиться код консольного додатку Test_Ellipse(рис. 3.23):

```
using System;
using El = SOLID_DLL.Ellipse;
namespace Test_Ellipse
{
  class Main_T_Ellipse
  {
    static void Main(string[] args)
    {
      double a = 2.0, b = 1.0, u = 0.0, v = 0.0;
      El ellipse = El.New(a, b, u, v, "test");
      ellipse.ToCalc(Ro, 1.0E-3);
      Console.WriteLine(ellipse.ToPrint());
    }
    static double Ro(double x, double y) => 4.0 / Math.PI;
  }
}
```

Рисунок 3.23 – Консольний додаток Test_Ellipse

Виконання програми дає наступні результати(рис. 3.24)(рис..3.25):

```
C:\Windows\system32\cmd.exe
Ellipse: test a =   2,00 b =   1,00 u =   4,00 v =   3,00

 Min_X =   1,00  Max_X =   7,00
 Min_Y =   1,00  Max_Y =   5,00

 Massa =   7,99994  Square =    6,28314
[ X =   4,00 ; Y =   3,00 ; Z =   0,00 ]  <Center>  Length =   5,00

 J_Oz =   210,00014  J_Cz =      9,99811
 J_Ox =    74,00012  J_Cx =      1,99914
 J_Oy =   136,00013  J_Cy =      7,99908
 J_xy =    96,00014 J_Cxy =     -0,00091

Для продолжения нажмите любую клавишу . . .
```

Рисунок 3.24 - Результат першого варіанту

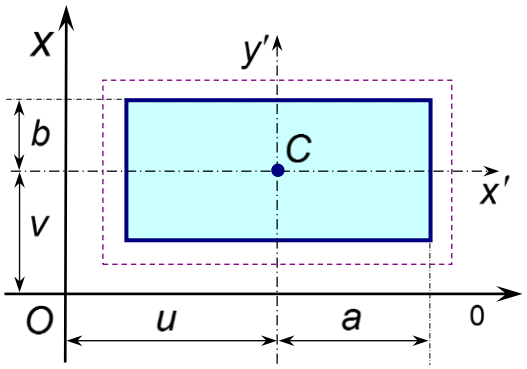
```
C:\Windows\system32\cmd.exe
Ellipse: test a =   2,00 b =   1,00 u =   0,00 v =   0,00

 Min_X =  -3,00  Max_X =   3,00
 Min_Y =  -2,00  Max_Y =   2,00

 Massa =   7,99994  Square =    6,28314
[ X =   0,00 ; Y =   0,00 ; Z =   0,00 ]  <Center>  Length =   0,00

 J_Oz =    10,00009  J_Cz =     10,00009
 J_Ox =     2,00018  J_Cx =      2,00018
 J_Oy =     8,00227  J_Cy =      8,00227
 J_xy =     0,00000 J_Cxy =      0,00000

Для продолжения нажмите любую клавишу . . .
```

Рисунок 3.25 - Результат другого варіанту

Результати обчислень для фігури еліпс можна вважати більш ніж задовільними, оскільки відмінність наближених значень параметрів (моментів інерції) по відношенню до їх точних значень складає менше 1%.

3.6 Клас Rectangle

Цей клас призначений задля математичного завдання фігури в формі прямокутника, для якого задаються координати (u, v) його центру C та довжини його сторін $2a$ та $2b$.



Сторони цього прямокутника мають бути паралельними до відповідних координатних осей.(рис. 3.26)

```
namespace SOLID_DLL
{
  public class Rectangle : Figure_2D
  {
    public readonly double a, b, u, v, S_Rectangle;
    public readonly string Name = "Rectangle: ";

    #region    < конструктори >
    public static
      Rectangle New(double ax, double by, double ux, double vy, string n)
    {
      if (ax < 1.0) return null; if (by < 0.5) return null;
      return new Rectangle(ax, by, ux, vy, n);
    }
    Rectangle(double ax, double by, double ux, double vy, string name)
    {
      a = ax; b = by; u = ux; v = vy;
      S_Rectangle = 4.0 * a * b; MinMax();
      if (name != "") Name += name;
    }
    #endregion < конструктори >

    public override void MinMax()...
    public override bool Flag(double x, double y)...
    public override string ToPrint()...
  }
}
```

Рисунок 3.26 – Клас Rectangle

Площа такої фігури обчислюється за формулою $S = 4ab$.

Оскільки сторони прямокутника є паралельними до відповідних координатних осей, ми можемо сформулювати достатньо прості умови для того, аби визначати для довільної точки із координатами (x, y), чи потрапляє вона «всередину» прямокутника або на її сторону, чи вона знаходиться поза межами прямокутника.

«*Якщо для довільної точки із координатами* (x, y) *одночасно виконуються нерівності:* $(x < u - a$ *або* $x > u + a)$ *або* $(y < v - b$ *або* $y > v + b)$*, то ця точка не потрапляє в задану фігуру – прямокутник*»(рис. 3.27):

```
public override bool Flag(double x, double y)
{
  bool fx = (x < u - a) || (x > u + a);
  bool fy = (y < v - b) || (y > v + b);
  return fx || fy;
}
```

Рисунок 3.27 - Метод Flag

Координати допоміжного прямокутника, який повністю «покриває» задану фігуру, обчислюються у відповідності до наступних формул:

$$Min_x = u - a - 1; \quad Max_x = u + a + 1;$$
$$Min_y = v - b - 1; \quad Max_y = v + b + 1.$$

Відповідна імплементація абстрактного метода MinMax() буде наступною(рис. 3.28):

```
public override void MinMax()
{
  Min_X = u - a - 1.0; Max_X = u + a + 1.0;
  Min_Y = v - b - 1.0; Max_Y = v + b + 1.0;
}
```

Рисунок 3.28 – Метод MinMax

Перевантажений віртуальний метод ToPrint() матиме код(рис.3.29):

```
public override string ToPrint()
{
  string txt = Name;
  txt += $" a ={a,7:F2} b ={b,7:F2} u ={u,7:F2} v ={v,7:F2} \r\n\r\n";
  return txt + Result;
}
```

Рисунок 3.29 - Метод ToPrint

Для екземплярів даного класу було прийнято штучне обмеження на довжини сторін прямокутника, а саме – кожна з сторін прямокутника не повинна бути меншою за 0.5.

Задля підтвердження коректності фізичного та програмного моделювання фігур в формі прямокутника був створений відповідний проект Test_Rectangle консольного додатку.

В цьому додатку перевірялися обчислення всіх шуканих фізичних характеристик для тора із заданими параметрами:

$$a = 2.0; \quad b = 1.0; \quad u = 4.0; \quad v = 3.0; \quad \rho_o = 3.0;$$

або $$a = 2.0; \quad b = 1.0; \quad u = -4.0; \quad v = -3.0; \quad \rho_o = 3.0.$$

З курсу теоретичної механіки відомо, що осьовий момент інерції прямокутника по відношенню до осі Cz', яка проходить через його центр C ортогонально його площині, буде дорівнювати: $J_{Cz'} = \frac{1}{3}m(a^2 + b^2)$.

Інші два моменти інерції, відносно осей Cx' та Cy' будуть відповідно: $J_{Cx'} = \frac{1}{3}mb^2$ та $J_{Cy'} = \frac{1}{3}ma^2$.

Тому, для заданого тора ми повинні отримати наступні результати:

$S = 4ab = 8.00000, \quad m = \rho_o S = 24.00000,$

$x_C = +4.00000, \quad y_C = +3.00000, \quad OC = 5.00000,$ або

$x_C = -4.00000, \quad y_C = -3.00000, \quad OC = 5.00000,$

$J_{Cz'} = 40.00000, J_{Oz} = J_{Cz'} + m \cdot OC^2 = 640.00000,$

$J_{Cx'} = 8.00000, J_{Ox} = J_{Cx'} + m \cdot y_C^2 = 224.00000,$

$$J_{Cy'} = 32.00000, J_{Oy} = J_{Cy'} + m \cdot x_C^2 = 416.00000,$$

$$J_{Cx'y'} = 0.00000, J_{xy} = m \cdot x_C \cdot y_C = 288.00000.$$

Нижче наводиться код консольного додатку Test_Rectangle(рис. 3.30):

```
using System;
using Rg = SOLID_DLL.Rectangle;
namespace Test_Rectangle
{
  class Main_T_Rectangle
  {
    static void Main(string[] args)
    {
      double a = 2.0, b = 1.0, u = 4.0, v = 3.0;
      Rg rectangle = Rg.New(a, b, u, v, "test");
      rectangle.ToCalc(Ro, 1.0E-3);
      Console.WriteLine(rectangle.ToPrint());
    }
    static double Ro(double x, double y) => 3.0;
  }
}
```

Рисунок 3.30 – Консольний додаток Test_Rectangle

Виконання програми дає наступні результати(рис. 3.31)(рис. 3.32):

```
C:\Windows\system32\cmd.exe
Rectangle: test a =   2,00 b =   1,00 u =   4,00 v =   3,00

 Min_X =   1,00  Max_X =   7,00
 Min_Y =   1,00  Max_Y =   5,00

 Massa =  24,00000  Square =    8,00000
[ X =   4,00 ; Y =   3,00 ; Z =   0,00 ]  <Center>  Length =   5,00

 J_Oz =   640,00000  J_Cz =     40,00000
 J_Ox =   224,00000  J_Cx =      8,00000
 J_Oy =   416,00000  J_Cy =     32,00000
 J_xy =   288,00000 J_Cxy =      0,00000

Для продолжения нажмите любую клавишу . . .
```

Рисунок 3.31 - Результат першого варіанту

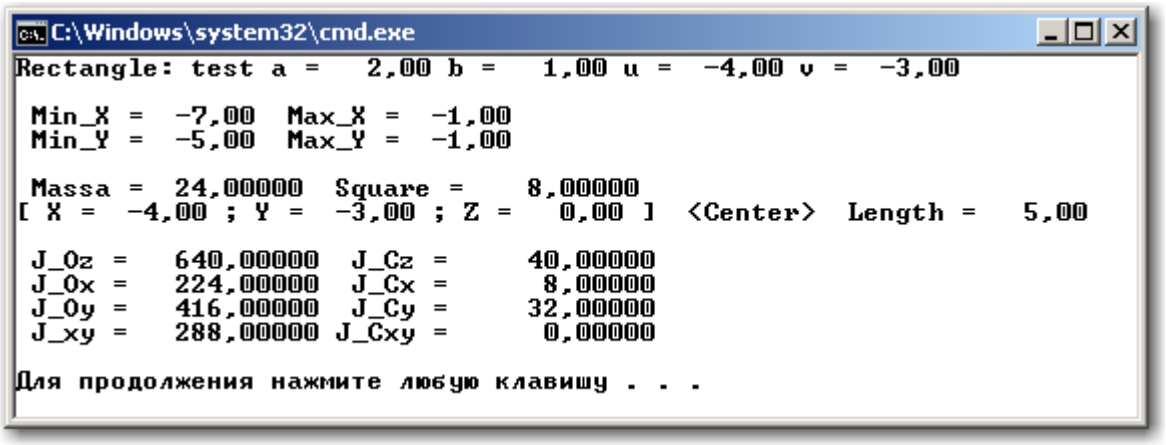
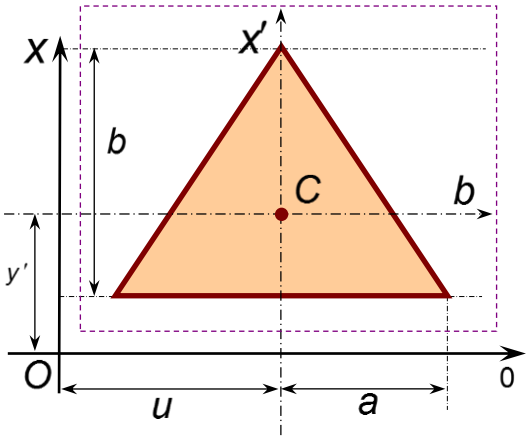


Рисунок 3.32 - Результат другого варіанту

Результати обчислень для фігури прямокутник можна вважати більш ніж задовільними, оскільки похибка обчислень відсутня як така.

Цей факт можна пояснити тим, що дана фігура має прямолінійні границі, які орієнтовані саме за осями обраної системи координат.

3.7 Клас Isosceles_Triangle



Цей клас призначений задля математичного завдання фігури в формі рівнобічного трикутника, який має основу $2a$ та висоту b, та задаються координати (u, v) його центру C.

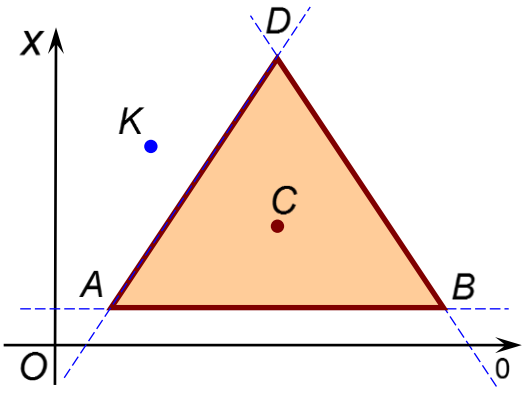
Центр C лежить на перетині медіан трикутника та поділяє всі медіани на відрізки у відношенні $2 \div 1$. Висота рівнобічного трикутника b, яка опущена на її основу $2a$, одночасно і є медіаною. Основа даного трикутника повинна бути паралельною до осі Ox, як то зображено на малюнку(рис. 3.33).

```
namespace SOLID_DLL
{
  public class Isosceles_Triangle : Figure_2D
  {
    public readonly double a, b, u, v, S_Isosceles_Triangle;
    public readonly string Name = "Isosceles_Triangle: ";
    double Ax, Ay, Bx, By, Dx, Dy, DAy, DAx, DBy, DBx;

    #region    < конструктори >
    public static Isosceles_Triangle
      New(double ax, double by, double ux, double vy, string n)
    {
      if (ax < 1.0) return null; if (by < 1.0) return null;
      return new Isosceles_Triangle(ax, by, ux, vy, n);
    }
    Isosceles_Triangle
      (double ax, double by, double ux, double vy, string name)
    {
      a = ax; b = by; u = ux; v = vy;
      S_Isosceles_Triangle = a * b; MinMax();
      if (name != "") Name += name;
    }
    #endregion < конструктори >

    public override void MinMax()...

    public override bool Flag(double x, double y)...

    public override string ToPrint()...
  }
}
```

Рисунок 3.33 – Клас Isosceles_Triangle

Площа такої фігури обчислюється за формулою $S = ab$.

Для довільної точки із координатами $K(x, y)$ ми повинні з'ясувати, чи потрапляє вона «всередину» даної фігури або на її сторону, чи вона взагалі знаходиться поза її межами.



Розв'язування даного питання потребує додаткових математичних дій та міркувань. Позначимо вершини рівнобічного трикутника літерами ABD так, як це зображено на малюнку.

Нам відомі координати цих вершин:

$$A = \left(u - a; \quad v - \frac{1}{3}b\right), B = \left(u + a; \quad v - \frac{1}{3}b\right), D = \left(u; \quad v + \frac{2}{3}b\right).$$

Отже, ми можемо отримати рівняння прямих, що проходять через вершини цього трикутника:

AD: $\frac{y-A_y}{D_y-A_y} - \frac{x-A_x}{D_x-A_x} = 0$, BD: $\frac{y-B_y}{D_y-B_y} - \frac{x-B_x}{D_x-B_x} = 0$,

AB: $y - \left(v - \frac{b}{3}\right) = 0$, та сформувати наступну сукупність шуканих умов:

$$\begin{bmatrix} \frac{y-A_y}{D_y-A_y} - \frac{x-A_x}{D_x-A_x} > 0, & \frac{y-B_y}{D_y-B_y} - \frac{x-B_x}{D_x-B_x} > 0, \\ y < \left(v - \frac{b}{3}\right). \end{bmatrix}$$

Ці умови реалізовані у відповідному перевантаженому методі(рис. 3.34):

```
public override bool Flag(double x, double y)
{
    bool f1 = ((y - Ay) / DAy - (x - Ax) / DAx) > 0;
    bool f2 = ((y - By) / DBy - (x - Bx) / DBx) > 0;
    bool f3 = (y < Ay);
    return f1 || f2 || f3;
}
```

Рисунок 3.34 – Метод Flag

Координати допоміжного прямокутника, який повністю «покриває» задану фігуру, обчислюються у відповідності до наступних формул:

$$Min_x = A_x - 1; \quad Max_x = B_x + 1;$$
$$Min_y = A_y - 1; \quad Max_y = D_y + 1.$$

Відповідна імплементація абстрактного метода MinMax() буде наступною(рис. 3.35) (тут також обчислюються допоміжні параметри, які використовуються в методі Flag()):

```
public override void MinMax()
{
  Ax = u - a; Ay = v - b / 3; Dx = u;
  Bx = u + a; By = v - b / 3; Dy = v + 2 * b / 3;
  DAy = Dy - Ay; DAx = Dx - Ax; DBy = Dy - By; DBx = Dx - Bx;
  Min_X = Ax - 1.0; Max_X = Bx + 1.0;
  Min_Y = Ay - 1.0; Max_Y = Dy + 1.0;
}
```

Рисунок 3.35 – Метод MinMax

Перевантажений віртуальний метод ToPrint() матиме код(рис. 3.36):

```
public override string ToPrint()
{
  string txt = Name;
  txt += $" a ={a,7:F2} b ={b,7:F2} u ={u,7:F2} v ={v,7:F2} \r\n\r\n";
  return txt + Result;
}
```

Рисунок 3.36 – Метод ToPrint

Для екземплярів даного класу було прийнято штучне обмеження на довжини сторін, а саме – основа не може бути меншою за 2.0, а висота не повинна бути меншою за 1.0.

Задля підтвердження коректності фізичного та програмного моделювання фігур в формі рівнобічного трикутника був створений відповідний проект Test_Isosceles_Triangle консольного додатку.

В цьому додатку перевірялися обчислення всіх шуканих фізичних характеристик для трикутника із заданими параметрами:

$$a = 1.0; \quad b = 3.0; \quad u = 0.0; \quad v = 0.0; \quad \rho_o = 6.0;$$

або $$a = 1.0; \quad b = 3.0; \quad u = 1.0; \quad v = 1.0; \quad \rho_o = 6.0.$$

З курсу теоретичної механіки відомо, що осьовий момент інерції рівнобічного трикутника по відношенню до осі Cz', яка проходить через його центр C ортогонально його площині, буде дорівнювати: $J_{Cz'} = \frac{1}{18} m(3a^2 + b^2)$.

Інші два моменти інерції, відносно осей Cx' та Cy' будуть відповідно: $J_{Cx'} = \frac{1}{18}mb^2$ та $J_{Cy'} = \frac{1}{6}ma^2$.

Тому, для заданого трикутника ми повинні отримати наступні результати:

$$S = ab = 3.00000, \quad m = \rho_o S = 18.00000,$$

$$x_C = +4.00000, \quad y_C = +3.00000, \quad OC = 5.00000, \text{ або}$$

$$x_C = -4.00000, \quad y_C = -3.00000, \quad OC = 5.00000,$$

$$J_{Cz'} = 12.00000, J_{Oz} = J_{Cz'} + m \cdot OC^2 = 48.00000,$$

$$J_{Cx'} = 9.00000, J_{Ox} = J_{Cx'} + m \cdot y_C^2 = 27.00000,$$

$$J_{Cy'} = 3.00000, J_{Oy} = J_{Cy'} + m \cdot x_C^2 = 21.00000,$$

$$J_{Cx'y'} = 0.00000, J_{xy} = m \cdot x_C \cdot y_C = 18.00000.$$

Код консольного додатку Test_Isosceles_Rectangle(рис. 3.37):

```
using System;
using IT = SOLID_DLL.Isosceles_Triangle;
namespace Test_Isosceles_Triangle
{
  class Main_T_Isosceles_Triangle
  {
    static void Main(string[] args)
    {
      double ax = 1.0, by = 3.0, ux = 1.0, vy = 1.0;
      IT i_triangle = IT.New(ax, by, ux, vy, "test");
      i_triangle.ToCalc(Ro, 1.0E-2);
      Console.WriteLine(i_triangle.ToPrint());
    }
    static double Ro(double x, double y) => 6.0;
  }
}
```

Рисунок 3.37 – Консольний додаток Test_Isosceles_Rectangle

Виконання програми дає наступні результати(рис. 3.38)(рис. 3.39) :

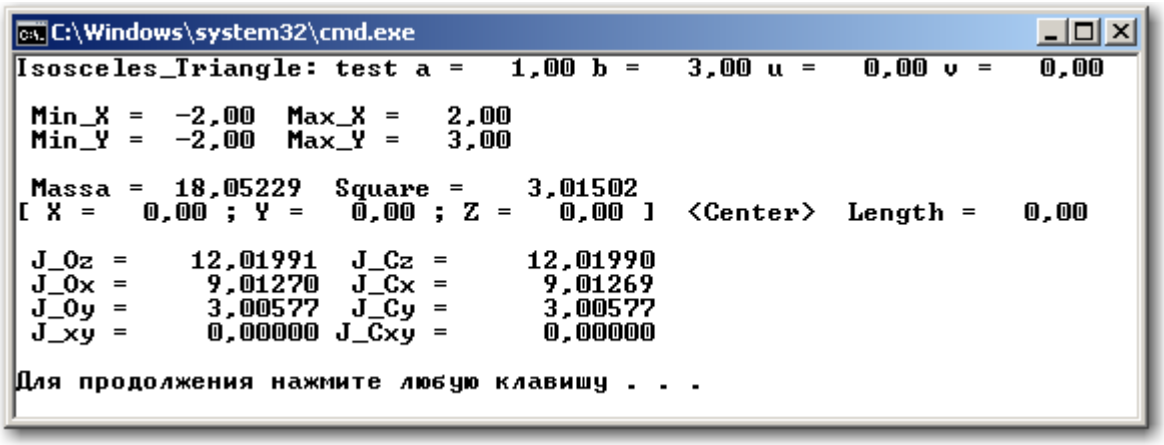


Рисунок 3.38 - Результат першого варіанту

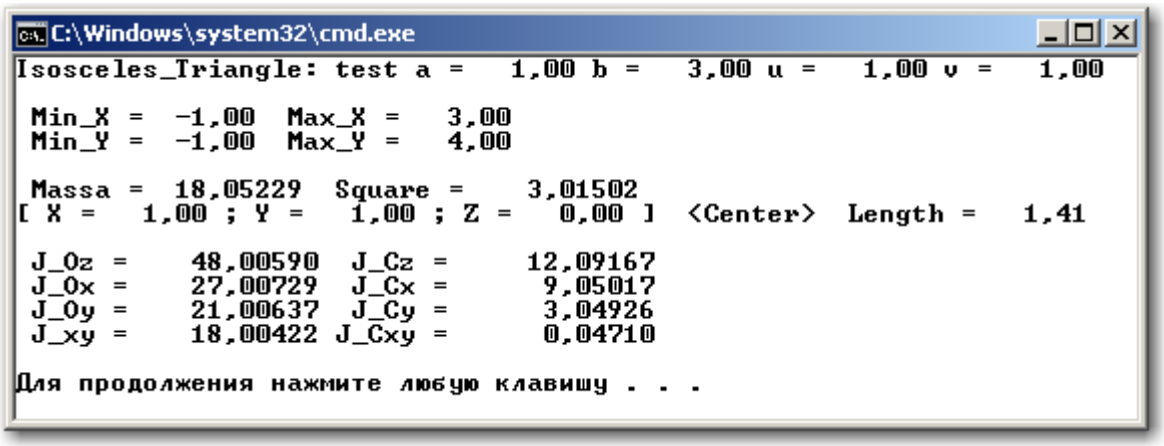


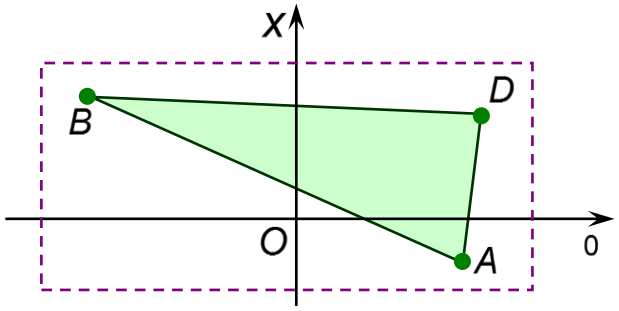
Рисунок 3.39 - Результат другого варіанту

Результати обчислень для фігури рівнобічний трикутник можна вважати гіршими ніж у попередніх випадках.

Цей факт можна пояснити тим, що дана фігура має кути між сторонами, які є гострими, та погано апроксимуються квадратурною формулою, що використовується в даних обчисленнях.

3.8 Клас Triangle

Цей клас призначений задля математичного завдання фігури в формі трикутника, для якого задаються декартові координати



трьох його вершин A, B, D.(рис. 3.40)

```
namespace SOLID_DLL
{
  public class Triangle : Figure_2D
  {
    public readonly Point_3D A, B, D;
    public readonly double S_ABD;
    public readonly string Name = "Triangle: ";

    #region    < конструктори >
    public static
      Triangle New(Point_3D a, Point_3D b, Point_3D d, string n)
    {
      if (Area(a, b, d) < 1.0E-5) return null;
      return new Triangle(a, b, d, n);
    }
    Triangle(Point_3D a, Point_3D b, Point_3D d, string name)
    {
      A = new Point_3D(a.X, a.Y, a.Z, a.Name);
      B = new Point_3D(b.X, b.Y, b.Z, b.Name);
      D = new Point_3D(d.X, d.Y, d.Z, d.Name);
      S_ABD = Area(A, B, D); MinMax();
      if (name != "") Name += name;
    }
    #endregion < конструктори >

    public override bool Flag(double x, double y)...
    public override void MinMax()...
    public override string ToPrint()...

    static double Area(Point_3D A, Point_3D B, Point_3D D)...
  }
}
```

Рисунок 3.40 – Клас Triangle

Основним допоміжним методом к цьому класі є метод, який обчислює площу трикутника через формулу векторного добутку двох векторів.

Якщо нам задані два вектори $\vec{a}$ і $\vec{b}$, які мають спільну основу, то векторний добуток цих векторів дасть нам третій вектор $\vec{c} = \vec{a} \times \vec{b}$, який буде перпендикулярним до площини векторів $\vec{a}$ і $\vec{b}$, буде мати ту саму основу, та буде утворювати із ними праву трійку векторів.

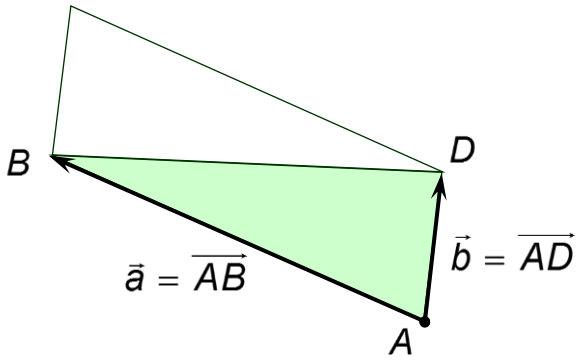
Вектор $\vec{c}$ можна обчислювати через проекції векторів $\vec{a}$ і $\vec{b}$ за допомогою визначника:

$$\vec{c} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} = (a_y b_z - a_z b_y)\vec{i} + (a_z b_x - a_x b_z)\vec{j} + (a_x b_y - a_y b_x)\vec{k},$$

$$c_x = a_y b_z - a_z b_y, \quad c_y = a_z b_x - a_x b_z, \quad c_z = a_x b_y - a_y b_x,$$

$$|\vec{c}| = \sqrt{c_x^2 + c_y^2 + c_z^2}.$$

Модуль вектора $\vec{c}$ дорівнює площі паралелограма, побудованого на векторах $\vec{a}$ і $\vec{b}$.



Отже, якщо в якості вектора $\vec{a}$ взяти вектор $\overrightarrow{AB}$, а в якості вектора $\vec{b}$ взяти вектор $\overrightarrow{AD}$, то площа трикутника ABD буде дорівнювати половині модуля вектора $\vec{c} = \overrightarrow{AB} \times \overrightarrow{AD}$. Тобто $S_{ABD} = \frac{1}{2}|\vec{c}|$.

Складові вказаних векторів $\vec{a} = \overrightarrow{AB}$ і $\vec{b} = \overrightarrow{AD}$ можна обчислити через координати вершин трикутника ABD, тобто:

$$\overrightarrow{AB} = (B_x - A_x, B_y - A_y, B_z - A_z)$$

та

$$\overrightarrow{AD} = (D_x - A_x, D_y - A_y, D_z - A_z).$$

Якщо модуль вектора $\vec{c}$ буде дорівнювати нулю, то ми можемо зробити висновок, що вектори $\overrightarrow{AB}$ і $\overrightarrow{AD}$ є колінеарними, а отже точки A, B, D лежать на одній прямій та взагалі не утворюють трикутник ABD.

Цю властивість ми будемо використовувати в конструкторі в якості умови створення екземпляра даного класу.

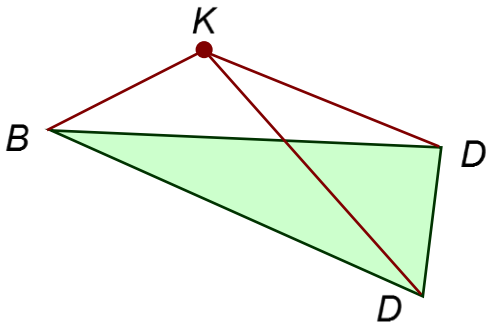
Якщо задані три точки A, B, D будуть лежати на спільній прямій – об’єкт трикутник ABD створюватись не буде.

Сказане вище реалізоване у вигляді відповідного статичного метода Area(), який обчислює за координатами трьох заданих точок площу трикутника, що утворюють ці три точки(рис. 3.41).

```
static double Area(Point_3D A, Point_3D B, Point_3D D)
{
  double ax = B.X - A.X, ay = B.Y - A.Y, az = B.Z - A.Z;
  double bx = D.X - A.X, by = D.Y - A.Y, bz = D.Z - A.Z;
  double cx = ay * bz - az * by;
  double cy = az * bx - ax * bz;
  double cz = ax * by - ay * bx;
  return 0.5 * Math.Sqrt(cx * cx + cy * cy + cz * cz);
}
```

Рисунок 3.41 – Метод Area

Математична умова для перевантаженого метода Flag() будується на основі наступних міркувань.



«Якщо будь-яка точка $K(x, y)$ *не лежить в площині заданого трикутника* ABD*, то виконується нерівність* $S_{ABK} + S_{ADK} + S_{BDK} > S_{ABD}$»(рис. 3.42):

```
public override bool Flag(double x, double y)
{
  Point_3D K = new Point_3D(x, y, 0.0, "Dummy");
  return S_ABD < (Area(A, B, K) + Area(A, D, K) + Area(B, D, K));
}
```

Рисунок 3.42 – Метод Flag

Координати прямокутника, який повністю покриває заданий трикутник ABC, також обчислюються через координати вершин A, B, D трикутника:

$$Min_x = Min\{A_x, B_x, D_x\} - 1; \quad Max_x = Max\{A_x, B_x, D_x\} + 1;$$
$$Min_y = Min\{A_y, B_y, D_y\} - 1; \quad Max_y = Max\{A_y, B_y, D_y\} + 1.$$

Вказані обчислення реалізовані в перевантаженому методі MinMax() :

```
public override void MinMax()
{
  Min_X = Math.Min(Math.Min(A.X, B.X), D.X) - 1.0;
  Min_Y = Math.Min(Math.Min(A.Y, B.Y), D.Y) - 1.0;
  Max_X = Math.Max(Math.Max(A.X, B.X), D.X) + 1.0;
  Max_Y = Math.Max(Math.Max(A.Y, B.Y), D.Y) + 1.0;
}
```

Рисунок 3.43 – Метод MinMax

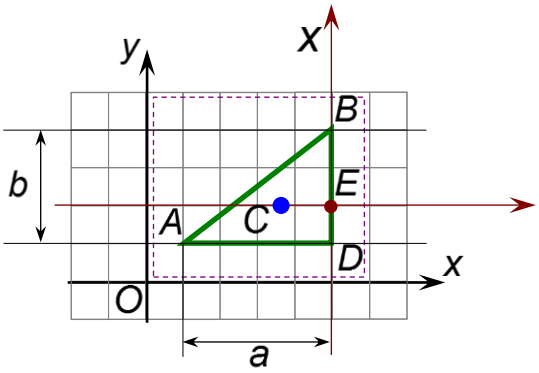
Перевантажений віртуальний метод ToPrint() матиме наступний код(рис. 3.44):

```
public override string ToPrint()
{
  string txt = Name + "\r\n";
  txt += A.ToPrint() + B.ToPrint() + D.ToPrint() + "\r\n";
  return txt + Result;
}
```

Рисунок 3.44 – Метод ToPrint

Задля підтвердження коректності фізичного та програмного моделювання фігур в формі трикутника був створений відповідний проект Test_Triangle консольного додатку.

В цьому додатку здійснювалися обчислення всіх шуканих фізичних характеристик для однорідного єгипетського трикутника (прямокутний трикутник із катетами $AD = a = 4$ та $BD = b = 3$, та гіпотенузою $AB = 5$, $ED = \frac{b}{3}$) із заданими вершинами (дивись малюнок):



$A = (1.0,\ 1.0,\ 0.0);\quad B = (5.0,\ 4.0,\ 0.0);\quad D = (5.0,\ 1.0,\ 0.0);\quad \rho_o = 2\ .$

З курсу теоретичної механіки нам відомі наступні формули для осьових моментів інерції прямокутного трикутника із катетами a і b по відношенню до осей $Exyz$, які проведені із точки E цього трикутника, буде дорівнювати:

$$J_{Ex} = \frac{1}{18}mb^2, J_{Ey} = \frac{1}{6}ma^2 \text{ та } J_{Ez} = \frac{1}{18}m(3a^2 + b^2),$$

$$x_C = x_A + \frac{2a}{3} \approx 3.66667,\quad y_C = y_A + \frac{b}{3} = 2.00000,$$

Тому, для заданого прямокутного трикутника ми повинні отримати наступні результати:

$$S = \frac{1}{2}ab = 6.00000,\quad m = \rho_o S = 12.00000, CE = 1.33333,$$

$x_E = +5, \quad y_E = +2, \quad OE = 5.38517, \; OC \approx 4.17665.$

де E – центр осей $Exyz$.

$$J_{Ex} = \frac{1}{18} mb^2 = 6.00000, J_{Cx} = J_{Ex} = 6.00000,$$

$$J_{Ox} = J_{Ex} + m \cdot E_y^2 = 54.00000,$$

$$J_{Ey} = \frac{1}{6} ma^2 = 32.00000, J_{Cy} = J_{Ey} + m \cdot CE^2 = 10.66667,$$

$$J_{Oy} = J_{Ey} + m \cdot (x_C^2 - CE^2) = 172.00000,$$

$$J_{Ez} = \frac{1}{18} m(3a^2 + b^2) = 38.00000, J_{Cz} = J_{Ez} - m \cdot CE^2 = 16.66667,$$

$$J_{Oz} = J_{Ez} + m \cdot (OC^2 - CE^2) = 226.00000.$$

Нижче наводиться код консольного додатку Test_Triangle:

```
using System;
using TR = SOLID_DLL.Triangle;
using P0 = SOLID_DLL.Point_3D;
namespace Test_Triangle
{
  class Main_T_Triangle
  {
    static void Main(string[] args)
    {
      P0 A = new P0(1.0, 1.0, 0.0, "A");
      P0 B = new P0(5.0, 4.0, 0.0, "B");
      P0 D = new P0(5.0, 1.0, 0.0, "C");
      TR ABD = TR.New(A, B, D, "ABD");
      ABD.ToCalc(ABD_density, 1.0E-3);
      Console.WriteLine(ABD.ToPrint());
    }
    static double ABD_density(double x, double y) => 2.0;
  }
}
```

Рисунок 3.45 – Консольний додаток Test_Triangle

Виконання даної програми дає наступні результати(рис. 3.46):

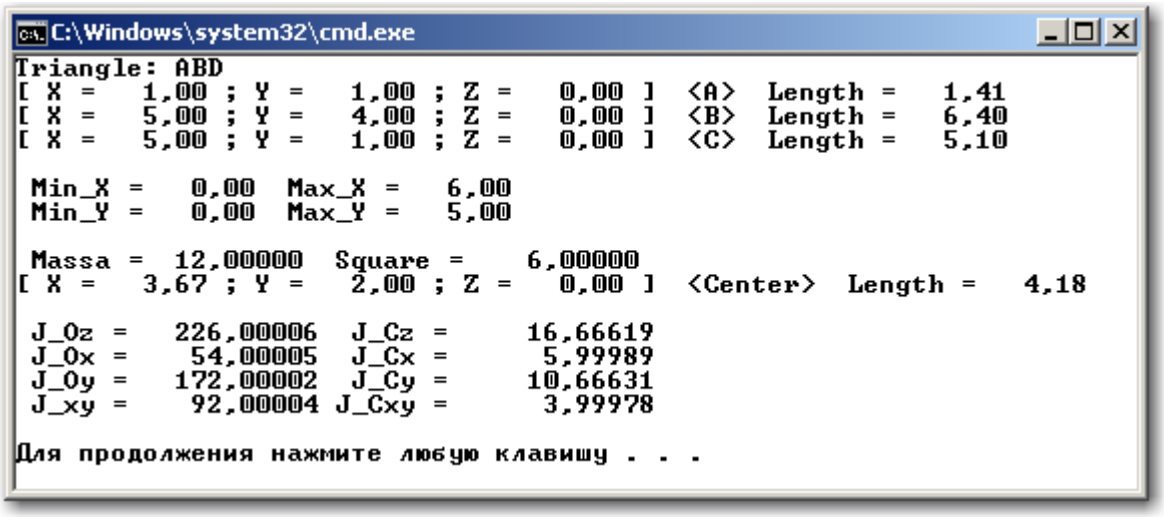


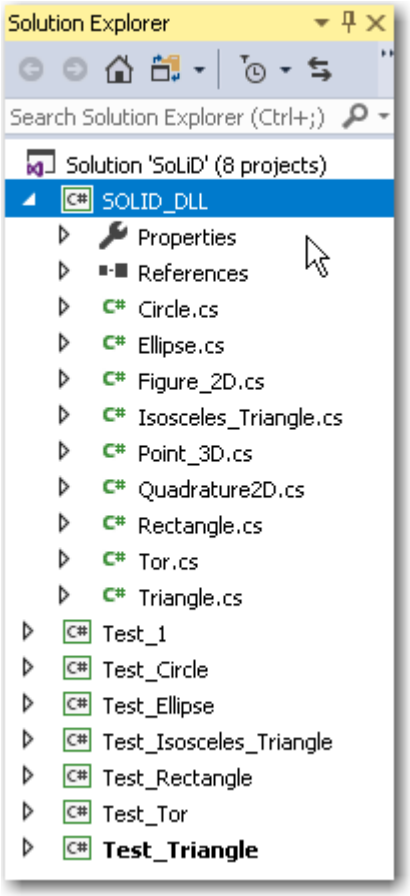
Рисунок 3.46 - Результат роботи програми

Співставлення числових даних, отриманих програмою та за допомогою калькулятора (тобто за формулами), говорить на користь того, що числовий алгоритм працює коректно та забезпечує задану похибку обчислень.

На цьому можна вважати етап тестування класів геометричних фігур в цілому завершеним.

Результат можна вважати позитивним.

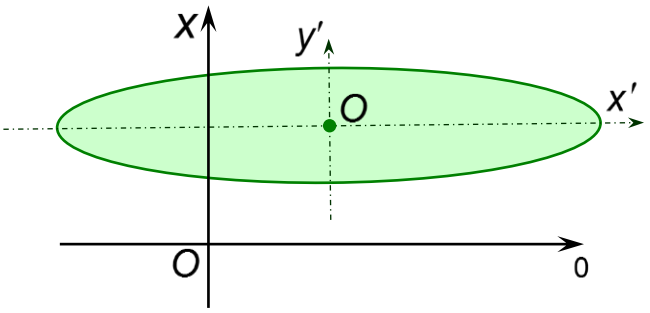
Дерево проектів за цією частиною роботи виглядає наступним чином:



РОЗДІЛ 4. ФІГУРИ. МОМЕНТИ ІНЕРЦІЇ

4.1 Загальні формули та позначення.

Нехай фізичне тіло, що досліджується, є плоскою фігурою, маса m якої розподілена по всіх точках цієї фігури, а площина фігури співпадає із координатною площиною Oxy.



Вісь Oz є перпендикулярною до площини малюнка.

Нас будуть цікавити наступні моменти першого та другого степенів:

$$x_C = \frac{1}{m}\iint_S(\rho(x,y)\cdot x)\,dS, \quad y_C = \frac{1}{m}\iint_S(\rho(x,y)\cdot y)\,dS, \quad m = \iint_S \rho(x,y)\,dS,$$

$$J_{Ox} = \iint_S(\rho(x,y)\cdot y^2)\,dS, \quad J_{Oz} = \iint_S\big(\rho(x,y)\cdot(x^2+y^2)\big)\,dS,$$
$$J_{Oy} = \iint_S(\rho(x,y)\cdot x^2)\,dS, \qquad J_{xy} = \iint_S\big(\rho(x,y)\cdot(xy)\big)\,dS,$$

де $\rho(x,y)$ – густина елемента фігури, $dS = dxdy$ – площа цього елемента.

Моменти інерції відносно осей $Cx'y'z'$, які є паралельними до системи координатних осей $Oxyz$, та проходять через центр мас $C(x_C, y_C, 0)$ даної фігури, обчислюються за відомою формулою Штейнера:

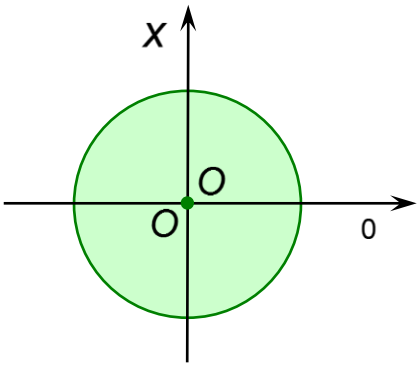
$$\begin{cases} J_{Ox} = J_{Cx'} + m\cdot y_C^2, \\ J_{Oy} = J_{Cy'} + m\cdot x_C^2, \\ J_{Oz} = J_{Cz'} + m\cdot OC^2, \\ J_{xy} = J_{Cx'y'} + m\cdot x_C y_C, \end{cases} \text{ звідки маємо } \begin{cases} J_{Cx'} = J_{Ox} - m\cdot y_C^2, \\ J_{Cy'} = J_{Oy} - m\cdot x_C^2, \\ J_{Cz'} = J_{Oz} - m\cdot OC^2, \\ J_{Cx'y'} = J_{xy} - m\cdot x_C y_C, \end{cases}$$

де $OC = \sqrt{x_C^2 + y_C^2}$ – відстань між центром O та центром мас C даної фігури.

Далі будуть розглянуті фігури, які вважаються однорідними за розподілом маси по площі фігури.

Тобто ми будемо вважати, що густина цих фігур є константою для кожної її точки $\rho(x,y) = \rho_o = \frac{m}{S}$, де S – її площа.

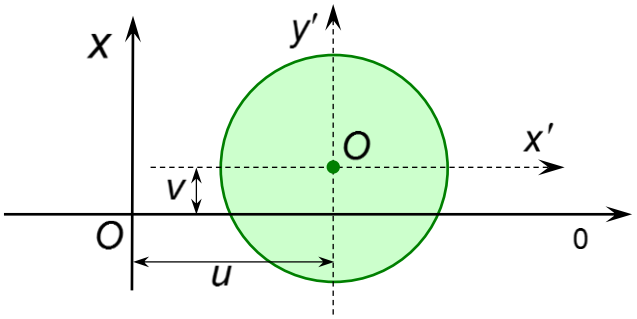
4.2 Фігура Круг



Нехай геометричний центр C однорідного круга, який має радіус R, співпадає із початком O системи координат Oxy.

Тоді ми маємо наступні формули обчислень для основних шуканих механічних характеристик такої фігури:

$$S = \pi R^2, \quad x_C = 0, \quad J_{Cx'} = \frac{1}{4}mR^2, \quad J_{Cy'} = \frac{1}{4}mR^2,$$
$$m = \rho_o S, \quad y_C = 0, \quad J_{Cz'} = \frac{1}{2}mR^2, \quad J_{x'y'} = 0.$$



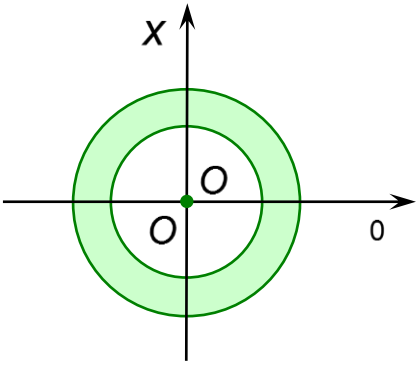
Якщо круг заданий умовою $(x-u)^2 + (y-v)^2 \le R^2$, то відповідні характеристики та моменти будуть дорівнювати:

$x_C = u, \quad y_C = v, OC = \sqrt{u^2 + v^2},$

$$J_{Ox} = \frac{1}{4}mR^2 + m \cdot v^2, \quad J_{Oz} = \frac{1}{2}mR^2 + m \cdot OC^2,$$
$$J_{Oy} = \frac{1}{4}mR^2 + m \cdot u^2, \quad J_{xy} = m \cdot uv.$$

За для програмного опису даних фігур створено відповідний клас **Circle**.

4.3 Фігура Тор

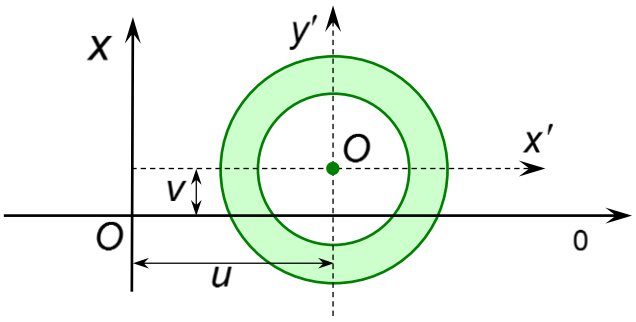


Нехай геометричний центр C однорідного тора, який має зовнішній радіус R та внутрішній радіус $0 < r < R$, співпадає із початком O системи координат Oxy.

Тоді ми маємо наступні формули обчислень для основних шуканих механічних характеристик такої фігури:

$$S = \pi(R^2 - r^2), \quad x_C = 0, \quad J_{Cx'} = \frac{1}{4}m(R^2 + r^2), \quad J_{Cy'} = \frac{1}{4}m(R^2 + r^2),$$
$$m = \rho_o S, \qquad y_C = 0, \quad J_{Cz'} = \frac{1}{2}m(R^2 + r^2), \qquad J_{x'y'} = 0.$$

Якщо тор заданий системою умов $\begin{cases}(x-u)^2 + (y-v)^2 \le R^2 \\ (x-u)^2 + (y-v)^2 \ge r^2\end{cases}$, то відповідні характеристики та моменти будуть дорівнювати:

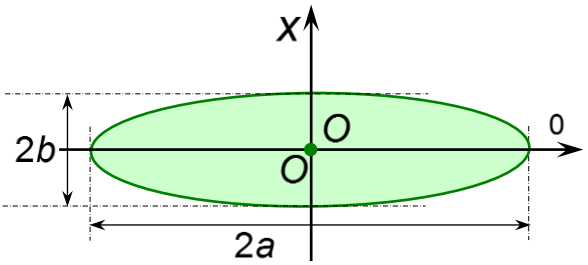


$x_C = u, \quad y_C = v, OC = \sqrt{u^2 + v^2},$

$$J_{Ox} = \frac{1}{4}m(R^2 + r^2) + m \cdot v^2, \quad J_{Oz} = \frac{1}{2}m(R^2 + r^2) + m \cdot OC^2,$$
$$J_{Oy} = \frac{1}{4}m(R^2 + r^2) + m \cdot u^2, \qquad J_{xy} = m \cdot uv.$$

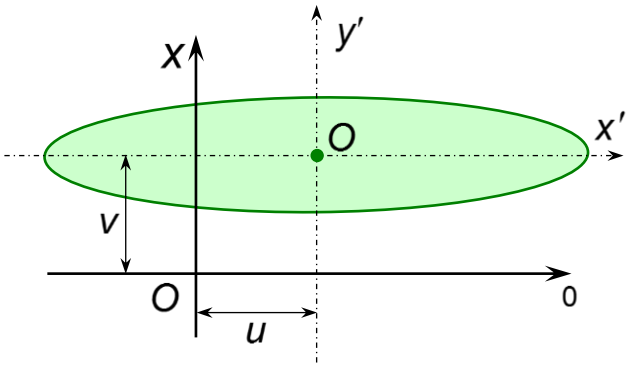
За для програмного опису даних фігур створено відповідний клас Tor.

4.4 Фігура Еліпс



Нехай геометричний центр C однорідного еліпса, який має півосі a і b відповідно, співпадає із початком O системи координат Oxy. Тоді ми маємо наступні формули обчислень для основних шуканих механічних характеристик такої фігури:

$$S = \pi ab, \quad x_C = 0, \qquad J_{Cx'} = \frac{1}{4}mb^2, \qquad J_{Cy'} = \frac{1}{4}ma^2,$$
$$m = \rho_o S, \quad y_C = 0, \quad J_{Cz'} = \frac{1}{4}m(a^2 + b^2), \qquad J_{x'y'} = 0.$$



Якщо еліпс буде заданий умовою $\frac{(x-u)^2}{a^2}+\frac{(y-v)^2}{b^2}\le 1$, то відповідні характеристики та моменти будуть дорівнювати:

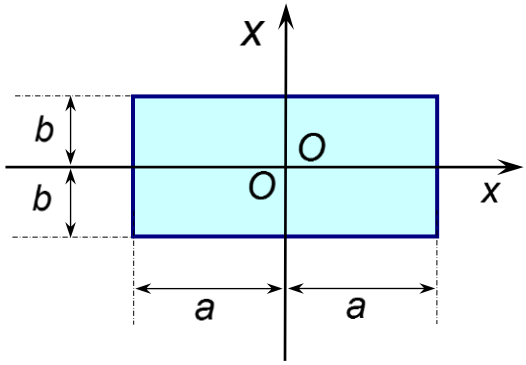
$$x_C = u, \quad y_C = v, OC = \sqrt{u^2+v^2},$$

$$J_{Ox} = \frac{1}{4}mb^2 + m\cdot v^2, \quad J_{Oz} = \frac{1}{4}m(a^2+b^2) + m\cdot OC^2,$$

$$J_{Oy} = \frac{1}{4}ma^2 + m\cdot u^2, \qquad J_{xy} = m\cdot uv.$$

За для програмного опису даних фігур створено відповідний клас Ellipse.

4.5 Фігура Прямокутник



Нехай геометричний центр C однорідного прямокутника, який має сторони $2a$ і $2b$, співпадає із початком O системи координат Oxy.

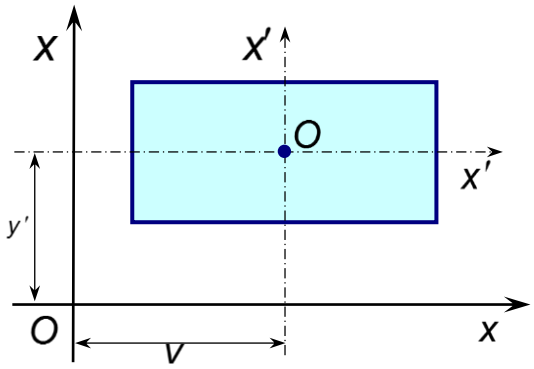
Сторони цього прямокутника мають бути паралельними до відповідних координатних осей, як то зображено на малюнку.

Тоді ми маємо наступні формули обчислень для основних шуканих механічних характеристик такої фігури:

$$S = 4ab, \quad x_C = 0, \qquad J_{Cx'} = \frac{1}{3}mb^2, \qquad J_{Cy'} = \frac{1}{3}ma^2,$$

$$m = \rho_o S, \quad y_C = 0, \quad J_{Cz'} = \frac{1}{3}m(a^2+b^2), \qquad J_{x'y'} = 0.$$

Якщо геометричний центр прямокутника заданий координатами $C(u, v)$, то відповідні характеристики та моменти будуть дорівнювати:



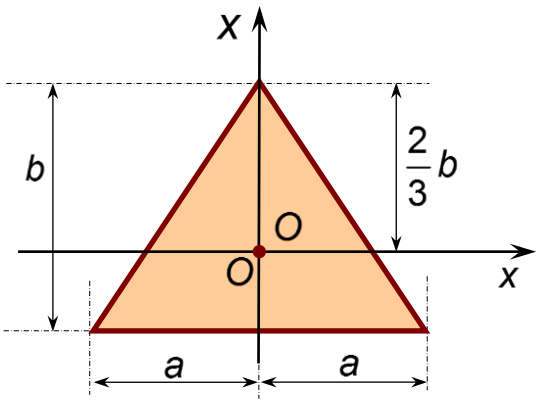
$$x_C = u, \quad y_C = v, OC = \sqrt{u^2 + v^2},$$

$$J_{Ox} = \frac{1}{3}mb^2 + m \cdot v^2, \quad J_{Oz} = \frac{1}{3}m(a^2 + b^2) + m \cdot OC^2,$$
$$J_{Oy} = \frac{1}{3}ma^2 + m \cdot u^2, \qquad J_{xy} = m \cdot uv.$$

За для програмного опису даних фігур створено відповідний клас Rectangle.

4.6 Фігура Рівнобічний трикутник

Нехай геометричний центр C однорідного рівнобічного трикутника, який має основу $2a$ та висоту b, співпадає із початком O системи координат Oxy.



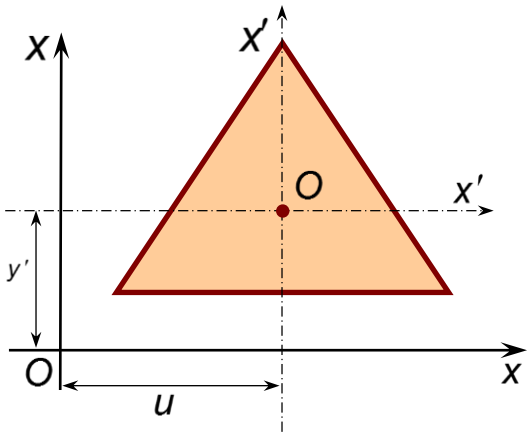
Цей центр C лежить на перетині медіан трикутника та поділяє всі медіани на відрізки у відношенні $2 \div 1$.

Висота рівнобічного трикутника, яка опущена на її основу, одночасно є медіаною. Основа даного трикутника повинна бути паралельною до осі Ox.

Тоді ми матимемо наступні формули обчислень для основних шуканих механічних характеристик такої фігури:

$$S = ab, \quad x_C = 0, \qquad J_{Cx'} = \frac{1}{18}mb^2, \qquad J_{Cy'} = \frac{1}{6}ma^2,$$
$$m = \rho_o S, \quad y_C = 0, \quad J_{Cz'} = \frac{1}{18}m(3a^2 + b^2), \qquad J_{x'y'} = 0.$$

Якщо геометричний центр даного трикутника заданий координатами $C(u, v)$, то відповідні характеристики та моменти будуть дорівнювати:



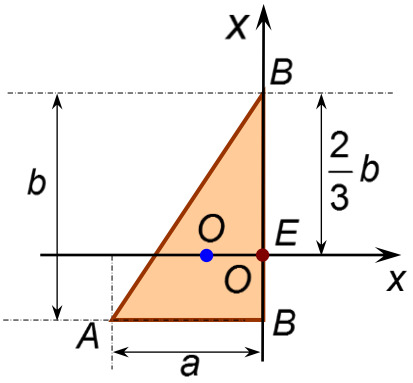
$$x_C = u, \quad y_C = v, OC = \sqrt{u^2 + v^2},$$

$$J_{Ox} = \frac{1}{18}mb^2 + m \cdot v^2, \quad J_{Oz} = \frac{1}{18}m(3a^2 + b^2) + m \cdot OC^2,$$
$$J_{Oy} = \frac{1}{6}ma^2 + m \cdot u^2, \qquad J_{xy} = m \cdot uv.$$

За для програмного опису даних фігур створено відповідний клас Isosceles_Triangle.

4.7 Фігура Прямокутний трикутник

Нехай точка E однорідного прямокутного трикутника, який має катети a і b, співпадає із початком O системи координат Oxy.

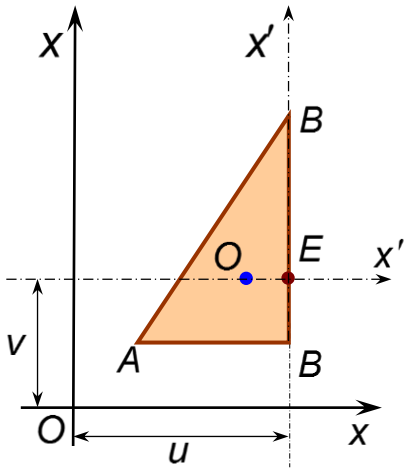


Центр C лежить на перетині медіан трикутника та поділяє всі медіани на відрізки у відношенні $2 \div 1$. На малюнку ми бачимо, що обидві точки E та C лежать на осі Ox.

Катет $AD = a$ повинен бути паралельним до осі Ox, а катет $BD = b$ лежить саме на осі Oy. Тоді $ED = \frac{b}{3}$, $CE = \frac{a}{3}$.

Маємо наступні формули обчислень для основних шуканих механічних характеристик такої фігури:

$$S = \frac{1}{2}ab, \quad x_C = -\frac{a}{3}, \qquad J_{Ex} = \frac{1}{18}mb^2, \qquad J_{Ey} = \frac{1}{6}ma^2,$$
$$m = \rho_o S, \quad y_C = 0, \quad J_{Ez} = \frac{1}{18}m(3a^2 + b^2), \quad J_{Exy} = 0.$$



Якщо точка E даного трикутника задана координатами $E(u, v)$, то відповідні характеристики та моменти будуть дорівнювати:

$OE = \sqrt{u^2 + v^2}$,

$x_C = u - \frac{a}{3}$, $y_C = v$, $OC = \sqrt{x_C^2 + y_C^2}$,

$J_{Ox} = J_{Ex'} + m \cdot v^2$, $J_{Oz} = J_{Ez'} + m \cdot (OC^2 - CE^2)$,

$J_{Oy} = J_{Ey'} + m \cdot (x_C^2 - CE^2)$.

РОЗДІЛ 5. АНАЛІЗ ТА ОБРОБКА ЦИФРОВИХ ФІГУР

Всі програмні компоненти були створені та налагоджені в середовищі розробки Microsoft Visual Studio® на мові C#.

Контейнером для проекту (Solution) був робочій простір DiplomBool. В цьому робочому просторі згенерований один проект:

1. Основний проект DiplomBool – проект Windows Forms Application, призначений для завантаження і обробки зображення, відображення їх з нанесеною декартовою системою координат, аналізу кольору пікселів і обчислення їх координати.
2. Основний класс MainForm - головна форма додатку, яка дозволяє завантажувати зображення та відображати його у новому вікні.
3. Додатковий клас ImageForm – після завантаження зображення, відображає його на новій формі, обробляє події кліків миші для визначення кольору пікселя та обчислення координат у декартовій системі.
4. Функціональний клас ImageWithShapes – завантажує зображення з файлу та визначає фоновий колір, обробляє піксельні дані для визначення кольору пікселів та їх відношення до фону, масштабує координати для відображення у декартовій системі

Загальна структура робочого простору(рис. 5.1):

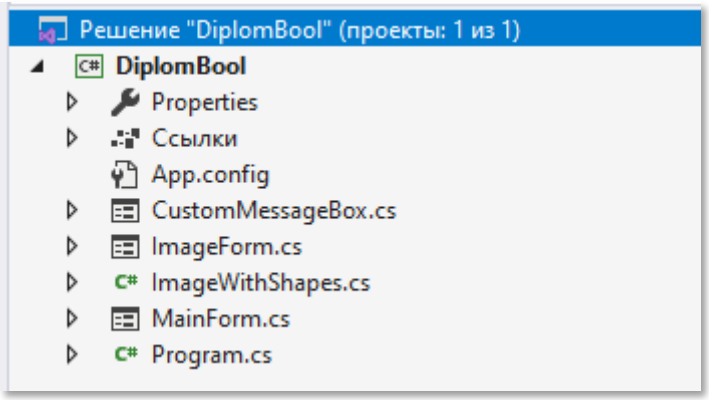


Рисунок 5.1 – Структура робочого простору

5.1 Клас ImageWithShapes

Програмна реалізація класу ImageWithShapes

Загальна структура класу(рис. 5.2):

```
1   using System;
2   using System.Collections.Generic;
3   using System.Drawing;
4   using System.Linq;
5
6   namespace DiplomBool
7   {
8       public class ImageWithShapes : IDisposable
9       {
10          public Bitmap Image { get; private set; }
11          public Color BackgroundColor { get; private set; }
12          public int Scale { get; private set; }
13
14          public ImageWithShapes(string filePath, int scale = 10)
15          {
16              Image = new Bitmap(filePath);
17              Scale = scale;
18              FindBackgroundColor();
19          }
20          Малювання декартової системи координат
32          Отримання кольору пікселів
42          Пошук кольору фону
73          Знаходження центру фігури
98
99          private Point ConvertToCartesian(Point point)...
103
104         public void Dispose()...
112     }
113 }
```

Рисунок 5.2 – Клас ImageWithShapes

Блок малювання декартової системи координат на зображенні відповідає за малювання декартової системи координат на зображенні(рис. 5.4).

1. Створюється об'єкт `Graphics`, який пов'язаний з зображенням `Image` за допомогою ключового слова `using`. Це забезпечує автоматичне вивільнення ресурсів після виконання коду в блоках `using`.

2. Створюється об'єкт `Pen`, який використовується для малювання ліній, з чорним кольором.

3. Малюється горизонтальна вісь декартової системи координат за допомогою методу `DrawLine` об'єкта `Graphics`. Ця лінія розташована внизу зображення та простягається від точки `(0, Image.Height - 1)` до точки `(Image.Width - 1, Image.Height - 1)`.

4. Малюється вертикальна вісь декартової системи координат за допомогою методу `DrawLine` об'єкта `Graphics`. Ця лінія розташована зліва зображення та простягається від точки `(0, 0)` до точки `(0, Image.Height)`.

Отже, цей блок створює декартову систему координат на зображенні, де горизонтальна вісь розташована внизу, а вертикальна - зліва.

Блок отримання кольору пікселя зображення за вказаними координатами точки(рис. 5.5) отримує колір пікселя на зображенні за вказаними координатами `point`.

1. Умова `if` перевіряє, чи знаходиться вказана точка `point` всередині меж зображення. Якщо хоча б одна з координат `X` або `Y` виходить за межі ширини або висоти зображення відповідно, генерується виняток `ArgumentOutOfRangeException` з повідомленням про те, що точка знаходиться поза межами зображення.

2. Якщо точка знаходиться в межах зображення, використовуючи метод `GetPixel` об'єкта `Image`, отримується колір пікселя на зображенні за вказаними координатами `point`.

3. Отриманий колір пікселя повертається з методу як результат роботи цієї функції.

Блок знаходження фонового кольору зображення та перевірки, чи є заданий колір фоновим(рис. 5.5) забезпечує функціональність для роботи зображення, включаючи знаходження фонового кольору, перевірку, чи є певний колір фоновим, і звільнення ресурсів після використання.

1. Метод FindBackgroundColor() виконує сканування кожного пікселя на зображенні для визначення фонового кольору. Шляхом ітерації через кожен піксель використовується словник для підрахунку кількості пікселів для кожного кольору на зображенні. Після завершення сканування обирається колір з найбільшою кількістю пікселів, який потім встановлюється як фоновий колір.

2. Метод IsBackgroundColor(Color color) призначений для перевірки, чи є заданий колір фоновим на зображенні. Він порівнює значення ARGB (альфа, червоний, зелений, синій) заданого кольору з фоновим кольором, який визначений методом FindBackgroundColor(). Якщо значення ARGB заданого коліру співпадає з фоновим, метод повертає `true`, вказуючи на те, що колір є фоновим. В протилежному випадку, метод повертає `false`.

3. Метод Dispose() використовується для звільнення ресурсів, пов'язаних з зображенням. Він перевіряє, чи існує зображення, і якщо так, то звільняє пам'ять, призначену для зображення, і призначає змінній `Image` значення `null`. Такий підхід дозволяє ефективно управляти пам'яттю і попереджує витоки ресурсів.

```
10          public Bitmap Image { get; private set; }
11          public Color BackgroundColor { get; private set; }
12          public int Scale { get; private set; }
13
14          public ImageWithShapes(string filePath, int scale = 10)
15          {
16              Image = new Bitmap(filePath);
17              Scale = scale;
18              FindBackgroundColor();
19          }
```

Рисунок 5.3 - Конструктор класу ImageWithShapes

```
20          #region Малювання декартової системи координат
21          public void DrawCoordinateSystem()
22          {
23              using (Graphics g = Graphics.FromImage(Image))
24              {
25                  Pen pen = new Pen(Color.Black);
26                  g.DrawLine(pen, 0, Image.Height - 1,
27                      Image.Width - 1, Image.Height - 1);
28                  g.DrawLine(pen, 0, 0, 0, Image.Height);
29              }
30          }
31          #endregion
```

Рисунок 5.4 - Блок малювання декартової системи координат на зображенні

```
32          #region Отримання кольору пікселів
33          public Color GetPixelColor(Point point)
34          {
35              if (point.X < 0 || point.X >= Image.Width
36                  || point.Y < 0 || point.Y >= Image.Height)
37                  throw new ArgumentOutOfRangeException("Point " +
38                      "is out of image bounds.");
39              return Image.GetPixel(point.X, point.Y);
40          }
41          #endregion
```

Рисунок 5.5 - Блок отримання кольору пікселя зображення за вказаними координатами точки

```
42          #region Пошук кольору фону
43          public void FindBackgroundColor()
44          {
45              Dictionary<Color, int> colorAreas = new Dictionary<Color, int>();
46
47              for (int y = 0; y < Image.Height; y++)
48              {
49                  for (int x = 0; x < Image.Width; x++)
50                  {
51                      Color pixelColor = Image.GetPixel(x, y);
52                      if (pixelColor != Color.Transparent)
53                      {
54                          if (!colorAreas.ContainsKey(pixelColor))
55                          {
56                              colorAreas.Add(pixelColor, 1);
57                          }
58                          else
59                          {
60                              colorAreas[pixelColor]++;
61                          }
62                      }
63                  }
64              }
65              BackgroundColor = colorAreas.OrderByDescending(kv => kv.Value).First().Key;
66          }
67
68          public bool IsBackgroundColor(Color color)
69          {
70              return color.ToArgb() == BackgroundColor.ToArgb();
71          }
72          #endregion
```

Рисунок 5.6 - Блок знаходження фонового кольору зображення та перевірки, чи є заданий колір фоновим

5.2 Клас ImageForm

Програмна реалізація класу ImageForm

Загальна структура класу(рис. 5.7):

```
1   using System;
2   using System.Drawing;
3   using System.Windows.Forms;
4
5   namespace DiplomBool
6   {
7       public partial class ImageForm : Form
8       {
9           private ImageWithShapes imageWithShapes;
10          private Point center;
11          private double radius;
12          Відображення зображення
27          private void ImageForm_Paint(object sender, PaintEventArgs e)
28          {
29              e.Graphics.DrawImage(imageWithShapes.Image, 0, 0);
30          }
31          Обробка події кліку мишею на формі зображення
57
58          private void ImageForm_FormClosed(object sender, FormClosedEventArgs e)...
62      }
63  }
```

Рисунок 5.7 – Клас Image Form

Блок відображення зображення(рис. 5.8) відповідає за створення нового вікна (форми) для відображення зображення та ініціалізацію необхідних подій та обробників подій для цієї форми.

1. `InitializeComponent();`: Цей метод викликається для ініціалізації компонентів форми, які були створені за допомогою редактора форм Visual Studio. Він виконує налаштування компонентів форми, таких як розміщення елементів керування.

2. `imageWithShapes = new ImageWithShapes(filePath);`: Створення нового екземпляру класу `ImageWithShapes` за допомогою переданого шляху до файлу зображення. Цей клас використовується для завантаження зображення, обробки його формату та додавання до нього різних графічних елементів.

3. `imageWithShapes.DrawCoordinateSystem();`: Виклик методу `DrawCoordinateSystem()` з екземпляру класу `ImageWithShapes`. Цей метод відповідає за малювання координатної системи на зображенні.

4. `this.ClientSize = new Size(imageWithShapes.Image.Width, imageWithShapes.Image.Height);`: Встановлення розміру клієнтської області форми на основі розміру зображення.

5. `this.Paint += ImageForm_Paint;`: Підписка на подію `Paint` форми, яка відбувається під час перетворення зображення для відображення на формі. Встановлення обробника події `ImageForm_Paint`.

6. `this.MouseClick += ImageForm_MouseClick;`: Підписка на подію `MouseClick`, яка відбувається при клацанні мишею на формі. Встановлення обробника події `ImageForm_MouseClick`.

7. `this.FormClosed += ImageForm_FormClosed;`: Підписка на подію `FormClosed`, яка відбувається при закритті форми. Встановлення обробника події `ImageForm_FormClosed`.

Метод малювання зображення на формі(рис. 5.9) відповідає за подію малювання зображення на формі.

1. `private void ImageForm_Paint(object sender, PaintEventArgs e)`: Цей метод викликається при події малювання (Paint) на формі зображення.

2. `e.Graphics.DrawImage(imageWithShapes.Image, 0, 0);`: Використовуючи об'єкт `e.Graphics`, який надає доступ до області малювання, малює зображення `imageWithShapes.Image` у верхньому лівому куті форми (починаючи з координат (0, 0)).

Блок обробки події кліку мишею на формі зображення відповідає за обробку події кліку мишею на формі зображення.

1. `Color pixelColor = imageWithShapes.GetPixelColor(e.Location);`: Отримання кольору пікселя на місці кліку миші за допомогою методу `GetPixelColor` об'єкта `imageWithShapes`. Параметр `e.Location` містить координати місця кліку миші.

2. `bool isBackground = imageWithShapes.IsBackgroundColor(pixelColor);`: Визначення, чи є отриманий колір фоновим кольором зображення. Для цього використовується метод `IsBackgroundColor` об'єкта `imageWithShapes`.

3. `int xCoord = e.Location.X / imageWithShapes.Scale;`: Обчислення координати X в декартовій системі координат на основі координати X миші, розділеної на масштаб `Scale`.

4. `int yCoord = (imageWithShapes.Image.Height - e.Location.Y) / imageWithShapes.Scale;`: Обчислення координати Y в декартовій системі координат на основі координати Y миші, переведеної у систему координат зверху вниз, та розділеної на масштаб `Scale`.

5. `int sum = 0;`: Ініціалізація змінної `sum`, яка буде містити суму координат X та Y у випадку, якщо кліку не відбулося на фоновому кольорі.

6. `if (!isBackground) { sum = xCoord + yCoord; }`: Перевірка, чи кольор пікселя не є фоновим. Якщо це так, то виконується обчислення суми координат X та Y.

7. `MessageBox.Show($"{!isBackground}: X = {e.Location.X}, Y = {e.Location.Y}, Cartesian X = {xCoord}, Cartesian Y = {yCoord}, Sum = {sum}");`: Відображення вікна повідомлення, що містить інформацію про координати місця кліку миші, координати в декартовій системі координат, а також суму координат, якщо клацання не відбулося на фоновому кольорі.

Метод закриття форми і звільнення ресурсів відповідає за закриття форми і звільнення ресурсів.

1. `private void ImageForm_FormClosed(object sender, FormClosedEventArgs e)`: Цей метод викликається при закритті форми.

2. `imageWithShapes.Dispose();`: Викликає метод Dispose для звільнення ресурсів, які були використані об'єктом `imageWithShapes`. Це може

включати вивільнення пам'яті, закриття файлів або ресурсів зображення. Все це робиться для того, щоб уникнути витоку ресурсів і підтримати чистоту пам'яті після закриття форми.

```
12      #region Відображення зображення
13      public ImageForm(string filePath)
14      {
15          InitializeComponent();
16          imageWithShapes = new ImageWithShapes(filePath);
17          imageWithShapes.DrawCoordinateSystem();
18          this.ClientSize = new Size(imageWithShapes.Image.Width,
19              imageWithShapes.Image.Height);
20          this.Paint += ImageForm_Paint;
21          this.MouseClick += ImageForm_MouseClick;
22          this.FormClosed += ImageForm_FormClosed;
23
24          center = imageWithShapes.FindCenter();
25      }
26      #endregion
```

Рисунок 5.8 - Блок відображення зображення

```
27      private void ImageForm_Paint(object sender, PaintEventArgs e)
28      {
29          e.Graphics.DrawImage(imageWithShapes.Image, 0, 0);
30      }
```

Рисунок 5.9 - Метод малювання зображення на формі

```
31          #region Обробка події кліку мишею на формі зображення
32          private void ImageForm_MouseClick(object sender, MouseEventArgs e)
33          {
34              Color pixelColor = imageWithShapes.GetPixelColor(e.Location);
35              bool isBackground = imageWithShapes.IsBackgroundColor(pixelColor);
36              
37              int xCoord = e.Location.X / imageWithShapes.Scale;
38              int yCoord = (imageWithShapes.Image.Height - e.Location.Y)
39                  / imageWithShapes.Scale;
40
41              int sum = 0;
42              if (!isBackground)
43              {
44                  sum = xCoord + yCoord;
45              }
46
47              string message = $"{!isBackground}: X = {e.Location.X},{Environment.NewLine}" +
48                       $"Y = {e.Location.Y},{Environment.NewLine}" +
49                       $"Cartesian X = {xCoord},{Environment.NewLine}" +
50                       $"Cartesian Y = {yCoord},{Environment.NewLine}" +
51                       $"Center: X = {center.X / imageWithShapes.Scale},{Environment.NewLine}" +
52                       $"Y = {center.Y / imageWithShapes.Scale}{Environment.NewLine}";
53
54              CustomMessageBox customMessageBox = new CustomMessageBox(message);
55              customMessageBox.ShowDialog();
56          }
57          #endregion
```

Рисунок 5.10 - Блок обробки події кліку мишею на формі зображення

```
59          private void ImageForm_FormClosed(object sender, FormClosedEventArgs e)
60          {
61              imageWithShapes.Dispose();
62          }
```

Рисунок 5.11 - Метод закриття форми і звільнення ресурсів

5.3 Клас MainForm

Програмна реалізація класу MainForm

Загальна структура класу(рис. 5.12):

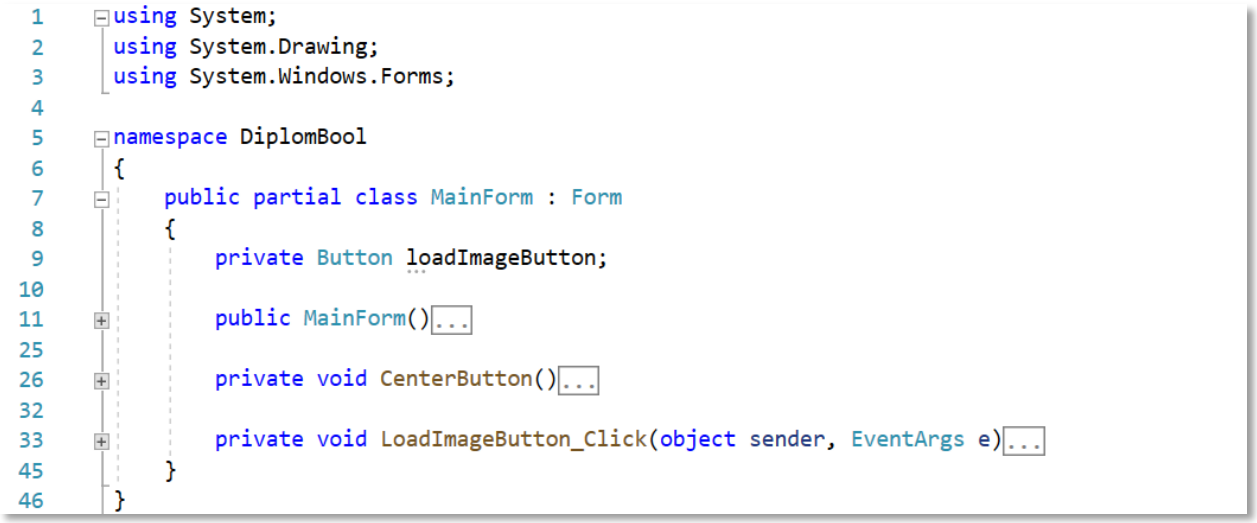


Рисунок 5.12 – Клас MainForm

Конструктор класу(рис.5.13) створює головну форму програми і додає на неї кнопку для завантаження зображення.

1. `public MainForm()`: Це конструктор класу `MainForm`, який викликається при створенні екземпляру цього класу.

2. `InitializeComponent();`: Цей метод ініціалізує всі компоненти форми, які були створені у режимі конструктора форми в різних інтегрованих середовищах розробки, таких як Visual Studio.

3. `loadImageButton = new Button { ... };`: Тут створюється новий об'єкт кнопки з параметрами, вказаними у фігурних дужках. Встановлюється текст кнопки і розташування на формі.

4. `loadImageButton.Click += LoadImageButton_Click;`: Додає обробник події Click для кнопки. При натисканні на кнопку буде викликаний метод `LoadImageButton_Click`.

5. `Controls.Add(loadImageButton);`: Додає кнопку на колекцію елементів управління формою. В результаті кнопка буде відображена на формі для користувача.

Метод знаходження центру форми та вирівнювання кнопки посередині форми(рис. 5.14) призначений для центрування кнопки з назвою "loadImageButton". У ньому використовуються розміри вікна програми (this.ClientSize.Width і this.ClientSize.Height) та розміри кнопки (loadImageButton.Width і loadImageButton.Height). Код обчислює нові координати місцезнаходження кнопки на екрані. Він розраховує положення кнопки по горизонталі, віднімаючи половину ширини кнопки від ширини вікна програми, а потім розраховує положення кнопки по вертикалі, віднімаючи половину висоти кнопки від висоти вікна програми. Таким чином, після виклику цього методу кнопка буде центруватися відносно вікна програми.

Метод обробки події натискання на кнопку(рис. 5.15) відповідає за обробку події натискання на кнопку "Load Image".

1. `private void LoadImageButton_Click(object sender, EventArgs e)`: Це обробник події Click для кнопки "Load Image". Він викликається при натисканні на цю кнопку.

2. `using (OpenFileDialog openFileDialog = new OpenFileDialog())`: Створюється новий об'єкт `OpenFileDialog`, який дозволяє користувачеві вибрати файл для відкриття. Оператор `using` використовується для автоматичного звільнення ресурсів, пов'язаних з `OpenFileDialog`, після завершення його використання.

3. `if (openFileDialog.ShowDialog() == DialogResult.OK)`: Показує вікно діалогу для вибору файлу. Якщо користувач вибрав файл і натиснув кнопку "OK", тоді продовжується виконання коду всередині цього блоку.

4. `string filePath = openFileDialog.FileName;`: Отримується шлях до вибраного файлу.

5. `ImageForm imageForm = new ImageForm(filePath);`: Створюється новий екземпляр форми `ImageForm`, передаючи йому шлях до обраного зображення.

6. `imageForm.Show();`: Відображається форма `ImageForm` для користувача з обраним зображенням.

```
11      public MainForm()
12      {
13          InitializeComponent();
14
15          loadImageButton = new Button
16          {
17              Text = "Load Image",
18              AutoSize = true,
19              Font = new Font("Arial", 14)
20          };
21          loadImageButton.Click += LoadImageButton_Click;
22          Controls.Add(loadImageButton);
23          CenterButton();
24      }
```

Рисунок 5.13 - Конструктор класу MainForm

```
26      private void CenterButton()
27      {
28          loadImageButton.Location = new Point(
29              (this.ClientSize.Width - loadImageButton.Width) / 2,
30              (this.ClientSize.Height - loadImageButton.Height) / 2);
31      }
```

Рисунок 5.14 - Метод знаходження центру форми та вирівнювання кнопки посередині форми

```
33      private void LoadImageButton_Click(object sender, EventArgs e)
34      {
35          using (OpenFileDialog openFileDialog = new OpenFileDialog())
36          {
37              if (openFileDialog.ShowDialog() == DialogResult.OK)
38              {
39                  string filePath = openFileDialog.FileName;
40                  ImageForm imageForm = new ImageForm(filePath);
41                  imageForm.Show();
42              }
43          }
44      }
```

Рисунок 5.15 - Метод обробки події натискання на кнопку “Load Image”

5.4 Клас CustomMessageBox

Програмна реалізація класу CustomMessageBox

Загальна структура класу(рис. 5.16):

```
1   using System;
2   using System.Windows.Forms;
3
4   namespace DiplomBool
5   {
6       public partial class CustomMessageBox : Form
7       {
8           public CustomMessageBox(string message)
9           {
10              InitializeComponent();
11              textBoxMessage.Text = message;
12              textBoxMessage.ReadOnly = true;
13          }
14
15          private void buttonOK_Click(object sender, EventArgs e)
16          {
17              this.Close();
18          }
19      }
20  }
```

Рисунок 5.16 – Клас CustomMessageBox

Цей клас потрібен для створення користувацької форми повідомлення, яка відображає текстове повідомлення і надає користувачеві можливість його закрити, натиснувши кнопку "OK". Він забезпечує більш гнучкий і налаштований спосіб відображення повідомлень, порівняно зі стандартними діалоговими вікнами повідомлень.

5.5 Результат роботи програми:

1. Користувач запускає програму і бачить основне вікно програми, на цьому вікні знаходиться кнопка “Load Image”(рис. 5.17)

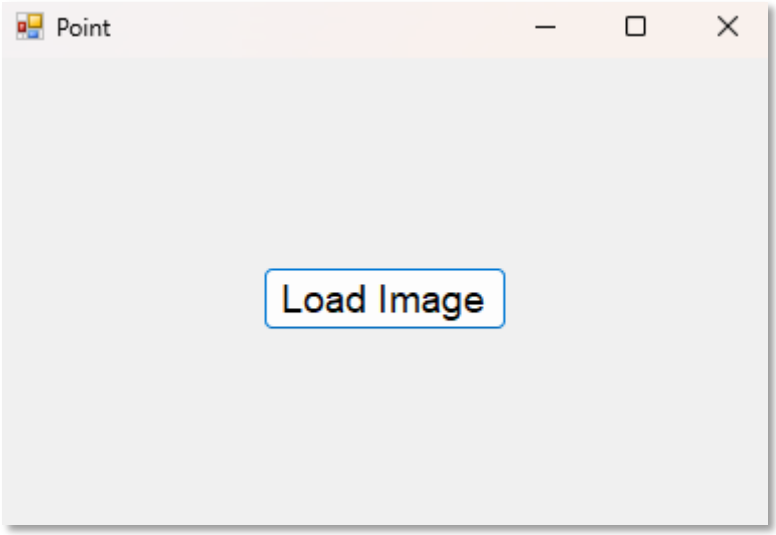


Рисунок 5.17 - Основне вікно програми

2. Користувач натискає на кнопку “Load Image” та з’являється вікно вибору файлу(рис. 5.18), де користувач може обрати зображення на своєму комп’ютері

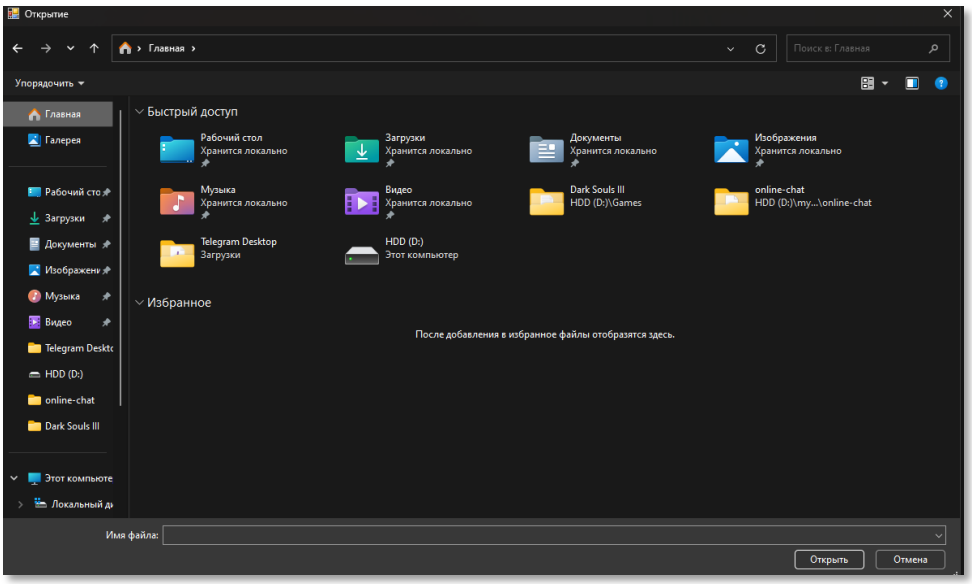


Рисунок 5.18 - Вікно вибору файлу

3. Користувач обирає зображення і натискає кнопку "OK", програма завантажує обране зображення та відображає його на новій формі(рис. 5.19).

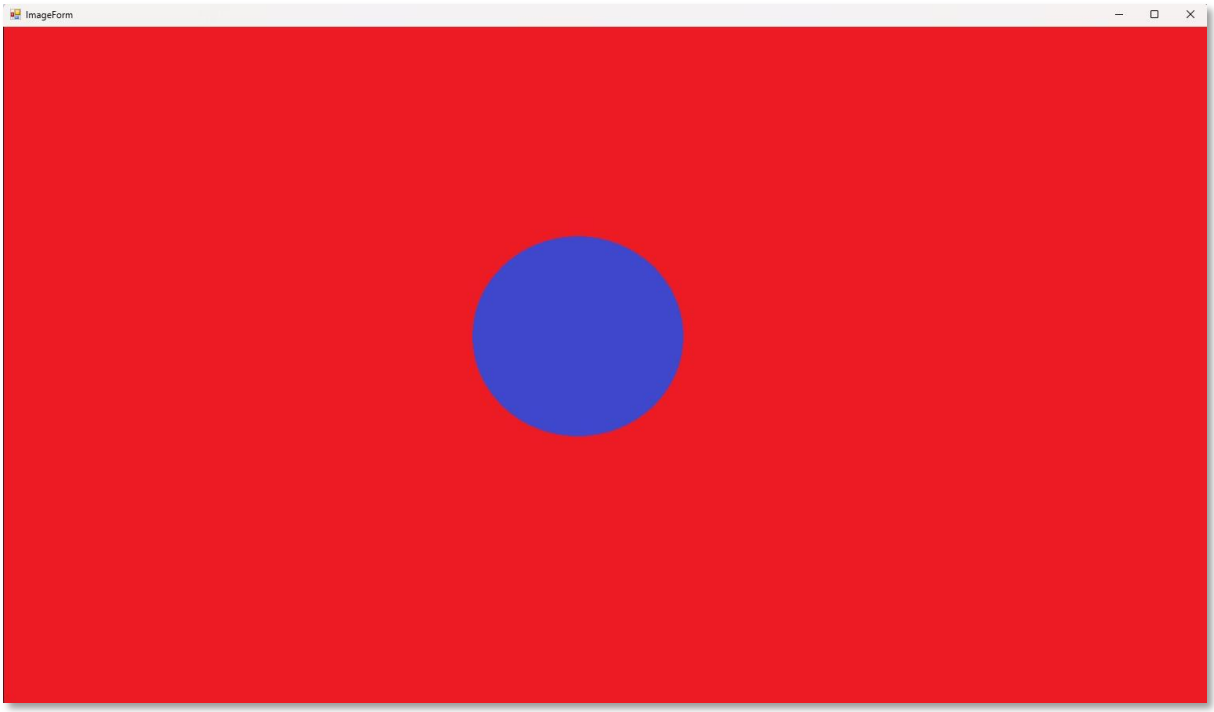


Рисунок 5.19 – Відображення обраного зображення

4. Користувач може натиснути мишею на будь-якій точці зображення. Після кліку на зображенні з'являється вікно сповіщення (CustomMessageBox), яке відображає інформацію про точку, на яку було здійснено клік, координати цієї точки у декартовій системі.

1) Випадок, якщо точка знаходиться не на фігурі(рис. 5.20)

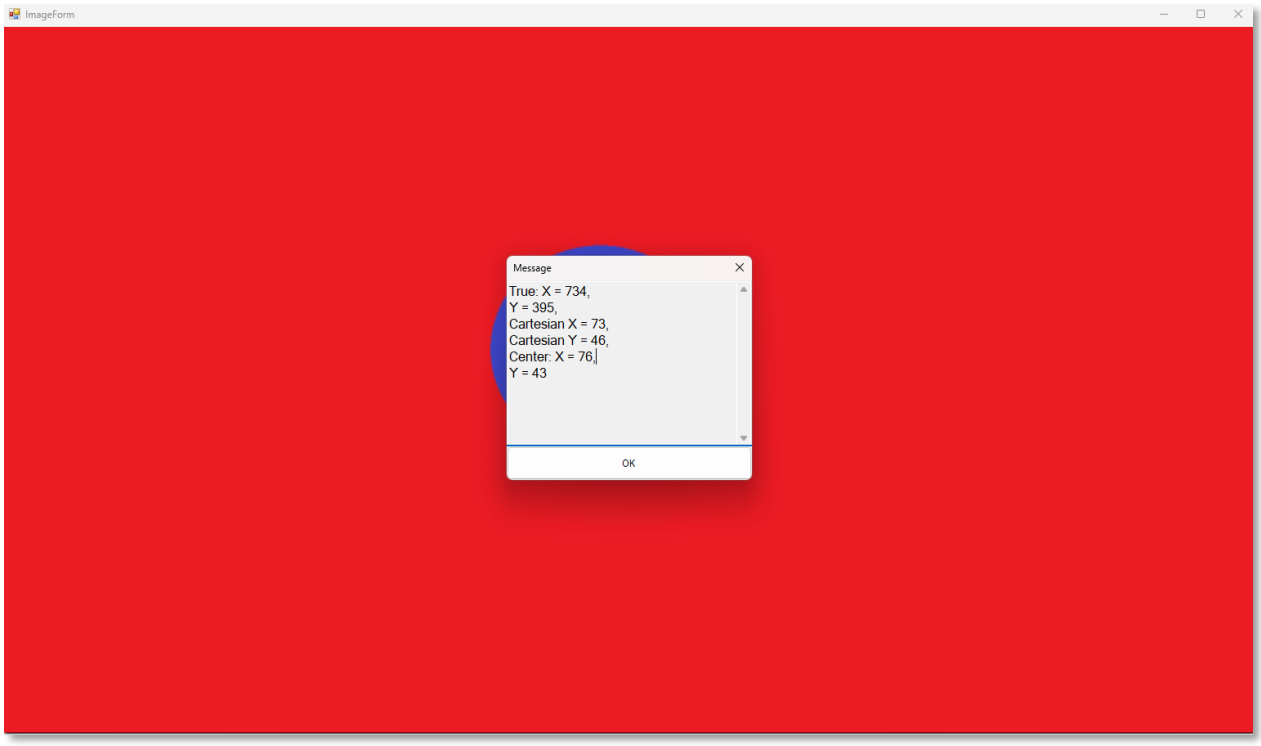


Рисунок 5.20 – Точка не належить фігурі

2) Випадок, якщо точка знаходиться на фігурі(рис. 5.21)

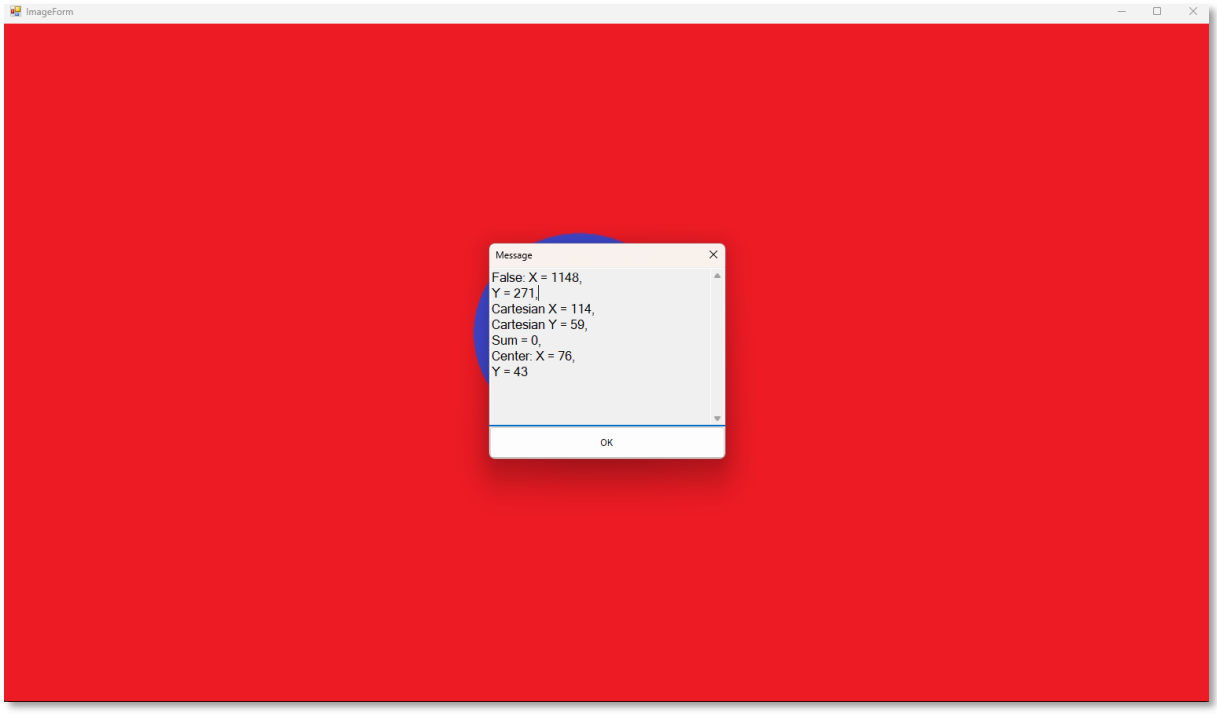


Рисунок 5.21 – Точка належить фігурі

5. Користувач може продовжувати взаємодіяти з програмою, обираючи і натискаючи на різних точках зображення. Після закриття програми зображення і всі інші ресурси, пов'язані з програмою, звільняються з пам'яті.

ВИСНОВКИ

Розробка програми для дослідження можливостей обробки зображень фігур та обчислення їхніх характеристик, таких як статичні моменти та моменти інерції, є важливим кроком у напрямку вирішення задач механіки абсолютно твердого тіла. Дана програма продемонструвала ефективність використання Windows Forms для створення інтуїтивно зрозумілого графічного інтерфейсу користувача, який дозволяє інтерактивно працювати з зображеннями та здійснювати необхідні обчислення у реальному часі.

В рамках цієї роботи було перевірено можливість визначення центрів мас та моментів інерції для плоских фігур довільної форми, які можуть бути задані як аналітично, так і за допомогою графічних файлів. Це дозволило розширити область застосування розроблених методик та алгоритмів на більш широкий спектр задач, включаючи аналіз форм, отриманих з реальних зображень. Успішна реалізація програми підтвердила ефективність чисельних методів для обчислення подвійних інтегралів та інших параметрів фігур з заданою точністю.

Розроблений додаток забезпечує можливість ефективно визначати необхідні характеристики фігур, що суттєво спрощує процес аналізу складних геометричних форм. Крім того, створена гнучка архітектура програмного забезпечення дозволяє легко адаптувати існуючі методи для різних типів плоских фігур та додавати нові функції.

Реалізація даного проекту продемонструвала ефективність застосування сучасних обчислювальних методів та інструментів програмування для вирішення задач механіки та геометрії. Успішна розробка програми не лише створила основу для подальших досліджень, але й може бути використана як навчальний матеріал у курсах об'єктно-орієнтованого програмування та чисельних методів.

Для програмної реалізації завдання на дипломну роботу було застосовано середовище розробки MS Visual Studio та мову програмування C#, що забезпечило високий рівень функціональності та зручності використання. На основі матеріалів даної дипломної роботи можна в подальшому розробити лабораторну роботу з навчальної дисципліни «Об’єктно-орієнтоване програмування» за темою «Класи. Успадкування».

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. І.С.Градштейн, І.М.Рижик Таблиці інтегралів, сум, рядів та добутків. 1971.
2. M.Abramowitz Handbook of mathematical functions with formulas, graphs and mathematical tables. – National Bureau of standards applied mathematics series, 1964. – 832 p.
3. H.B.Dwigth Tables of integrals and other mathematical data. – New York. The MacMillan Company, 1961. – 230 p.
4. М.А. Павловський Теоретична Механіка. – Київ.: «ТЕХНІКА», 2002, – 512 с.
5. Л.М. Березін, С.О. Кошель Теоретична механіка. – «Центр учбової літератури»: – 2020, – 218 с.
6. І. Кузьо Теоретична механіка. – «Фоліо», – 2017, – 780 с.
7. В.М.Булгаков, В.В. Яременко, О.М. Черниш, М.Г.Березовий Теоретична механіка. – «Центр учбової літератури»: – 2021, – 640 с.
8. Цасюк В.В. Теоретична механіка: Навчальний посібник. — Київ: „Центр навчальної літератури”, 2004. — 402 с.
9. Яскілка М.Б. Збірник завдань для розрахунково-графічних робіт з теоретичної механіки. – Посібник. – К.: Вища школа: Веселка, 1999. – 351 с.
10. Коноваленко І. В., Марущак П. О., Савків В. Б. Програмування мовою C# 7.0 : навчальний посібник. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2017. 300 с.
11. Герберт Шилдт. C# 4.0 : Пер. з англ. – ООО «И.Д. Вильямс», 2011. 1056 с.
12. Коротєєва Т. О. Алгоритми та структури даних: навчальний посібник. Львів:Львівська політехніка, 2014. 280 с.

13. Ільман В.М., Іванов О.П., Панік Л.О. Алгоритми, дані і структури: навчальний посібник. Дніпро : Дніпропет. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна, 2019. 134 с.
14. Підручник з мови C# 12 та платформи .Net 8 [Електронний ресурс]- https://metanit.com/sharp/tutorial
15. Документація Visual Studio [Електронний ресурс] - https://learn.microsoft.com/ru-ru/visualstudio/windows/?view=vs-2022&preserve-view=true
16. Документація з мови C# [Електронний ресурс]- https://learn.microsoft.com/ru-ru/dotnet/csharp/